Oracle Fusion Cloud SCM

Modeling Configurations for SCM

24D

Oracle Fusion Cloud SCM
Modeling Configurations for SCM

24D

G12626-01

Author: carl casey

Contents

Get Help

There are a number of ways to learn more about your product and interact with Oracle and other users.

Get Help in the Applications

Use help icons to access help in the application. If you don't see any help icons on your page, click your user image or name in the global header and select Show Help Icons.

Get Support

You can get support at *My Oracle Support*. For accessible support, visit *Oracle Accessibility Learning and Support*.

Get Training

Increase your knowledge of Oracle Cloud by taking courses at *Oracle University*.

Join Our Community

Use *Cloud Customer Connect* to get information from industry experts at Oracle and in the partner community. You can join forums to connect with other customers, post questions, suggest *ideas* for product enhancements, and watch events.

Learn About Accessibility

For information about Oracle's commitment to accessibility, visit the *Oracle Accessibility Program*. Videos included in this guide are provided as a media alternative for text-based topics also available in this guide.

Share Your Feedback

We welcome your feedback about Oracle Applications user assistance. If you need clarification, find an error, or just want to tell us what you found helpful, we'd like to hear from you.

You can email your feedback to *oracle_fusion_applications_help_ww_grp@oracle.com*.

Thanks for helping us improve our user assistance!

1 Overview

What's New in Modeling Configurations

Get a summary about content that's new or significantly revised in each release of Modeling Configurations.

To get details about.

- New features in each update, go to *Cloud Applications Readiness*, then click Order Management What's New.
- Known issues for each update, see *Oracle Supply Chain Management Cloud Functional Known Issues and Maintenance Packs (Doc ID 1563075.1)*.

Update 24B

Topic	Description
Overview of Creating Rules for Configurator	Revised. Get an overview of how your rules use Oracle ACT (Advanced Constraint Technology) and its artificial intelligence to help you solve complex problems. We also revised the Rules chapter so that a wider audience can use it more effectively.

Update 23D

Topic	Description
Get Rule Explanations When You Test Your Model	New. Get details about how your rule affects runtime behavior when you test it in the Configurator Modeling work area.
Release Your Workspace	Revised. See the new Specify What Versions to Update subtopic.
Overview of Creating Rules for Configurator	Revised. Get an overview of how your rules use Oracle ACT (Advanced Constraint Technology) and its artificial intelligence to help you solve complex problems.

Update 23B

Topic	Description
Reduce Start Times When You Have Large Option Classes	New. Enable a placeholder item for an option class, and Oracle Configurator will replace the optional items in the option class with a single placeholder item.
Use Integer Values and Decimal Values in Configurator Rules	New. You must make sure that the result of multiplying integer values in your rule doesn't exceed Java's integer limit.

Update 22A

Topic	Description
Snapshots *Manage Your Workspace*	Revised. We revised the Snapshots chapter and the Workspaces chapter so that a wider audience can use them more effectively.

Update 21D

Topic	Description
Configurator Models	Revised. We revised the Overview chapter so that a wider audience can use it more effectively.

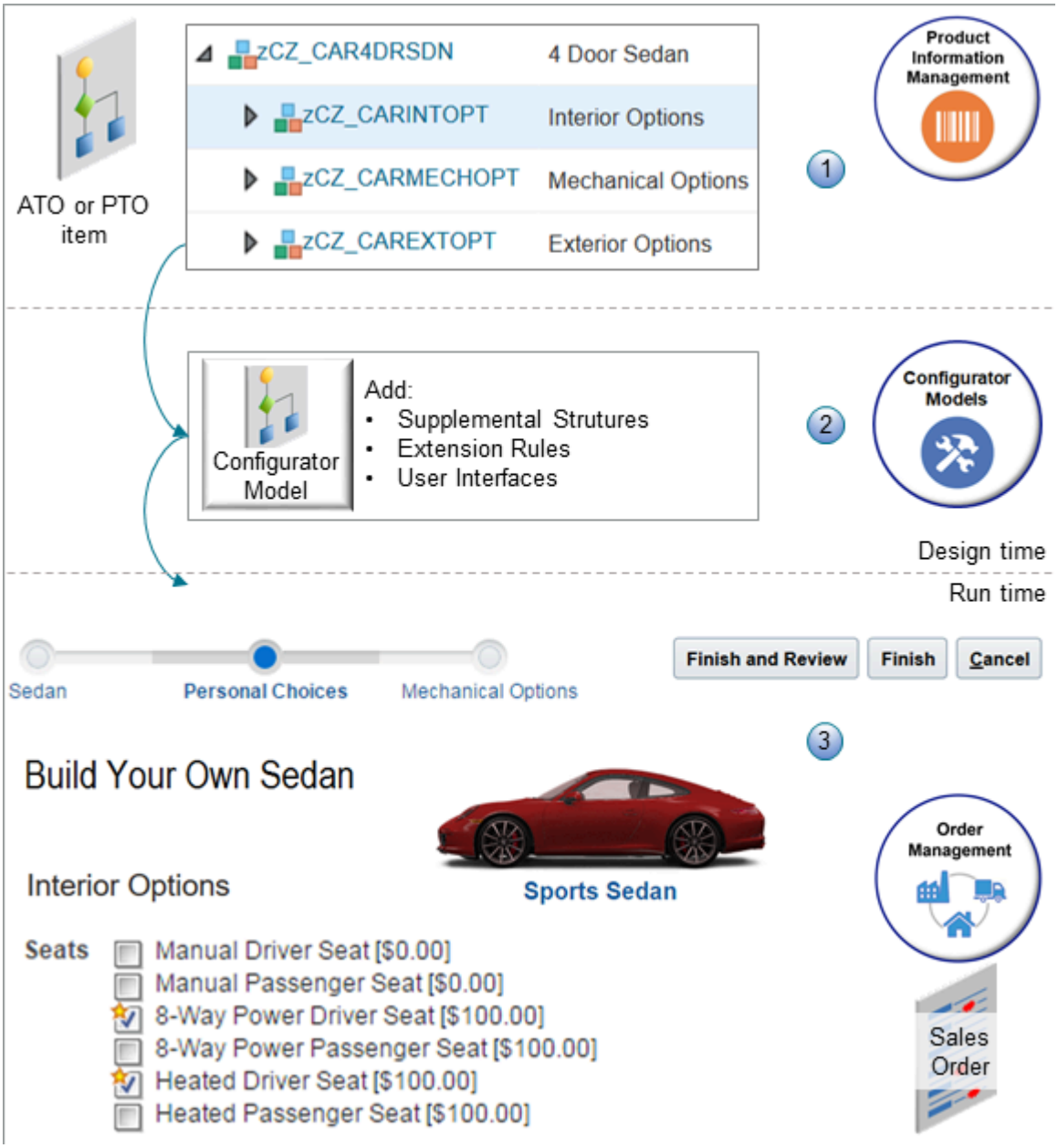
Configurator Models

A configurator model is a hierarchical representation of an item that you import from the Product Information Management work area into the Configurator Models work area. You set up a configurator model to help your users configure the item in your host application.

A model in the Product Information Management work area has all the structures and attributes that are part of the model, including components that aren't optional.

A configurator model in the Configurator Models work area has only the structure for the optional components of the model that you import from Product Information Management. It also has the attributes that are part of that optional structure. A configurator model can also have supplemental structures, rules, and user interfaces that you create in Configurator.

For example, assume you create a Car configurator model, and you add a user interface to it that helps you select the exterior color, interior color, engine, wheels, seats, radio, and so on.



Here's what you can do.

1. Set up a basic model in the Product Information Management work area.

   For details, see *Overview of Configure-to-Order*.
2. Import your model into the Configurator Models work area, then use the Configurator Models work area to add a supplemental structure, rules, and user interfaces to the model.
   - Add a supplemental structure and rules to help simplify life for your users when they configure the item.
   - Use predefined templates to help you create the user interface that your host application displays. Include images, stylized text, various controls for configurable options, and so on.
   - Rapid prototype the user interface to make sure it works as you expect it to, then release the model into your production environment.

3. Use the Configurator user interface in your host application to configure the item so it meets your customer's unique requirements.

   Configurator comes predefined with an integration to these host applications:

   - Oracle Order Management so you can configure an item in a sales order
   - Configure, Price, and Quote Cloud (CPQ) so you can configure an item in a quote in a sales channel

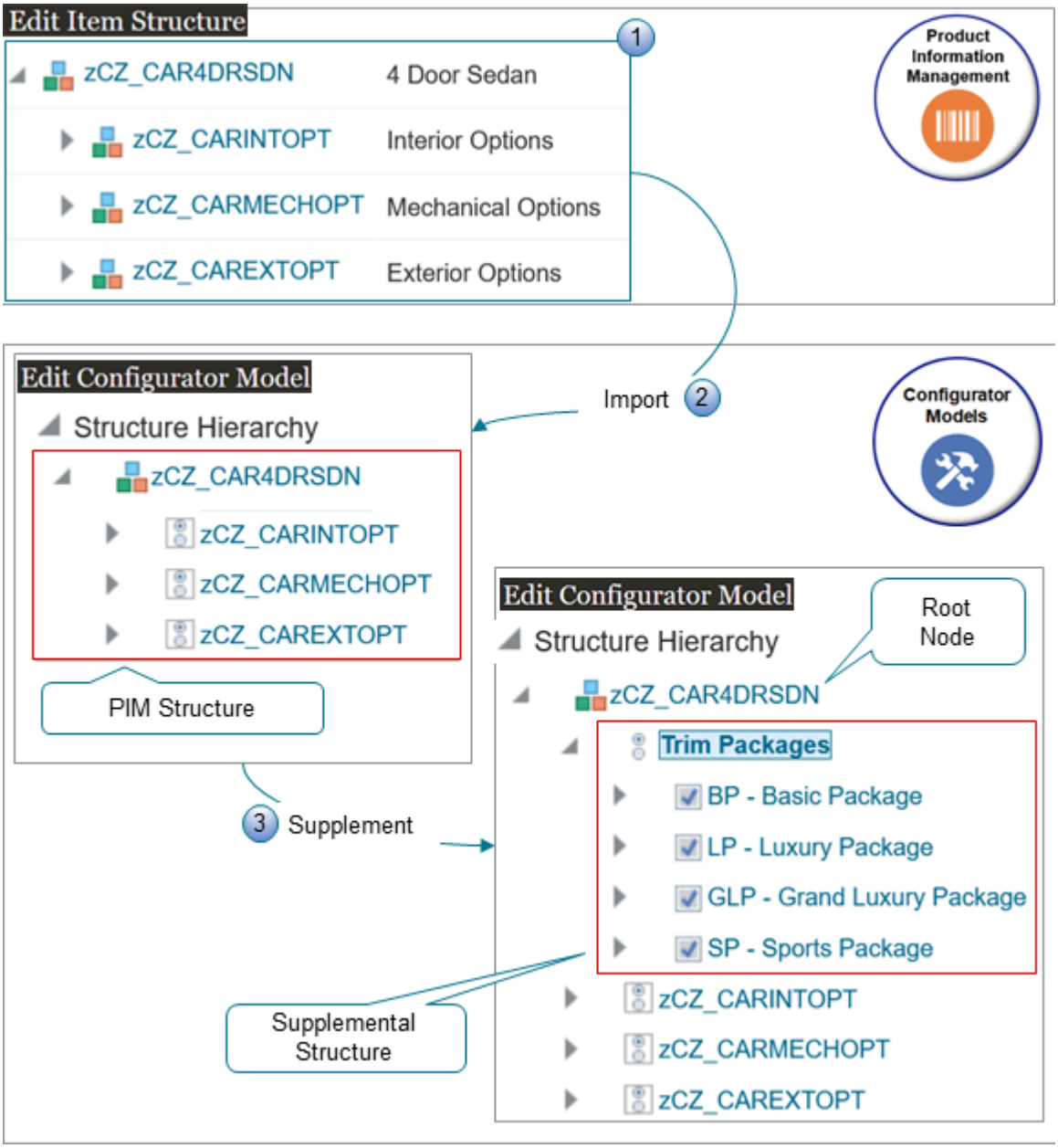
Supplemental Structure

An item that you import from Product Information Management already has a structure, but it might not be the structure that your customer needs, so you can supplement it.

A supplemental structure is a structure that you add to your configurator model after you import the item from the Product Information Management work area into the Configurator Models work area. Use a supplemental structure to:

- Add choices and structure to your model.
- Provide guided questions to help your end-users easily configure the item.
- Use rules to reduce the number of options that the user has to choose in the host application, and to make sure that the choices will work.

Assume you have a model named zCZ_CAR4DRSDN in Product Information Management. Buying a new car can be a daunting process because there are so many options to choose from, such as 20 different exterior colors, 10 different interior colors, 4 types of wheels, 6 different types of seats, 2 different radios, and so on. You can import the zCZ_CAR4DRSDN model into Configurator.



What the Numbers Mean

1. The car already has a hierarchical structure in Product Information Management where you organize these options into option classes. For example, the Seat option class contains six different types of seats, such as Manual Driver Seat, Manual Passenger Seat, 8-Way Power Driver Seat, and so on.

```
4 Door Sedan
 Interior Options
 Seats
 Manual Driver Seat
 Manual Passenger Seat
 8-Way Power Driver Seat
 . . .
 Mechanical Options
 Exterior Options
```

2. You import the zCZ_CAR4DRSDN model into the Configurator Models work area. The import creates the exact same structure in Configurator that you have in Product Information Management (PIM). We refer to this as a PIM structure.
3. Each model in Configurator has a root node. In this example, the root node is zCZ_CAR4DRSDN. You can add a supplemental structure to the root node.

   To simplify life for your users, you can add a supplemental structure named Trim Packages, then add options to it, such as Basic Package, Luxury Package, and so on.

```
4 Door Sedan
 Trim Packages
 BP - Basic Package
 LP - Luxury Package
 GLP - Grand Luxury Package
 SP - Sports Package
 Interior Options
 Mechanical Options
 Exterior Options
```

   Your user can select the package they want at run time.

Rules

You can use the Trim Packages supplemental structure to ask a question and store an answer, but the answer doesn't tell Configurator what items and options to select in reply to the question. Instead, you use a rule to do that.

Assume the Trim Packages structure asks your customer an important question.

- What are you looking for in your car: price, comfort, or performance?

You can create rules that specify the model's behavior in the host application. For example:

- If the customer wants comfort and selects the LP Luxury Package, then automatically select the 8-Way Power Driver Seat option for the seat type, select the Leather option for the seat trim, select the Surround Sound Radio option for the radio, select the Stainless Steel option for the tires, and so on.

The supplemental structure and the rules work together to simplify the configuration and make life easier for your end-users, and for your customers. Rules also help to validate the configuration so it meets your business requirements.

You use the Constraint Definition Language (CDL) in a rules editor to create each rule. For details, see *Overview of Creating Rules for Configurator*.

User Interfaces

You can use user interface templates that come predefined with Configurator to help you create the user interface that Configurator displays in the host application.

- The host application uses templates from the configurator model to dynamically create the user interface at run time.
- Templates determine how the interface looks and behaves, and they allow the end-user to interact with and configure the item.
- You can use a WYSIWYG editor in the Configurator Models work area to modify and test the user interface before you deploy it to your production environment.

- Create more than one user interface for each model. For example, you might need different interfaces for various host applications, languages, geographic areas, and so on.
- Create rules that determine whether to display or hide each element in the user interface according to choices that your user makes, which simplifies and improves the experience.

Assume you sell a model named CN92777 Custom Desktop, and Order Management is the host application.

The Order Entry Specialist is an end-user who uses the Order Management work area to create a sales order. The Order Entry Specialist creates a sales order, searches for the CN92777 item on the catalog line, then clicks **Configure and Add**.

- This example uses the predefined control template, layout template, and shell template.
- Order Management embeds the Configurator user interface directly on the Create Order page of the Order Management work area, and the Order Entry Specialist uses it to configure the item.

- The CN92777 has configurable options that the user can set, such as memory, casing, the processor, and the operating system.
- The Order Entry Specialist finishes the configuration and Order Management adds the item as an order line.
- The user clicks **Submit**, then Order Management uses a validation service to make sure the user selected all the required options, and that the configuration will work. For example, if one of the options is to choose the operating system, then the validation makes sure the configured item includes an operating system because a computer doesn't work without one.

For details, see *Overview of Model User Interfaces*.

Pricing

Configurator integrates with Oracle Pricing to display accurate pricing details in the host application.

- Pricing uses the pricing segment and pricing strategy on the sales order to price the configured item.
- Configurator displays prices and totals for the item in the user interface and on the configuration Review page.
- You use the Pricing Administration work area to set up pricing for a configured item.

Learn how to set up pricing. For details, see *Overview of Oracle Pricing*.

Who Manages a Configurator Model

You do tasks in the Configurator Models work area to create and maintain your models.

Who You Are	What You Do	Why You Do This	How You Do It
Product Expert	Maintain the item in the Product Information Management work area.	Use product data that's current and valid when you create the model.	• Models
Product Configurator Manager	Bring product data from the Product Information Management work area into the Configurator Models work area. Prepare the model so you can implement the configuration behavior that you need.	Use a workspace so you can experiment with and test different ways to model your item.	• Snapshots • Workspaces • Draft Models
• Model Designer • Rules Programmer • UI Designer	Add business logic to the selections that your user makes for the configuration.	• Add supplemental structures so your users can do more complex configurations in the host application. • Add rules to control how your users configure the item. • Add user interfaces to display the model in the most effective way.	• Supplemental structure • Configurator rules • User interfaces
• Product Release Strategist	Make the model available in your production environment.	Put the latest version of your model into production so your users can use it configure the item.	• Workspaces • Release process

Who You Are	What You Do	Why You Do This	How You Do It
• Product Life Cycle Strategist	Maintain the model.	Keep up with or get ahead of changes in your product line as it evolves.	• Versions

More Resources

The book you're reading now has a lot of the detail that you need to create and maintain configurator models, but you can get more to meet your specific needs.

- Get the latest details from Configurator Forum at *https://community.oracle.com/customerconnect/categories/scm-configurator*.
- Go to List of Customer Connect Training Replays at *https://community.oracle.com/customerconnect/discussion/477651/list-of-customer-connect-training-replays*, then view the presentations. Here are some presentations you might find useful.
    - Manage Supplemental Structure using spreadsheets
    - Building Compelling User Experiences using Cloud Configurator
    - Using Statement Rules within the Configurator Cloud
    - Best Practices and Lessons: Implementation Part 1
    - Oracle Business Analytics Reporting Overview for SCM
    - Tips and Tricks for Extending Oracle Applications, Part 1
    - Configurator Cloud Blank Screen Demo
    - Configuring Products and Services Together with Connectors
- Get more:
    - *Configurator Modeling Walk Through (Doc ID 2303991.1)*
    - Oracle CPQ Cloud and Configurator Cloud Integration at *https://community.oracle.com/customerconnect/discussion/45663/oracle-cpq-cloud-and-configurator-cloud-integration*
    - Use Transactional Attributes with Configurator Extensions at https://www.youtube.com/watch?v=7HptkpicOBM.

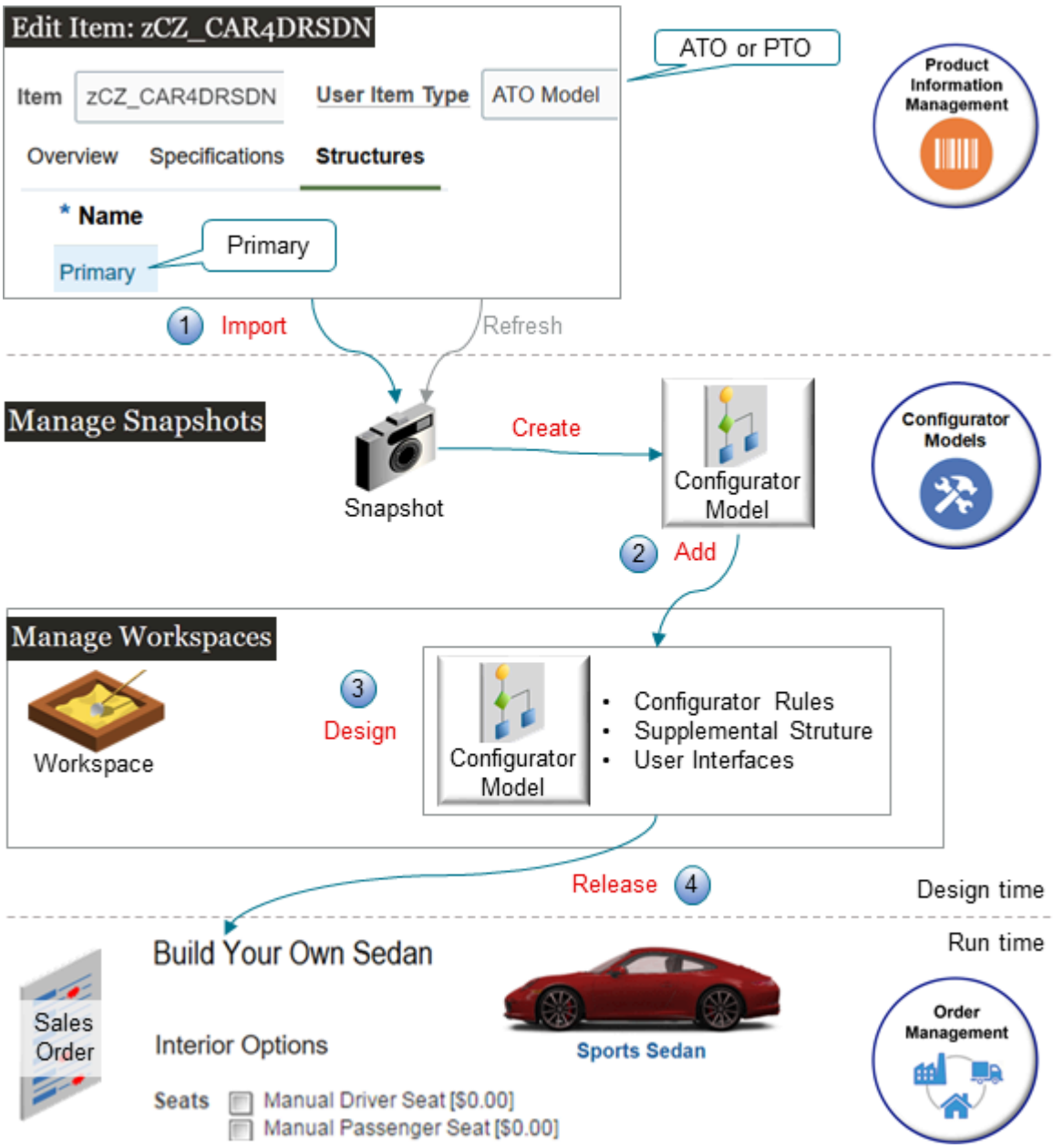
Related Topics

- Snapshots
- Manage Your Workspace
- Overview of Supplementing Your Model

Create and Maintain Configurator Models

You use the Configurator Models work area to create and maintain your configurator model.

Assume you create a model named zCZ_CAR4DRSDN in the Product Information Management work area, and Oracle Order Management is your host application. The end-user can use the model to configure the item when adding it to an order line in a sales order.



What the Numbers Mean

Here's what you do.

1. **Import**. Import a snapshot of the zCZ_CAR4DRSDN from the Product Information Management work area into the Configurator Models work area.
    - Configurator uses the snapshot to automatically create a configurator model for you.
    - If you change anything on the zCZ_CAR4DRSDN in Product Information Management after you import it, then you refresh the snapshot. You don't import the item again.

2. **Add**. Create a workspace and add the zCZ_CAR4DRSDN to it.
3. **Design**. Use the workspace as a kind of sandbox where you can experiment, modify, and test the model. The workspace includes a draft of the model where you can add a supplemental structure, rules, and user interfaces to meet your specific needs.
4. **Release**. Release the workspace to create a new, numbered version of the configurator model and publish it into your host application in a production environment.
    - The released model includes the supplemental structure, rules, and user interfaces that you added to the model.
    - The model goes into effect according to the start date that you specify on the workspace.
    - Users in the host application interact with the model to configure the item. The Configurator makes sure the configuration that you configure is valid.

Summary of the Setup

Here's a high-level summary of the setup that you do to create and maintain a configurator model.

1. Import your model.
2. Create a workspace.
3. Design your configurator model.
4. Release your workspace into production.
5. Maintain your configurator model.

CAUTION: This topic provides a summary. It doesn't include all the stepwise details and other information that you need to create a configurator model. Make sure you read the other chapters of this book when you set up your model.

1. Import Your Model

Import your model into the Configurator Models work area.

1. Sign into Oracle Applications with the privileges from the Product Configurator Manager job role so you can use the Configurator Models work area.

   This topic uses predefined job roles. You must create your own job roles, depending on your security requirements. For details, see *Privileges That You Need to Implement Order Management*.
2. Go to the Configurator Models work area.
3. On the Overview page, click **Tasks > Manage Snapshots**.
4. On the Manage Snapshots page, click **Actions > Import Model Item**.
5. In the Search and Select dialog, search for your item.

Attribute	Value
Item	zCZ_CAR4DRSDN

Note

- The User Item Type attribute on the item in Product Information Management must specify an ATO (assemble-to-order) or PTO (pick-to-order) model. If it doesn't, then the Search and Select dialog won't display it in the search results.
- The item in Product Information Management must have a structure, and it must be the primary structure. If not, you can still click Submit, but the scheduled process will fail with errors.

- You can import an item only one time. If you already imported the item and attempt to import it again, it won't show up when you search for it. The dialog won't display items that you already imported into a snapshot. So, if you make changes to the item in Product Information Management, then refresh your snapshot instead of trying to import it again.
- The import only imports optional children. If a child isn't optional, that means the child is required. The import will still import the parent but your configurator model won't have the required child in the Configurator Models work area.

  You can create child components and make them optional. For details, see these topics.

  - Create Your Configuration Model
  - Determine Whether a Component is Optional
- Click **Advanced > Add Fields** to display more attributes that you can use to refine your search.

6. Click **Submit**.

   Configurator starts a scheduled process, then displays a message. For example:

   ```
   The Import Product Model Item process was submitted with request ID 53718. After the process completes, the related snapshots appear in the search results.
   ```

   The scheduled process imports the item and creates a snapshot of the item's data.

   The request ID identifies the scheduled process.
7. Go to the Scheduled Processes work area and monitor the *Import Product Model Item* scheduled process until its status is Succeeded. Click Refresh, if necessary.
8. Go back to the Configurator Models work area, then search for the new snapshot on the Manage Snapshots page.

Attribute	Value
Name	zCZ_CAR4DRSDN

Note

- The search results will include the item that you imported in a new snapshot.
- If at some earlier time you imported another model that has the same children that you just imported, then the Configurator might or might not refresh them.

Have You Modified the Children in Product Information Management Since You Imported Them?	The Current Import Will
No	Not refresh each child.
Yes	Refresh each child that you modified. Configurator will set the status of the snapshots that contain the refreshed children to Modified.

Have You Modified the Children in Product Information Management Since You Imported Them?	The Current Import Will

Note

- If the item doesn't show up in your search, then go to the Scheduled Processes work area and look for your process, such as 53718. Verify that the Status for the process equals Succeeded.
- The scheduled process also creates a configurator model from the snapshot. You can search for the new model on the Manage Models page. You use the new model as the baseline for any drafts, modifications, or versions that you create.

Configurator creates a baseline version of the zCZ_CAR4DRSDN item. Configurator needs to use this baseline to help manage your model, so you can't modify the baseline. Instead, you add the baseline as a participant to a workspace, then modify the participant to meet your needs.

2. Create a Workspace

Create a workspace that you can use to modify and test your configurator model.

1. Click **Tasks > Manage Workspaces**.
2. On the Manage Workspaces page, click **Actions > Create**.
3. In the Create Workspace dialog, set the values, then click **Save and Close**.

Attribute	Value
Name	zCZ_CAR4DRSDN
Effective Start Date	Set a date that happens in the future. You can't use a date that has already happened. Any change that you make to any object in the workspace will go into effect on the date that you set. You can change the effective start date anytime up until you release the workspace.

Note

- Use the workspace to modify your configurator models, as necessary.
- You can add models to the workspace and edit them, as necessary.
- The workspace uses the In Development status to indicate a workspace that you're editing but haven't released.

4. In the Search Results, click **zCZ_CAR4DRSDN**.
5. On the Workspace page, in the Workspace Participants area, click **Actions > Select and Add > Model**.

   A workspace participant is any object that you add to the workspace.

6. In the Select and Add dialog, search for the name of the item that you created in the Product Information Management work area.

Attribute	Value
Name	zCZ_CAR4DRSDN

   The participant that you add becomes a draft for the next version of the configurator model that you will use in your production environment.
7. Click **Apply > OK**.

3. Design Your Configurator Model

You can add supplemental structures, rules, and user interfaces to your model. You edit a draft of the model in a workspace.

- In the Workspace Participants area, in the Name column, click **zCZ_CAR4DRSDN**.

  The work area displays the Edit Configurator Model page. Examine the layout of this page. Notice that you can access the structures, rules, and user interfaces.
- Use the Structure tab to supplement the structure that you imported from Product Information Management.
- Use the Rules tab to add conditional logic that determines how your users can configure the item.
- Use the User Interfaces tab to create a user interface that meets your item's and your users' specific requirements.
- Click **Test Model** to test the behavior of your draft at any time while you're editing it. This feature uses the test parameters that you specify to open a test session in the Test Model tab.

Consider a model that represents a car.

Part of the Model	Description
Structure	A set of options that model the question: "What kind of driving do you plan to do?" • Local trips • Highway commuting • Off-road camping Note that this kind of question is an option feature, and the response that your user provides is an option.
Rule	A set of rules that helps your user select the kind of driving that their customer does, then choose the engine, suspension, and tires that most effectively supports that kind of driving.
User interface	A user interface that displays the name and photo of each engine, suspension, or tire package that the rules select.

4. Release Your Workspace Into Production

You must release the workspace that contains your draft so Configurator can put the modifications that you made in the workspace into production. The draft modifications take effect in your production environment when the effective start date of the workspace happens.

1. Finish editing your model on the Edit Configurator Model page, then click **Test Model**.

   > **CAUTION:** You must test your model before you release the workspace. Your modifications might have far-ranging effects, you will have to undo the release of the model to fix any errors, and you can undo a release only on a workspace that goes into effect in the future. Use the test to validate that your model works as you expect it to. If possible, have some of your users use your model and incorporate their feedback into the model's design before you release it.

2. Finish editing your model on the Edit Configurator Model page, then click **Save and Close**.
3. Make sure that the effective start date of the workspace is correct. You can change the date right up until you release the workspace.
4. On the Workspace page, click **Release**.

   A scheduled process creates a new version of the model. The Manage Models page displays the new version and the version number. It increments the version number each time you create a new version.

   Your production environment uses the new version. The new version also becomes the baseline for any new draft modifications that you make of the model.

Maintain Your Configurator Model

You can update a configurator model after you release it.

What You Need to Do	How You Do It
Add new functionality to your model.	1. Create a new workspace on the Manage Workspaces page, then add the model to it as a new draft. 2. In the new workspace, add structures, rules, and user interfaces. Test the model and release the workspace.
Reflect changes that you make to the model in the Product Information Management work area.	1. Use the Manage Snapshots page to refresh your snapshots, then make sure that they're in the Modified status. 2. Open the workspace for the item. 3. Add each of your modified snapshots to the workspace as a participant. 4. On the Workspace page, in the Workspace Participants area, click the row that contains your model, then click **Actions > Add Updated Item Snapshots for Models**. The Configurator Models work area will: • Incorporate any changes that you have made to the item in Product Information Management. • Add these changes as a modified snapshot to the workspace. • Update the structure of the model so it reflects the new changes. • Create a new version of the model when you release the workspace.

Determine Whether a Component is Optional

Assume you need to determine whether the Manual Driver Seat item is optional in the 4 Door Sedan model.

1. Sign into Oracle Applications. Make sure you have the privileges that you need to manage items.
2. Go to the Product Information Management work area, then click **Tasks > Manage Items**.
3. Search for, then open the 4 Door Sedan item for editing.
4. On the Edit Item page, click **Structures**.
5. In the Item Structures area, in the Name column, click **Primary**.
6. On the Edit Item Structure page, click **View**, then enable the Component Order Management option.
7. The Manual Drive Seat item is in the CARINTOPT (Car Interior Options) hierarchy, so expand CARINTOPT.
8. The Manual Drive Seat item is in the CARINTSTS (Car Interior Seats) hierarchy, so expand CARINTSTS.
9. Click the row that contains CARINTSTS01 (Manual Driver Seat), then click **Actions > Edit**.
10. In the Edit Components dialog, examine the Optional attribute. If it contains a check mark, then Manual Driver Seat is optional, and you can import it into the Configurator.

Related Topics

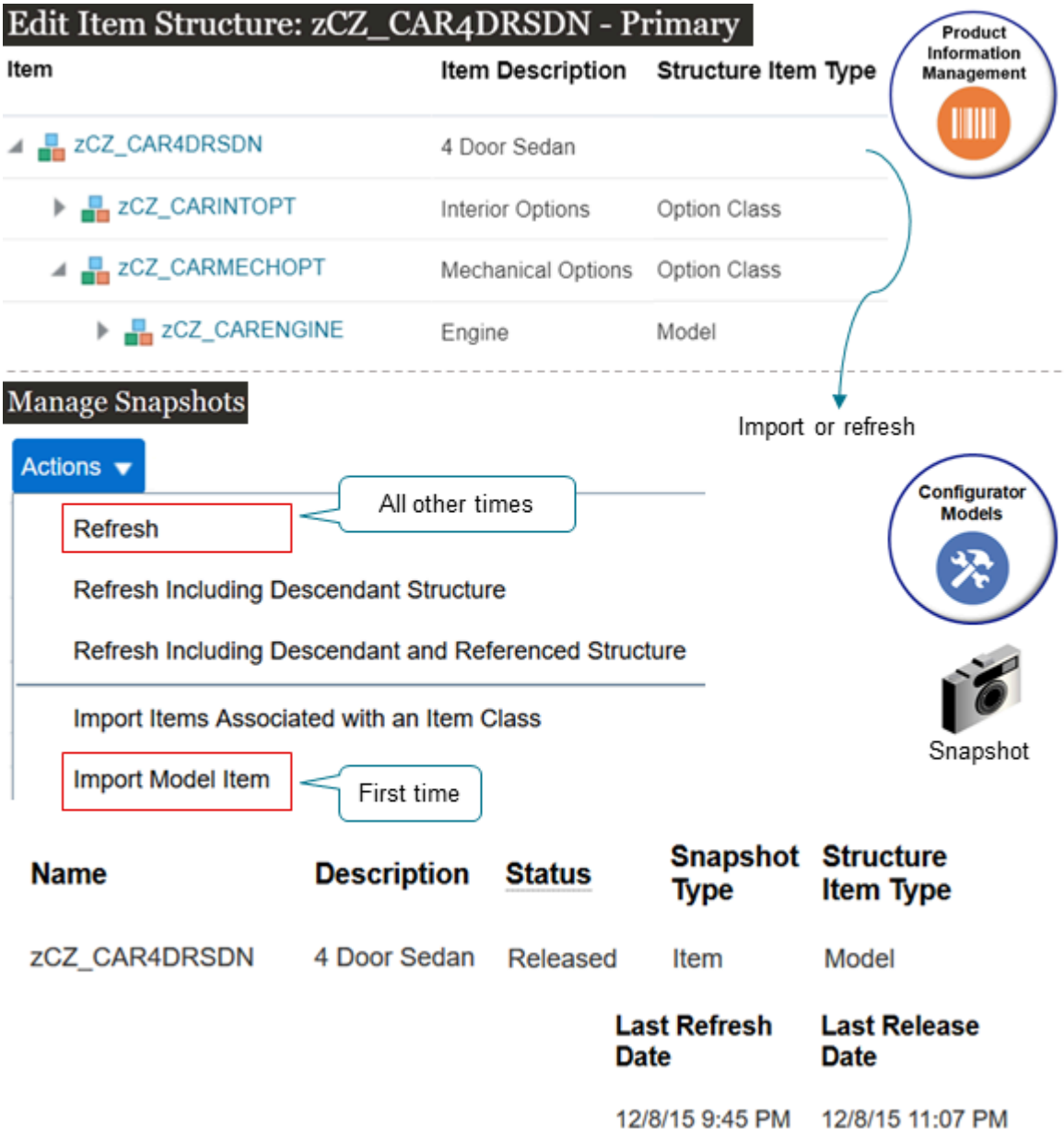
- Work on the Same Participant in Different Workspaces
- Create Statement Rules
- Configurator Models

2 Snapshots

Snapshots

A snapshot is a read-only copy of data that you import from the Product Information Management work area into the Configurator Models work area. It provides a picture of the item, item class, and value set that exists at a point in time.

Consider an example.



Note

- You use the Import Model Item action on the Manage Snapshots page the first time that you import your model from Product Information Management into Configurator.
- You use the Refresh action to bring changes that you've made to your items, item classes, and value sets in Product Information Management into Configurator.
- You can have only one copy of the item that you import in a snapshot. You can't import the same item into the snapshot again, or into another snapshot.

A snapshot has attributes that it imports from Product Information Management.

Attribute	Description
Name	Name or number that identifies the item.
Description	Description of the item.
Organization	Organization that manages the item.
Structure Item Type	Type of item: • Model • Option Class • Standard Item

A snapshot also has attributes that Configurator sets when you import the item into a snapshot.

Attribute	Description
Status	Values are: • **Released**. A snapshot that the host application uses. It is available in your production environment. • **Modified**. Configurator sets this value when you do **Actions > Refresh**. It indicates that there's a refresh and it has changes. If you refresh but there aren't any changes, or if the refresh fails, then the status stays at Released. The host application doesn't see these modifications until you release the snapshot.
Snapshot Type	Values are: • Item • Item Class • Value Set

A snapshot also has attributes that Configurator sets when you refresh the item in a snapshot.

Attribute	Description
Last Refreshed By	User who most recently refreshed the snapshot.
Last Refresh Date	Date when the refresh most recently updated any part of the snapshot.
Last Released By	User who most recently released the workspace that contains the snapshot.
Last Release Date	Date when the release most recently updated any part of the snapshot.

Snapshots and Versions

You use a snapshot with a configurator model, but a snapshot doesn't have a version because you can't modify a snapshot. You do modify the configurator model, so only the model has a version.

- Configurator releases version 0 of the model the first time you import that model from Product Information Management.
- Configurator also releases version 0 of any other referenced models that are in the model's structure.
- Version 0 also references all parts of the model, such as option classes, standard items, item classes, value sets, and so on.

Import Items Into Configurator Models

You import an item from the Product Information Management work area into the Configurator Models work area.

The item that you import must be a model. You can tell because the Structure Item Type attribute for the item equals Model in Product Information Management. For details, see *Configurator Models*.

Results After You Import

Configurator copies some of the attributes of the model and the model's children that you import and stores them in a snapshot.

Item Attributes	Child Attributes
• Description • Item Type • Primary UOM Code • Indivisible Flag • Organization • Serial Generation • Track in Installed Base	• Minimum Quantity • Maximum Quantity • Default Quantity • Optional Children Are Mutually Exclusive • Required When Parent Is Selected • Start Date • End date • Sequence Number

Item Attributes	Child Attributes
	• Instantiation Type • Show in Sales

Note

- A model is a type of item, so the Item Attributes column contains the model's attributes and attributes on the model's child items.
- The Child Attributes column contains attributes for items that are children of a model or an option class.
- Some attributes in Product Information Management don't affect the snapshot. If you modify the value of an attribute in Product Information Management, and if the attribute that you modify isn't part of the import, then your modification doesn't affect the snapshot.
- The import sets the name of the snapshot to the name of the item that you import, and the description of the snapshot to the description of the item that you import.
- If a snapshot of the item's item class doesn't already exist, then the import imports the item class.
- If a snapshot of the parent of this item class doesn't already exist, then the import imports the parent item class.
- A referenced model is a model that another model references. If the model that you import references another model, then the snapshot will also include the referenced model. For example, assume the parent Car model's hierarchy in Product Information Management references a child model named Engine Assembly. The Engine Assembly is a referenced model in the Car model.

Transactional Attributes

If your item in Product Information Management has a transactional attribute, then the transactional attribute for your item is part of the item's item class in Product Information Management.

- The import implicitly imports your item classes and the value sets that your transactional attribute references when you import an item from Product Information Management into Configurator.
- The import creates a snapshot of each value set that's associated with each transactional attribute that's part of the item's item class.
- The value set that you use with the transactional attribute in Product Information Management determines the values that the host application displays. To set the value of a transactional attribute in the host application, the end-user selects an edit control on the item, then selects or enters a value in a dialog for the transactional attribute.

There are some restrictions when you use a transactional attribute with Configurator.

- You can use a transactional attribute only with an Independent, Subset, or Format Only value set. You can't use a transactional attribute with a Dependent, Table, or View Object value set.
- Your set up for a transactional attribute in the hierarchy of the item class in Product Information Management work area can constrain the transactional attribute, but it must not relax an inherited definition. This rule also applies to any modification that you make to a child item class in the hierarchy.

User Defined Attributes

User defined is a bit of a misnomer. Usually, when we say user, we mean end user who does transactions in the host application, such as an Order Entry Specialist who creates a sales order. But here, user defined means you, an administrator who creates an attribute in Product Information Management.

The snapshot only includes the user defined attributes that you specify when you use the Manage Item page in Product Information Management.

If you need to import a user-defined attribute on a different item, you can associate an attribute group in different item classes that:

- Currently have values
- You create at the item level

For details, see *User-Defined Item Attributes and Attribute Groups*.

Related Topics

- Work on the Same Participant in Different Workspaces

Refresh Your Snapshot

Refresh your snapshot to keep your configurator model in the Configurator Models work area synchronized with updates that you make in the Product Information Management work area, such as on the item, the item's item class, or the item's value sets.

Guidelines

- If you import an item, and then update the model in Product Information Management, then you must do a refresh to get the update into Configurator. Don't try to reimport the item. If you try to reimport, then the import won't include the modifications that you made in Product Information Management.
- If you modify the item, item class, or value set, then Configurator sets the snapshot's status to Modified when you do the refresh, indicating that you might need to update the configurator model that uses the snapshot.
- Import only imports children that you specify as optional in the Product Information Management work area. If a child isn't optional, then the import won't import it, and you won't see the child in the Configurator Models work area.
- You can only modify a configurator model that's a draft and that you lock for editing. If you can't edit the model, the it's probably a released version, or it's a draft version but someone else is editing the draft.
- You must add the modified snapshot to each workspace that contains a draft of your model. You can then update and test your model, release the workspace into production, and Configurator will update the snapshot status from Modified to Released. The modifications that you make in Product Information Management go into production only after you release the modified snapshot.
- A configurator model in Configurator doesn't have all the same objects that a model in Product Information Management has. A configurator model in the Configurator Models work area has only the optional components' structure of the model that you import from Product Information Management. It also has the attributes that are part of that optional structure. A configurator model can also have supplemental structures, extension rules, and user interfaces that you create in Configurator.

Try It

1. Go to the Configurator Models work area.
2. On the Overview page, click **Tasks > Manage Snapshots**.
3. Search for your snapshot.

   You can refresh a snapshot for:

   - An item that's a model, option class, or standard item
   - Item class
   - Value set
4. Click **Actions**, then click the type of refresh.

Type of Refresh	Description
Refresh	Brings all the updates that you have made on the item in Product Information Management into the snapshot in the Configurator Models work area. The refresh will update: - Each attribute that you modified since you originally imported the attribute into Configurator. - The item structure's children, including a model's child items, and an option class' child options. The refresh doesn't update a child item's snapshots. It updates only the attributes of the children that are part of the item's snapshot. - The same attributes that you originally imported for child items or options. - Any items that you added to the item's structure in Product Information Management. - Any item class, value set, standard item, model item, or option class that you modified and that's part of the snapshot.
Refresh Including Descendant Structure	Same as for Refresh, but the refresh will also include structures that are children of the model in Product Information Management, such as an option class under an option class. This type of refresh applies only for an item's snapshot. It doesn't refresh child models that the parent references.
Refresh Including Descendant Structure and Referenced Structure	Same as for Refresh, but the refresh will also include all structures that are children of a referenced structure, such as a referenced model.

   A scheduled process starts and displays a message.

   ```
   The refresh snapshot process was submitted with request ID 71901. After the process completes, snapshots
    appear in the search results with updated status.
   ```

   The scheduled process refreshes the snapshot.
5. Click **View > Columns**, then add a check mark to the Last Refresh Date option.
6. Search for your item again in the search area to update the page.
7. Examine the Last Refresh Date option and verify that the date has happened in the last few minutes.

Note that Last Refresh Date reports time according to your user preferences.

8. Verify that the refresh was successful. Go to the Scheduled Processes work area and look for your process, such as 71901. Verify that the Status for the process equals Succeeded. The process name is Refresh Snapshot.

Update Your Descendants

Consider an example.

1. You create an item named AS54888 in Product Information Management as a pick-to-order model. The AS54888 is a desktop computer.
2. You import the AS54888 from Product Information Management into the Configurator Models work area in a snapshot named AS54888 Snapshot, creating a configurator model named AS54888.
3. You do some design work on the AS54888 model in Configurator, then release it into production.
4. You have a marketing campaign that's introducing a new, hot selling feature. New software designed specifically for virtual reality. AS54888 already has an option class named Software that allows users in your host application to choose their own software packages. You can use it to add the new feature. So, you go to the Product Information Management work area, then add the new Virtual Environments item to the Software option class in the AS54888 model.

   But now the AS54888 snapshot in Configurator is no longer synchronized with the AS54888 item in Product Information Management. The AS54888 snapshot in Configurator doesn't have the Virtual Environments item, and your users can't choose Virtual Environments as an option in the host application. They're stuck in a two dimensional world.

Here's how you can fix that.

1. Go to the Configurator Models work area.
2. On the Overview page, click **Tasks > Manage Snapshots**.
3. On the Manage Snapshots page, search for your snapshot.

Attribute	Value
Name	AS54888 Snapshot

4. Click **Actions > Refresh Including Descendant Structure**.

   This action refreshes the model and all of the model's descendants. It finds all of the changes that you made to the model and to the model's children.

   - In this example, it finds that you changed the snapshot for the Software option class.
   - As an alternative, you can search for the Software option class, and refresh it instead of the refreshing the entire model.
   - This action only refreshes models and structures that the model contains. It doesn't refresh models and structures that the model references.
5. Verify the refresh. Verify that:
   - The refresh finished successfully. Examine the scheduled process, if necessary.
   - The Status attribute of the snapshot for the Software option class is Modified.
   - The Software option class contains the new Virtual Environments item.

6. Add the snapshot for the Software option class to a new workspace that contains a draft of the AS54888 configurator model.

   Adding the snapshot will add the latest data from Product Information Management to the AS54888 configurator model, including the Virtual Environments item.
7. Revise the AS54888 model in Configurator to pick up your design changes, as necessary.

   For example, redesign the user interface, revise the supplemental structure, add rules, and so on.
8. Test your AS54888 configurator model.
9. Release the workspace into production, then verify that you can choose Virtual Environments as an option for the Software option class in your host environment.

Did you Modify the Structure's Item Type?

You can't change the item's type in Product Information Management after you import it into the Configurator Models work area. If you change the type and then attempt to refresh the snapshot, the update will fail or you will encounter unexpected behavior when you view or test the model in Configurator. You might also encounter problems when you release the model to production.

Assume your Virtual Environments software has been out for a few years now and has matured into two different market segments, one for homes and one for gardens. You now have two different types of virtual environments to sell to these segments, so you need to create an option class for Virtual Environments and add two options to it:

- Virtual Homes
- Virtual Gardens

Assume

1. You have a model in Product Information Management named AS54888, AS54888 contains the Virtual Environments item, and the Structure Item Type attribute for Virtual Environments is Standard Item.
2. You import AS54888 into Configurator in a snapshot.
3. You change the Structure Item Type attribute from Standard Item to Option Class for Virtual Environments in Product Information Management.
4. You refresh the snapshot for Virtual Environments, but the refresh fails.

To fix this problem:

1. End date the Virtual Environments item that has the Structure Item Type set to Standard Item.

   You can also delete it from the structure, but only if you didn't set up the model to create sales orders.
2. Create a new option class named Virtual Environment, and make sure it has the item type that you need, in this case, Option Class.

   You can't use a name that already exists in Product Information Management regardless of whether the name is for an item or for an option class. To avoid a conflict, change your name slightly, such as Virtual Environment.
3. Add the Virtual Environments option class to AS54888's structure.
4. Add your child options to the Virtual Environment option class:
   - Virtual Homes
   - Virtual Gardens
5. Go to the Configurator Models work area, then:
   - Refresh the model.

    - Update the model's descendants. For details, see the Update Your Descendants section earlier in this topic.

6. Test your revised configurator model and release it.
7. Go to your host application, add the AS54888 to an order line, verify that the AS54888 has a Virtual Environments option class, and that you can set Virtual Environments to Virtual Homes or to Virtual Gardens.

Troubleshoot

Trouble	Shoot
I can't see whether my snapshot data is up to date.	Go to the Edit Configurator Model page or the View Configurator Model page. In the Structure Hierarchy area, click the node that you're interested in, then examine the Snapshot Status attribute. • Released. The snapshot is up to date. • Updates in Workspace. There are updates in the workspace but these updates aren't in the model, and the snapshot is in the model's workspace. • Updates Not in Workspace. There are updates in the workspace but these updates aren't in the model, and the snapshot isn't in the model's workspace. If the snapshot isn't up to date, then go to the Workspace page, then click Actions > Add Updated Item Snapshots for Models.

3 Workspaces

Manage Your Workspace

A workspace is a kind of sandbox that you can use in the Configurator Models work area to modify and test your configurator model.

- Use a workspace to create a new version of your configurator model and to control how and when you release it into your production environment.
- Add a snapshot to your workspace when you have changes in the snapshot that you want to release or that you need to add to the model that you're working on.
- Create a draft model, which is a configurator model that has modifications that you've done in the workspace.
- Use the Manage Model page to add a supplemental structure, rules, or user interfaces to the model, then use Manage Workspaces to create a draft version of a model.
- Use the workspace's Effective Start Date attribute to specify when each change that you make to a draft model goes into effect.
- Release the workspace into your production environment.

Note

- You must import the model, then add it to the workspace before you can modify it.
- You create a draft of the model when you add it to the workspace.
- You can modify the draft model.
- You can only modify a model when its a draft in the workspace.
- If you remove a draft model from the workspace, then you lose all the changes you've made to the draft and you can't recover them.
- You can't delete a workspace after you create it.

Assume you import an item named zCZ_CAR4DRSDN from the Product Information Management work area into the Configurator Models work area. Here's what Configurator does the first time that you import it.

- Creates a new workspace and appends the name of the workspace with `Model Creation Workspace`. For example, if the item's name is zCZ_CAR4DRSDN, then Configurator sets the name of the 0.0 version of this workspace to `zCZ_CAR4DRSDN Model Creation Workspace`. You can't edit this workspace.
- Creates a new snapshot named zCZ_CAR4DRSDN, and adds the zCZ_CAR4DRSDN model to the zCZ_CAR4DRSDN snapshot.
- Releases all parts of the zCZ_CAR4DRSDN model, including option classes, standard items, child models, item classes, and value sets.
- Sets the status of the snapshot to Released.
- Adds the zCZ_CAR4DRSDN snapshot to zCZ_CAR4DRSDN (V1) Model Creation Workspace.
- Sets the status of the workspace to Released.

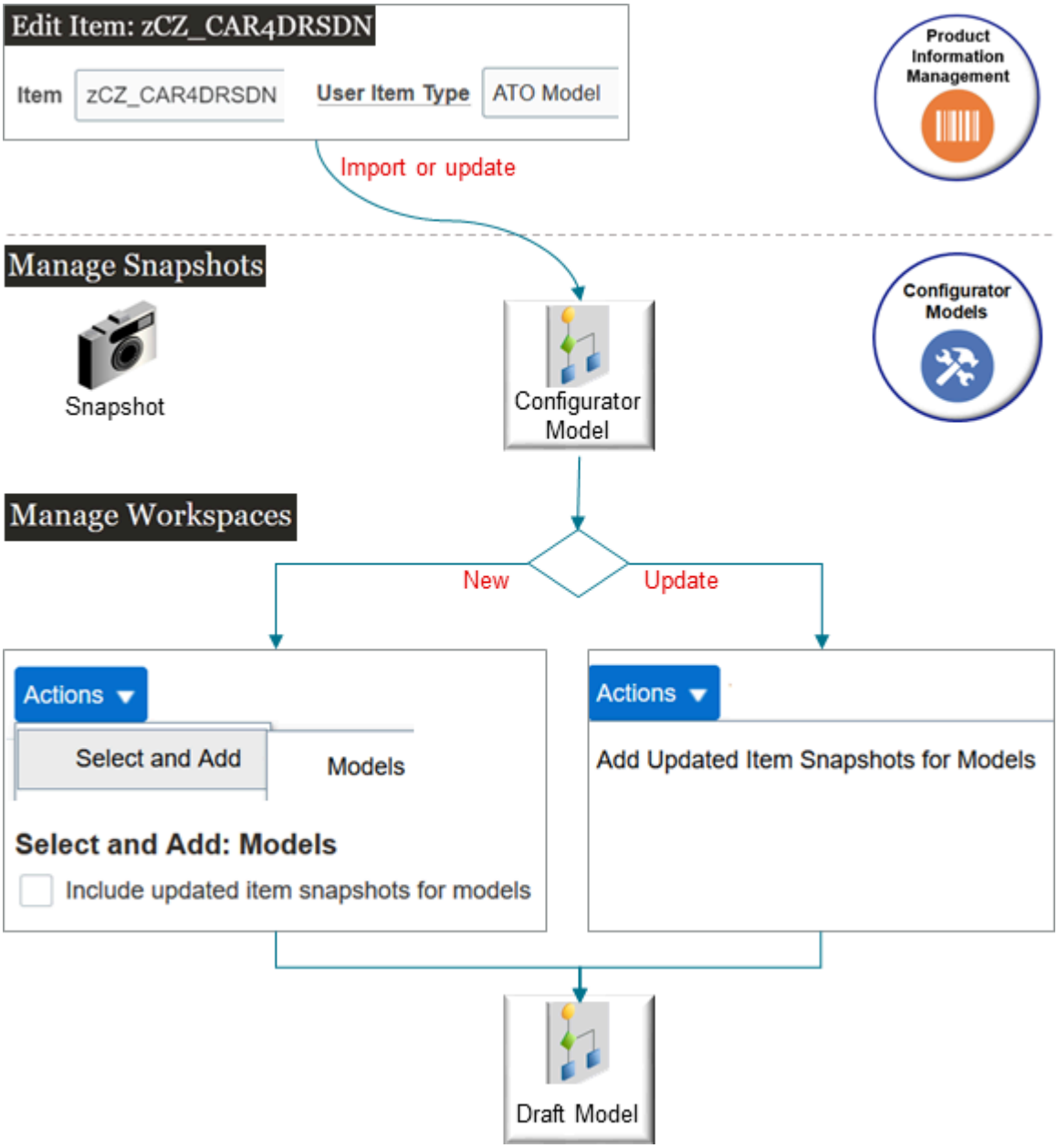
Create Your Workspace and Add Participants

Create a workspace then add participants to it. A participant is any object that you add to a workspace.

Type of Participant	Description
Model	You can add a configurator model from a list of the items that you already imported into a snapshot. • You can add a child model when you add the parent model, or at some later time. • If you only need to test changes that you make to a value set, item class, item snapshot, rule, or user interface, then you still need to add your model to a workspace so you can test your changes in the context of the model.
Item Snapshot	If you add a new item in an option class, then you can refresh its snapshot, add that snapshot to the workspace, then test your work to make sure your host application displays the new item.
Item Class Snapshot	An item gets transactional attributes from its class. If you add a transactional attribute to the item's class in Product Information Management, then you can refresh the item class' snapshot, add that snapshot to your workspace, then test your work to make sure your host application displays the transactional attribute.
Value Set Snapshot	Your transactional attribute might use a value set. For example, the Color value set might have Red, Blue, and Green. If you add a new color such as Magenta to the value set in Product Information Management, then you can refresh the Color snapshot, add that snapshot to your workspace, then test your work to make sure your host application displays Magenta as a choice.

Update Snapshots in Your Workspace

You can update the snapshots that you have in your workspace so that they remain synchronized with changes that you make to the item in Product Information Management.



Try it.

1. Manage the snapshot.
    - Import the zCZ_CAR4DRSDN model from Product Information Management into a snapshot in Configurator.
    - Click **Tasks > Manage Snapshots**.
    - On the Manage Snapshots page, search for your snapshot.

2. Manage the workspace.
    - Click **Tasks > Manage Workspaces**.
    - On the Manage Workspaces page, do one of:

What do You Need to Do?	Description
Add a New Model	Click **Actions > Select and Add > Models**. In the dialog that displays, make sure the Include Updated Item Snapshots for Models option contains a check mark.
Update a Model	Click the row that has the participant you need to update, then click **Actions > Add Updated Item Snapshots for Models**.

Workspace Status

Examine the Status attribute on the Workspace page.

Workspace Status	Description
In Development	You created the workspace but haven't released it.
Released	You released the workspace into production.
In Development, Release Failed	Go to the Scheduled Processes work area, examine the log files for the *Release Workspace* scheduled process, then take corrective action. If you can't fix the problem, create a service request for Oracle Support. Include the process ID of the Release Workspace process that failed in the service request.

You can't manually change the workspace status.

What if More Than One Person Uses My Workspace?

You might have more than one person working in a workspace at the same time. You can lock and unlock each participant to make sure you don't overwrite each other's changes.

Try it.

1. Go to the Configurator Models work area.
2. Click **Tasks > Manage Workspaces**.
3. On the Manage Workspaces page, search for, then open your workspace.
4. On the Workspace page, in the Workspace Participants list, click the row that has the participant you want to lock, then click **Lock**.

Note

- Configurator automatically locks the participant when you add it to the workspace.
- Lock each participant before you edit it.
- Locking prevents someone from changing a draft in the same workspace, but it doesn't prevent someone from changing a draft of the same object in a different workspace.
- Click **Unlock** to unlock the participant.
- You can't lock a workspace.

Develop Models Together as a Group

You can edit and test different drafts of your models together as a group.

- Create a new workspace and add the models that you want to modify.
- Test the runtime behavior of these models as a group.
- Use the same effective start date to release them as a group.

Related Topics

- Manage Your Workspace Dates
- Work on the Same Participant in Different Workspaces
- Release Your Workspace
- Manage Workspace Versions

Release Your Workspace

Release your workspace so you can get your draft from the Configurator Models work area into a production environment, such as Oracle Order Management.

Try it.

1. Go to the Configurator Models work area, then open your workspace.
2. Simulate your release to make sure the workspace doesn't have any errors.

   We recommend that you simulate the release before you actually do the release. Here's how:

   - On the Workspace page, click **Release > Generate Prerelease Report**.

     The report simulates the release without actually releasing the workspace. You can create a report at any time to test your workspace, even if you aren't planning to release right away.
   - In the dialog that displays, note the process ID's value, such as 493400.
   - Go to the Scheduled Processes work area, then monitor the status of the process until it says Succeeded.

Attribute	ParameterValueCode
Name	Workspace Prerelease Report

Attribute	ParameterValueCode
Process ID	493400
Status	Succeeded

- In the search results, click the row that has your process.
- In the Log and Output area, click the **link** next to Attachment, then examine the log file.
- If the process never gets to Succeeded or if it fails, then examine your setup.
  - Make sure the end date for each participant, transactional attribute in the item's class, or value in a value set happens before the workspace's end date.
  - Verify the status of all configurator rules that the workspace uses for the model.

3. Go to the Configurator Models work area, open your workspace, then notice the values.

Attribute	Value
Prerelease Report Process ID	493400 It's the ID of the most recent report that you created.
Status	Contains one of: - **Generating Prerelease Report**. The scheduled process is currently running and creating the report. - **In Development**. The scheduled process is done running.

4. If all looks good, click **Release**.

What Happens When I Release?

Here's what Configurator does when you release your workspace.

- Updates the workspace status from In Development to Released.
- Puts any modifications that you have made to participants into production according to the workspace's effective start date.
- Changes each model that's a participant from a draft to a new version, and sets a new version number for the model.
  Note that you can no longer edit drafts of these participants in this workspace, but you can test a released model in a released workspace.
- Removes all draft participants that you haven't modified from the workspace.
- Writes an impact analysis to the log for the scheduled process that does the release. If there's a problem, you can view suggestions in the log for how to fix it.

Configurator also recompiles each rule that the release affects, including rules in models that aren't in the released workspace. If you make a change in a workspace and release it, and if the change causes a rule to become invalid, then the result depends on whether the model that you release is already in production or is still a draft:

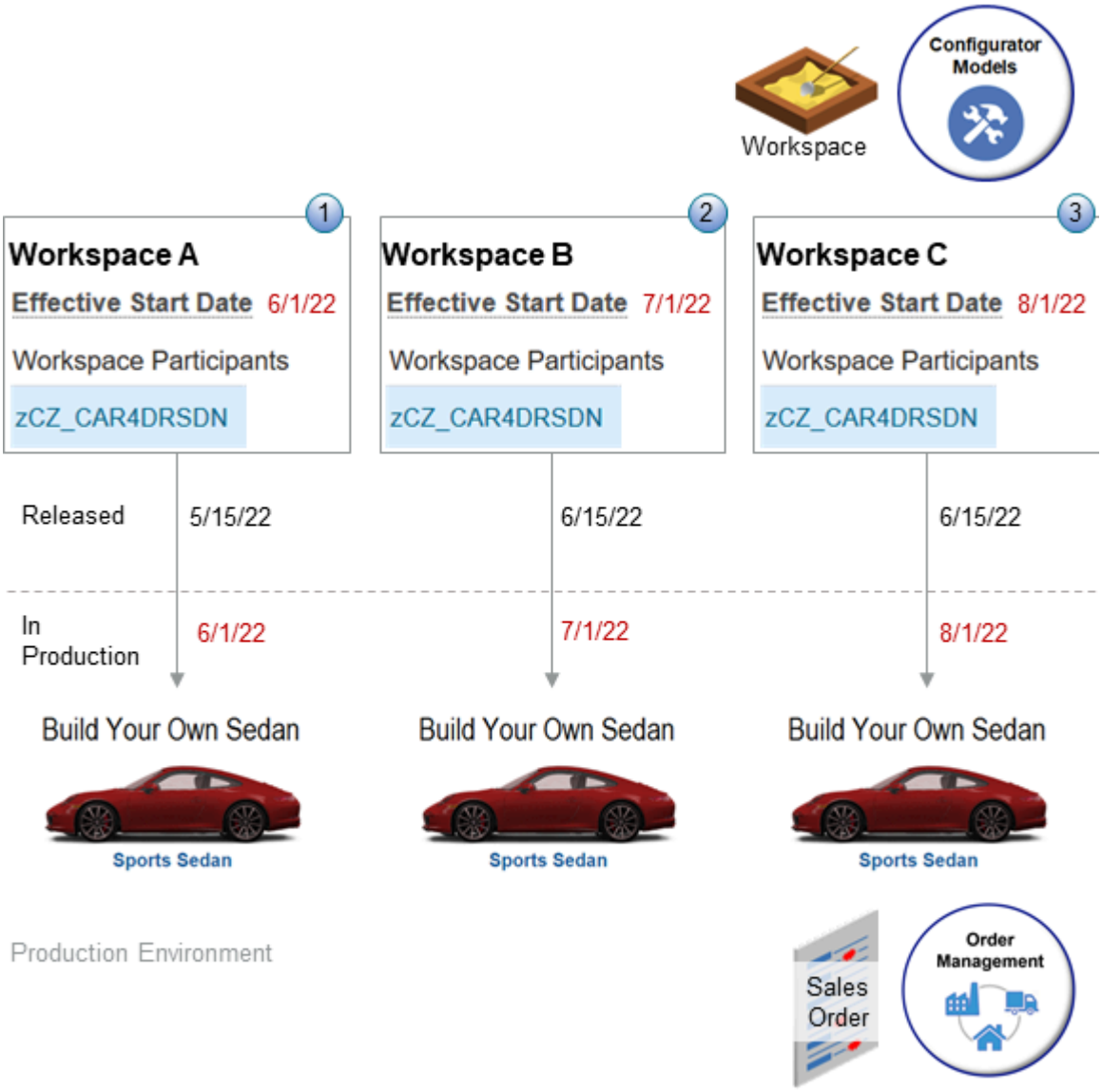
- **In production**. Configurator prevents the release. You must add the model to the workspace and fix the rules.
- **A draft**. Configurator does the release, and adds an entry in the log of the *Release Workspace* scheduled process that identifies the rule that you need to fix.

Note

- You can't modify a workspace's effective start date after you release it, so make sure you release your changes immediately before you need them in production.
- The release includes all participants that exist in the workspace. You can't release only some of the participants in a workspace.
- You can't modify a participant after you release a workspace.

Manage Releases Across Workspaces

Consider an example where you have three workspaces, all of them have the same zCZ_CAR4DRSDN model as a participant, and the model is in draft status in all workspaces.



What the Numbers Mean

Number	You Set The Effective Start Date in Workspace A To	You Release it On	Configurator Puts the Model Into Production On
1	6/1/22	5/15/22	6/1/22
2	7/1/22	6/15/22	7/1/22
3	8/1/22	6/15/22	8/1/22

Note

- Notice how versions of the model move from draft to production after you release the workspace.
- You can release a workspace before its effective start date, but the model doesn't go into production until after the effective start date.
- You can release different workspaces that contain the same model on the same date as long as the effective start dates of these workspaces aren't the same. If you do this, then you must release the workspace that has the effective start date that happens first, and then release the workspace that has the effective start date that happens later.

Releasing an updated model might affect other models that also have the model but that aren't part of the workspace that you release. Assume you release workspace A. If you haven't released workspace B, and if the model's effective start date in workspace B happens after the model's effective start date in workspace A, then Configurator will apply whatever changes that you made to the model in A to the model in B.

Don't Release Your Workspace Until It's Ready

You can set your workspace's effective start date to any date in the future, but we recommend that you release it into production as close to the effective start date as possible. You can't release changes to workspace participants and you can't release future changes until after the effective start date happens.

Assume today is January 1, you release your workspace, and the effective start for the workspace is February 1. This means you released changes that don't go into effect until February 1, and you can't reverse these changes until after February 1. You're stuck with them for a whole month, even though you might need to change them in the interim. To help avoid this problem, Configurator comes predefined to prevent you from releasing a workspace more than 1 day before its effective start date.

But you can modify this behavior to meet your needs. In this example, you set up a parameter to tell Configurator not to release your workspaces until 2 days before the effective start date.

1. Make sure you have the privileges that you need to administer Order Management.
2. Go to the Setup and Maintenance work area, then select the Order Management offering.
3. Search for the Manage Pricing Parameters task, but don't open it.
4. Download your setup data.
    - In the search results, in the row that has Manage Pricing Parameters in the Task column, click the **icon** in the Actions column, then click **Export to CSV File > Create New**.
    - On the Export Setup Data to CSV File page, click **Submit**.
    - On the Setup page, in the search results, in the row that has Manage Pricing Parameters in the Task column, click the **icon** in the Actions column, then click **Actions > Export to CSV File > View All**.
    - On the Export Setup Data to CSV File History page, look at the date to determine the row that has your export, make sure the Status for that row is Completed Successfully, then click **Actions > Download > CSV File Package** on that row.
    - In the dialog that displays, save the file to your computer.

      This file contains the setup parameters. For this example, assume the file name is `Manage Pricing Parameters_20210909_093224_303.zip`.
    - On the Export Setup Data to CSV File History page, click **Done**.
5. Modify the parameter.
    - Use a compression utility, such as WinZip, to extract the file that you downloaded.

- Open the ORA_QP_PRICING_PARAMETER_VALUE.csv file.
- Delete all the rows in the file except for the row that has QP_WS_RELEASE_THRESHOLD in the ORA_QP_PRICING_PARAMETER.ParameterCode column.
- Set the value.

ORA_QP_PRICING_PARAMETER.ParameterCode	ParameterValueCode
QP_WS_RELEASE_THRESHOLD	2 The default value for the parameter is 1, which means 1 day. You can set it to any decimal value that's greater than zero.

- Save the ORA_QP_PRICING_PARAMETER_VALUE.csv file, then add it to the zip file that you downloaded, replacing the original ORA_QP_PRICING_PARAMETER_VALUE.csv file.

6. Upload the file.
   - Go back to the Setup page in the Setup and Maintenance work area.
     In the search results, in the row that has Manage Pricing Parameters in the Task column, click the **icon** in the Actions column, then click **Import from CSV File > Create New**.
   - On the Import Setup Data from CSV File page, click **Browse**, select `Manage Pricing Parameters_20210909_093224_303.zip`, then click **Submit**.
   - In the search results, in the row that has Manage Pricing Parameters in the Task column, click the **icon** in the Actions column, then click **Import from CSV File > View All**.
   - On the Import Setup Data from CSV File History page, make sure the Status attribute contains Completed Successfully. If it doesn't, wait a few minutes, then click **Refresh** until it does.
7. Test your work.
   - Open a workspace that has an effective start date that happens more than two days after today.
   - Click **Release**.
   - Verify that Configurator doesn't allow you to release the workspace but instead displays a message, similar to:

     ```
     You can't release the workspace because its effective start date happens after the latest allowed release date of 01/1/2022 12:00:00 PM.
     ```

For details, see *Updating the Workspace Release Threshold for Oracle Configurator Cloud (Doc ID 2471288.1)* on My Oracle Support.

Specify What Versions to Update

If you have a snapshot that participates in a rule in different versions of a model, then you can use the ORA_QP_RULE_RECOMP_DAYS_PAST parameter to specify which earlier versions of those rules you want to use when you release changes to the snapshot.

ORA_QP_RULE_RECOMP_DAYS_PAST specifies the number of days to look into the past, starting from when you release the snapshot.

Some of the changes that you make to a snapshot are always in effect and might prevent rules in earlier versions from working correctly. If the rules don't work correctly, then you might not be able to release the modified snapshot.

Changing a component's minimum or maximum value or deleting an item are examples of a change that Configurator assumes is always in effect.

Configurator can update rules in your model's earlier versions when you release the modified snapshot that participates in those rules.

If you make a change in your snapshot, and if that change affects your model's behavior in an earlier version, then updating the earlier version will help to make sure your model behaves as you expect it to across versions.

Guidelines

- To specify ORA_QP_RULE_RECOMP_DAYS_PAST, do the same procedure that you do in the Don't Release Your Workspace Until Its Ready subtopic, except modify the ORA_QP_RULE_RECOMP_DAYS_PAST parameter.
- ORA_QP_RULE_RECOMP_DAYS_PAST specifies the number of days in the past to update earlier versions up until your snapshot's release date.
- Specify the number of days before today. If releasing the snapshot affects the rule's version, and if that version is older than the value that you specify in ORA_QP_RULE_RECOMP_DAYS_PAST, then Configurator won't update that version.
- ORA_QP_RULE_RECOMP_DAYS_PAST comes predefined with a default value of 10,000, which means that Configurator will update all prior versions when you release your snapshot.
- If you don't need to validate any sales orders that are older than the current date, then set ORA_QP_RULE_RECOMP_DAYS_PAST to 0. Configurator won't update rules in prior versions when you release your snapshot.
- If you attempt to release a snapshot but it fails with an indication that a rule in an earlier version is failing, and if you don't need to validate runtime configurations on sales orders that you already submitted, then we recommend you set ORA_QP_RULE_RECOMP_DAYS_PAST to 0. Configurator will ignore all earlier versions.
- Configurator updates your rules when you release the snapshot.

Consider the Configuration Effective Date Parameter

If the value of the Configuration Effective Date parameter is:

- **Current Date**. We recommend that you set ORA_QP_RULE_RECOMP_DAYS_PAST to 0. Configurator will ignore all of your model's earlier versions, it will only update versions that are in effect on the current date or in the future.
- **Any value that isn't Current Date**. You must consider how far back into history you want the snapshot changes to affect your rule's behavior. We recommend that you use the default value of 10,000 days for ORA_QP_RULE_RECOMP_DAYS_PAST, or set it to 0.

For details about Configuration Effective Date, see *Manage Your Validations*.

Remove a Model from Production

You can remove a model from production. See *Remove Your Model From Production*.

Related Topics

- Work on the Same Participant in Different Workspaces
- Manage Your Workspace Dates

Manage Your Workspace Dates

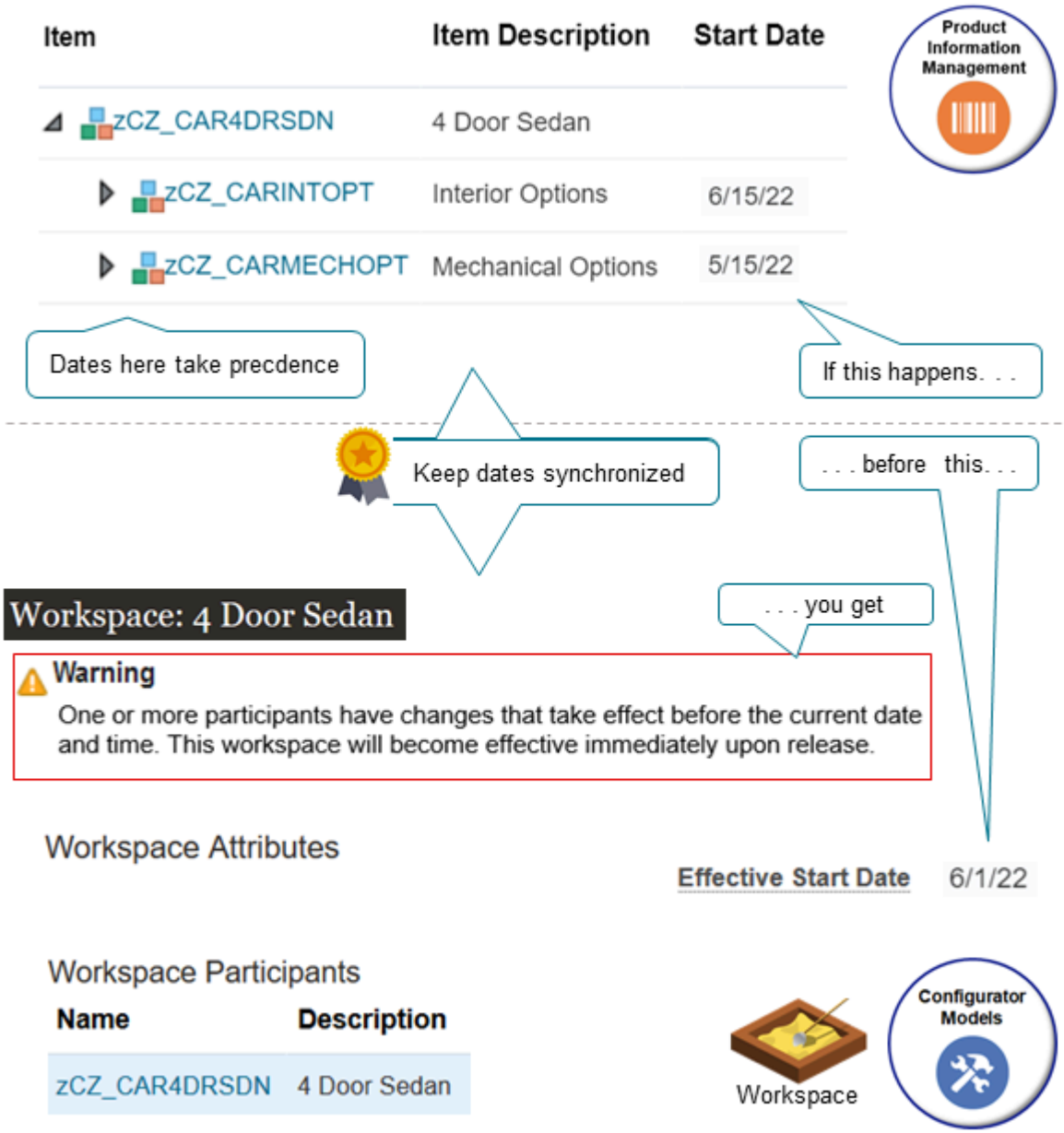
Use these guidelines to help manage dates in a workspace.

The workspace's effective start date determines when the modifications that you make on your model go into production. If you change the effective start date to a later date, and if you release a new latest version of the model before the new effective start date happens, then your draft model will have a different baseline than it did when you added it to the workspace.

Keep Your Dates Synchronized

Make sure you keep the effective start dates that you use in the Configurator Models work area synchronized with the effective start dates that you set for your model, option classes, and components in the Product Information Management work area.

If you add a snapshot that's already in effect to a workspace, then the workspace goes into effect as soon as you release it, regardless of the effective start date that you assign to the workspace. Consider an example.



Note

- You set the workspace's effective date to tomorrow.
- You add a new option class named Interior Options to the model in Product Information Management. The class includes Standard, Deluxe and Sporty so your users can choose the interior trim style.
- You set the effective date on the Interior Options option class in Product Information Management so it goes into effect today.
- You refresh the interior option's snapshot and add it to the workspace, but then encounter a message.

```
One or more participants have changes that take effect before the current date and time. This workspace
 will become effective immediately upon release.
```

- You release the workspace and it goes into production today even though the workspace's effective date is tomorrow. That's because Configurator uses the effective dates from Product Information Management.

Here's another example.

- You have a snapshot in the workspace, the snapshot includes the Interior Options option class, and you set the effective start date on the snapshot to June 15.
- Your workspace also contains another snapshot that includes the Mechanical Options option class, and it has an effective start date of May 15.
- You release the workspace on or before June 10, for example, June 1.
- Your users can't choose the interior options because the Interior Options option class isn't available until June 15, but they can set the mechanical options because Configurator releases the workspace immediately according to the effective start date of the Mechanical Options option class, not the effective start date of the workspace.

Related Topics

- Work on the Same Participant in Different Workspaces
- Release Your Workspace

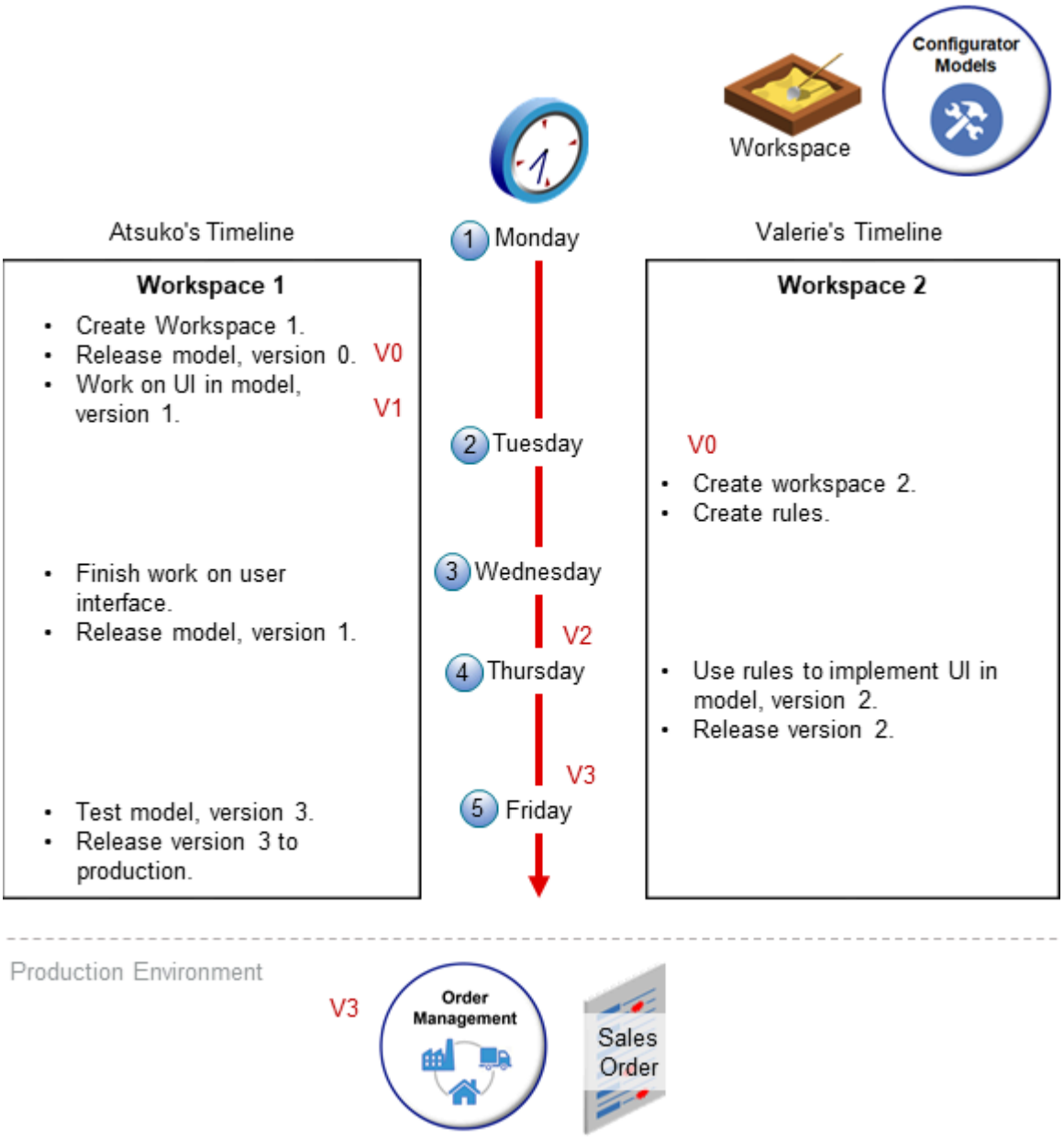
Work on the Same Participant in Different Workspaces

You can work on the same participant in different workspaces at the same time.

For example:

- Two different developers need to work on different parts of a configuration model at the same time.
- You like to use the same participant in more than one workspace to test out different sets of structures, rules, and user interfaces that meet the needs of different market segments, different sets of customers, and so on.

Consider an example where two different developers need to work on the same configuration model at the same time. Atsuko uses workspace 1, and Valerie uses workspace 2.



Here's how it works.

1. Monday. Atsuko imports the 4 Door Sedan model from the Product Information Management work area for the first time.

   The import creates version 0 the model, creates Atsuko's workspace, adds the model to Atsuko's workspace, then immediately releases Atsuko's workspace. Version 0 is an exact copy of the model that Product Information Management has. Atsuko can't modify version 0, but instead modifies version 1 of the model in Atsuko's workspace. Atsuko specializes in the user experience, and starts using workspace 1 to design the user interface that customers will use to configure the 4 Door Sedan.
2. Tuesday. Valerie creates workspace 2, then adds version 0 of the 4 Door Sedan model to workspace 2.

   Valerie is a software engineer and starts adding the rules that the model needs to implement the user interface that Atsuko is creating.

3. Wednesday. Atsuko finishes designing the user interface and releases version 1 of the 4 Door Sedan model.

   Configurator automatically updates the model in Valerie's workspace from version 0 to version 1.
4. Thursday. Valerie uses the rules from Atsuko's workspace to implement the user interface that Atsuko created. This feature helps to make sure that Valerie is using Atsuko's latest work, and avoids having to manually update the workspaces to keep them synchronized.

   Valerie releases version 2, and Configurator updates the model in both workspaces from version 2 to version 3.
5. Friday. Atsuko tests the user interface and makes sure that the interface and rules are working together correctly. Hopefully they do, then Atsuko releases version 3 of the model into production.

Effective Start Date

Make sure the effective start date in your workspaces aren't the same.

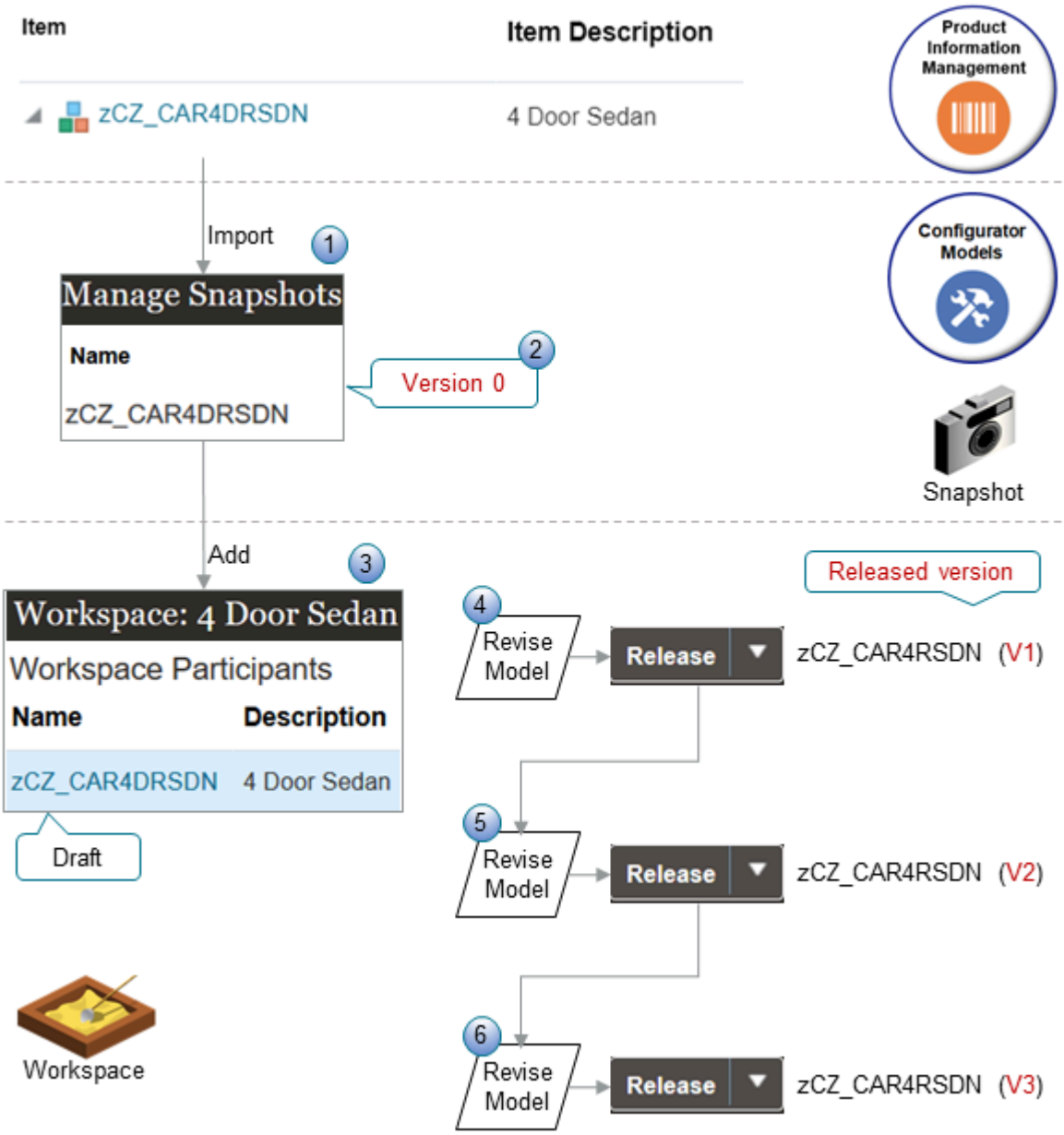
Assume you add the 4 Door Sedan model as a participant x to Atsuko's workspace. If you add participant x to Valerie's workspace, then Configurator displays a message that tells you another workspace already has participant x. If Atsuko's workspace has the same effective start date as Valerie's workspace, then you can't add participant x to Valerie's workspace because it would cause a conflict when both drafts go into effect. You must change the effective start date for Atsuko's workspace. Alternatively, change the effective start date for Valerie's workspace, then add x to Valerie's workspace.

Related Topics

- Manage Your Workspace Dates

Manage Workspace Versions

A configurator model's version is its definition that exists at a specific point in time. If you release a workspace that includes a draft of the model into production, then Configurator creates a new version of the model and increments the model's version number.



What the Numbers Mean

Assume you need to import and work on the zCZ_CAR4DRSDN model.

1. You import the zCZ_CAR4DRSDN from Product Information Management for the first time into a snapshot.
2. The import automatically creates version 0 (zero) of the zCZ_CAR4DRSDN model in the Configurator Models work area.
3. You create a new workspace named 4 Door Sedan and add the zCZ_CAR4DRSDN model to your new workspace.

   From this point forward, you make changes to different versions of the model in the workspace. You can't modify version 0.

4. You use the Edit Configurator Model page to revise the model and save your changes. You click Release on the Workspace page, Configurator releases version 1 of the model, then updates the version of the zCZ_CAR4DRSDN model to version 2.
5. You repeat step 4, except now you're working on version 2, Configurator releases version 2, then updates the model to version 3.
6. You repeat step 4, except now you're working on version 3, Configurator releases version 3, then updates the model to version 4.

   This sequence continues every time you release the model.

Note

- The model remains in Draft status while its a participant in a workspace that's in the In Development status.
- Each version of the model includes the item structure and item attributes that you set up for the item in the Product Information Management work area. It also includes the supplemental structure, configurator rules, and user interfaces that you set up in the Configurator Models work area.
- The version doesn't contain the same data that a copy of the model contains. A version contains only the changes that you make to the draft model since the last time you released the model.
- The changes that you make to the draft are in effect only in your workspace until you release it and when the draft becomes a released version in your production environment. Your draft doesn't become a version until you release it to production.
- The model is the only type of participant that Configurator versions.

Version Zero

Version 0 (zero) is the baseline version. It is an exact copy of the model that you import from Product Information Management. You can't modify version zero, and any change that you make to any higher version won't affect version zero, but you can view version zero and create a new version from version 0. You can use the baseline version to help you manage and troubleshoot your model. If necessary, you can always redo your development work starting over at the baseline version.

Assume you import the zCZ_CAR4DRSDN model into a snapshot, create workspace A, import the zCZ_CAR4DRSDN into it as a snapshot, then release workspace A. zCZ_CAR4DRSDN in workspace A is the baseline version. Next, you create workspace B, and add the zCZ_CAR4DRSDN to workspace B. At this point, you're working on a draft of zCZ_CAR4DRSDN in workspace B, and this draft uses the baseline version as a starting point.

Configurator tracks each change that you make in workspace B to the zCZ_CAR4DRSDN as a change on top of the baseline. Your production environment won't have any of the changes that you make in the draft zCZ_CAR4DRSDN until you release workspace B.

Version Number

Configurator numbers each version that it creates. It increments the version number on each release of the model, and each version is unique for that model. If you release a workspace that has more than one model, then it increments the version for each model independently of the other models. Use the Manage Models page to search for and view any version of a model, including released and draft versions.

Latest Version

Configurator creates a new version of each model that participates in the workspace when you release the workspace.

The latest version of a participant is the version that has the latest effective start date. Its the version that happens most recently in the version's history.

The latest version isn't necessarily the most recently released version because the release date of a version is different from and doesn't depend on the version's effective start date.

The latest version becomes the baseline version for each draft of each model in each workspace that you haven't released, and that also has an effective start date that happens after the latest version. If the effective start date on your draft model happens before the effective start date of the latest version, then the draft won't use the latest version as the baseline. If you want to use the latest version as the baseline, you must change the effective start date of the draft workspace so it happens after the effective start date of the latest version.

Versions for Referenced Models

Configurator versions a referenced model in the same way that it versions a model that isn't referenced.

The scheduled process that you use to release a model automatically examines how the release might affect models that are already in production. Make sure you examine the process log file for any conflicts that the process finds during the release.

Related Topics

- Work on the Same Participant in Different Workspaces
- Release Your Workspace
- Manage Your Workspace Dates

Remove Your Model From Production

You can remove the latest version of your model from production.

You can also do another release to pick up any changes that you need to make. Configurator puts the changes that you've made to participants in the workspace into production, and creates a new version that become the baseline for the next draft. You can use this baseline as a starting point, make your changes, and do another release.

You can't make any changes to the participants or attributes of the workspace after you release it, including the effective start date.

Let's say you release a model into production but then realize it has some errors that you need to correct. Here's how that works.

1. You release model RBG-538 from workspace A into production. The model is at version 2.0. Version 2.0 doesn't have any errors that you need to fix.
2. You update the model, then release it with an effective start date that happens on the first day of next month. This new model is version 3.0.
3. The next day, but before the model's effective start date, you notice that you have an error in version 3.0.
4. You use the *Unrelease Model Participants* scheduled process to remove RBG-538 from production. Here's what the scheduled process does.
    - Removes version 3.0 from production, and places version 2.0 into production.
    - Creates a new workspace, B.
    - Adds version 3.0 to workspace B, and sets the status for B to Draft.
    - version 3.0 in workspace B contains the supplemental structure that you added or modified, rules that you created or modified, and user interfaces that you created or modified.

- Sets the effective start date of workspace B to one minute later than the effective start date of workspace A.

5. You fix the error on version 3 of the RBG-538 model in workspace B.
6. You test and verify your changes, then release version 3 of RBG-538 into production.

Note

- You can repeat the unrelease of a model, one version at a time, to get back to an earlier release version.
- If you have any other drafts of the RBG-538 model, then Configurator updates them so they use the 2.0 version, not the 3.0 version as their baseline. This applies until release version 3.0 into production.
- You can only remove a model from production. You can't remove parts of a model, such as only the supplemental structure.

  The version of the model that you remove from production must have an effective start date that happens in the future.
- You can use the Configurator Models work area or REST API to remove the model from production.

Use the Configurator Models Work Area

Assume you need to remove the RBG-538 model from production.

1. Open the workspace that has the RBG-538.

   The workplace status must be **Released**. If it isn't, then you can't use the Unrelease Models action.
2. Select the RBG-538 model, then click **Unrelease Models**.
3. Configurator starts the Unrelease Model Participants scheduled process.
4. In the dialog that displays, enter a name and description for the new workspace where Configurator will save the model RBG-538.

   Note the process ID of the scheduled process. Assume its 67589.
5. Go to the Scheduled Process work area, then examine the log of the Unrelease Model Participants process that has the 67589 ID. Monitor the process and examine the log file, as necessary.

Use REST API

1. Get the name of the workspace.
   - Go to the Configurator Models work area.
   - On the Overview page, copy the value in the Name attribute of the workspace that contains the RBG-538 model to your clipboard.

     For this example, assume the workspace name is RBG-538-Fall-2020.
2. Identify the released workspace that contains the RBG-538 model.
   - Use the configuratorWorkspaces REST resource and the ByNameOrDescriptionFinder finder to get the workspace ID. For example:

```
https://yourServerName:yourPort/fscmRestApi/resources/11.13.18.05/configuratorWorkspaces?
finder=ByNameOrDescriptionFinder;name=RBG-538 Fall-2020%
```

   where

   - `name` contains the name of your workspace, such as `name=RBG-538 Fall-2020`.

Assume you get this REST response.

```
{
 "items" : [ {
 "WorkspaceId": 300100176467904,
 "Name": "RBG-538 Fall-2020",
 "Description": "RBG-538 early rollout for Fall 2020",
 "EffectiveStartDate": "2020-12-01T00:02:00+00:00",
 "StatusCode": "DEVELOPMENT",
 "Status": "In development",
 "ReleaseProcessId": null,
 "PrereleaseReportProcessId": null,
 "CreationDate": "2019-04-20T00:51:58+00:00",
 "CreatedBy": "PRODUCT_CONFIGURATOR_MANAGER",
 "LastUpdateDate": "2020-10-02T00:52:07.211+00:00",
 "LastUpdatedBy": "PRODUCT_CONFIGURATOR_MANAGER"
 }, {
 ...
 } ],
 ...
}
```

where

- `WorkspaceId` contains the value you seek, such as `"WorkspaceId": 300100176467904`.

3. Identify the modelParticipants resource in the configuratorWorkspaces resource to identify your model. For example:

```
https://yourServerName:yourPort/fscmRestApi/resources/11.13.18.05/
configuratorWorkspaces/300100176467904/child/modelParticipants
```

where

- `300100176467904` is the ID that identifies the workspace in your first REST response.

Here's the REST API response. It includes the participant IDs of the models that are in the workspace.

```
"items": [
 {
 "ParticipantId": 300100217170232,
 "Name": "Model Name ",
...
```

This example has only one model.

4. Run the Unrelease Model Participants scheduled process.

- Use the `unrelease` action of the configuratorWorkspaces resource. You will POST the input payload that contains the parameters that you use to run the process. For example:

```
curl -X POST \
 https://yourServerName:yourPort/fscmRestApi/resources/11.13.18.05/
configuratorWorkspaces/300100176467904/action/unrelease \
 -H 'Authorization: Basic cHJvZF9jb25maWd1cmF0b3JfbWdyOldlbGNvbWUx' \
 -H 'Content-Type: application/vnd.oracle.adf.action+json' \
 -H 'REST-Framework-Version: 7' \
 -d '
 { "targetWorkspaceName": "RBG-538 rework",
 "targetWorkspaceDescription": "Correction of errors in RBG-538 fall 2020",
```

```
 "participants": [ 300100217170232 ]}'
```

where

- `targetWorkspaceName` specifies the name of the new workspace that will contain the RBG-538 model that you remove from production.
- `targetWorkspaceDescription` specifies the description that you want to use for the new workspace.
- `participants` specifies the ID that identifies the RBG-538 model.

Here's the response payload.

```
{
 "result": 145218
}
```

It includes 145218, which is the process ID of the Unrelease Model Participants scheduled process.

5. Go to the Scheduled Process work area, then examine the log of the Unrelease Model Participants process that has the 145218 ID. Monitor the process and examine the log file, as necessary.

   Here's an example log.

```
The unrelease operation from the workspace RBG-538 Fall-2020 completed with warnings.
The workspace RBG-538 rework was created, with participants added from the workspace RBG-538 Fall-2020.
The model participant RBG-538-A-4011 was added from the workspace RBG-538 Fall-2020.
The following workspace participants have warnings:
 Configurator Model Name: RBG-538-A-4011
 The model RBG-538-A-4011 has one or more drafts in other workspaces.
```

For more examples and details, see the Unrelease Changes to a Released Model use case in *REST API for Oracle Supply Chain Management Cloud*.

Related Topics

- Release Your Workspace
- Manage Workspace Versions

4 Structures

Overview

Overview of Supplementing Your Model

Use a supplemental structure to improve your user's experience, such as adding guided selling questions.

PIM Structure

Use a supplemental structure to improve your user's experience, such as adding guided selling questions.

A PIM (Product Information Management) structure is a structure that you use in your configurator model where you import the structure from Product Information Management. Consider these points when you use a PIM structure.

- The Primary structure of a model that you create in the Product Information Management work area determines the structure for your configurator model.
- The structure can include model items, option classes, and standard items.
- Configurator duplicates the item's structure from Product Information Management into Configurator when you import the item.
- The item in Product Information Management is a PTO or ATO model, and it's the root of the structure.

Get details about how to import a PIM structure. See *Create and Maintain Configurator Models*.

Some Reasons to Use Supplemental Structures

- You need to modify an attribute on the item. You can't use Configurator to modify attributes on the items that are in the PIM structure that you import. For example, you can't modify the name or description of an item, attributes on the option class, such as minimum, maximum, or quantity, or user-defined attributes.
- You need to include a required component. The snapshot doesn't include required components. If you specify that a component item in the structure is required in the Product Information Management work area, then the import doesn't import it. Your configurator model won't include these components.
- You need to add or remove a child item. You can't add or remove a child item from a configurator model. Instead, you must modify the structure in Product Information Management, then import or refresh your snapshot.

You can use a supplemental structure to meet these needs while maintaining the PIM structure that you import.

Supplemental Structure

You import your model from Product Information Management into Configurator, then supplement it in Configurator to meet your specific needs.

A PIM structure that you import can include:

- Model items
- Option classes

- Standard items
- Transactional attributes

But these might not meet your needs. Here's what you can add as a supplemental structure in Configurator:

- Option features and options
- Integer features
- Decimal features
- Boolean features
- Text features
- Connectors

Use a supplemental structure when you need to add:

- More choices to your configured item than what you can have on a PIM structure.
- Questions that help your user configure the item at run time.
- Questions that your rules can use to reduce the number of option features at run time.
- Temporary storage for details about your rules.

A complex item, such as a car, might have a lot of options that your customers can choose. Some options might be compatible with each other, while others aren't. You can use a supplemental structure to reduce the number of items that you display at run time.

You can use a supplemental structure only with the configurator model where you add it. For example, you can't add supplemental structure y to model x, and then add structure y to model z. You must recreate structure y on model z.

Related Topics

- Add a Supplemental Structure
- Chunk Your Large Option Classes
- Option Features
- Integer and Decimal Features
- Boolean Features

Set Up Features and Attributes

Option Features

Use an option feature to allow your users to choose one or more options from a list of predefined options. For example, an option feature named Color might have the options Red, Green, and Blue.

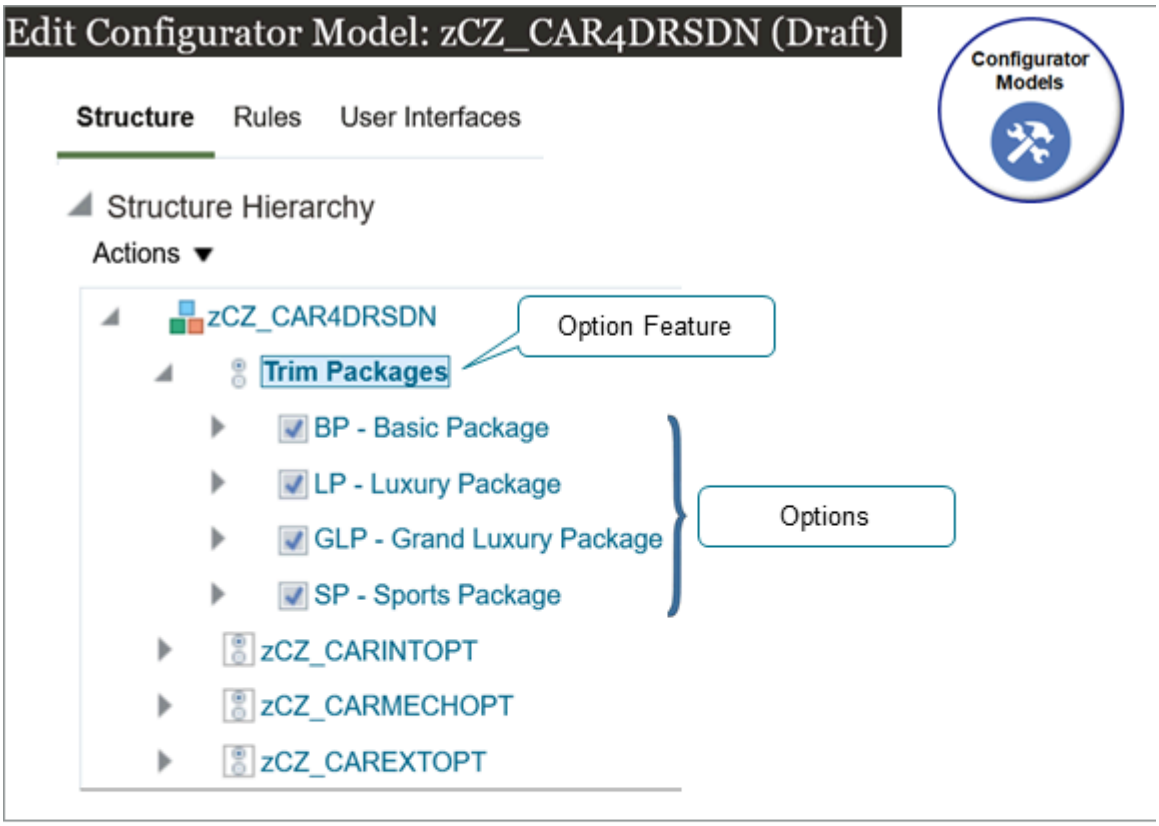
Here are some of the things you can do with an option feature.

- Include a multiple choice question.
- Control how many choices for each question.
- Specify whether to include a quantity according to the option that you choose.

- Allow your user to select more than one option for the same option feature. For example, select Red and Green.
- Create a configurator rule that automatically selects the option according to other values that your user sets.

Example

Assume you import a PIM (Product Information Management) model named zCZ_CAR4DRSDN into Configurator. You sell the car in four different trim packages, but the PIM model doesn't have these packages, so you use an option feature to add them.



Try it.

1. Open the workspace that has your model.
2. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
3. Click **Actions > Create > Option Feature**.

4. In the dialog that displays, set the values.

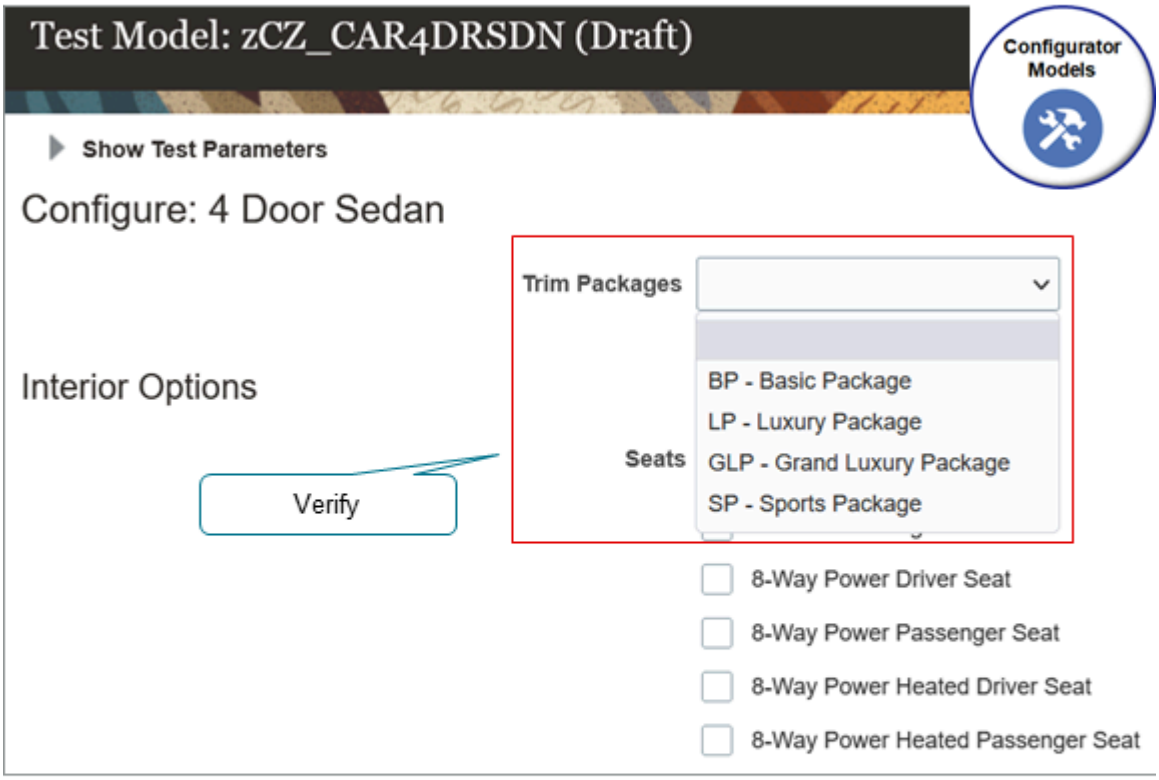
Attribute	Value
Name	Trim Packages
Minimum Selections	1 Specify the minimum number of options in this option feature that your user can select at one time.
Maximum Selections	1 Specify the maximum number of options in this option feature that your user can select at one time.
Enable Option Quantities	No Set this attribute to Yes to allow your user to enter a quantity for each option. If you set Enable Option Quantities to Yes, then you must also set the Maximum Quantity per Option attribute.

5. Click **OK**, then notice that the Structure Hierarchy area now includes the Trim Packages option feature.
6. In the Structure Hierarchy area, click **Trim Packages**, then click **Actions > Create > Option**.
7. In the dialog that displays, enter the value, then click **Apply and Create Another**.

Attribute	Value
Name	BP Basic Package

8. Repeat step 6 for the other options.
    - LP Luxury Package
    - GLP Grand Luxury Package
    - SP Sports Package
9. Click **Save**, click **Test Model**, then click **OK** in the dialog that displays.

10. On the Test Model page, verify that the user interface displays your new option feature, and that you can select each option.



Some More Example Usages

Description	Description
You can select only one option for the entire option feature, and the options must be mutually exclusive.	Set Minimum Selections to 1. Set Maximum Selections to 1.

Description	Description
	Set Option Quantities to No. If you set Minimum Selections to 1, then you must select at least one option.
You have an option feature named Car Keys, they come in red, green, or blue, but all keys must be red, green, or blue. You sell up to 10 copies of the key for each car. The user must select at least one key.	Create an option feature named Car Keys. Set Minimum Selections to 1. Set Maximum Selections to 1. Set Option Quantities to Yes. Set Maximum Quantity per Option to 10. Add these options to Car Keys: Red, Green, Blue.
You have an option feature named Car Keys, they come in red, green, or blue, and you want to allow your customer to specify the number of keys that they want for each color. For example: • 3 red keys and 3 blue keys • 2 red keys, 2 green keys, and 2 blue keys You sell up to 10 copies of the key for each car.	Create an option feature named Car Keys. Set Minimum Selections to 1. Set Maximum Selections to 3. Set Option Quantities to Yes. Set Maximum Quantity per Option to 10. Add these options to Car Keys: Red, Green, Blue.
You sell a laptop computer. As an incentive, you allow the customer to choose from 0 to 3 free applications.	Create an option feature named Free Applications. Set Minimum Selections to 0. Set Maximum Selections to 3. Set Option Quantities to No. Add these options to Free Applications: • Digital Art • Video Editing • Music Editing • Website Development • Travel Guide

Sort Your Options

Configurator displays your options in the runtime user interface in the same sequence that you created them in Configurator, by default. You can sequence them differently, and Configurator will apply your sort at runtime.

Assume you want to display your options in this sequence:

- SP Sports Package
- LP Luxury Package
- GLP Grand Luxury Package
- BP Basic Package

Try it.

1. On the Test Model page, click **Finish**.
2. On the Edit Configurator Model page, click your Trim Packages **option feature**.
3. In the Details area, click **UI Presentation**, then set the Option Sort Order attribute to one of these values.
    - As Displayed in the Structure Hierarchy
    - Ascending by Name
    - Ascending by Description
    - Descending by Name
    - Descending by Description

    For this example, set it to Descending by Name.
    If you select Ascending by Description or Descending by Description, then you must also add a description to each of your options.

Related Topics

- Chunk Your Large Option Classes
- Integer and Decimal Features
- Boolean Features

Integer and Decimal Features

Use an integer or decimal feature to allow your users to enter an integer or a decimal number when configuring the item.

Here's what you can do.

- Enter a whole number for the minimum value and the maximum value for an integer feature. Each value can be a positive or negative number. Your minimum must be less than the maximum.
- Enter a whole number or a decimal value for the minimum value and the maximum value for a decimal feature. Enter a positive number or a negative number. Your minimum must be less than the maximum.
- Enter an integer value in a decimal feature.
- If you to enter a decimal value in an integer feature, then Configurator rounds the value that you enter to the nearest integer and displays an error message.

- Use a configurator rule to set the value of an integer or a decimal feature.
- Refer to the value of another integer or a decimal feature in your configurator rule.
- Use an integer or decimal feature as an input to or an output from a calculation. For example, with a calculation that you do in a rule. Assume you have integer features x, y, and z, and have this rule:

  ```
  x + y = z
  ```

  If the user enters 5 in x and 5 in y, then z will contain 10.
- Set up a constraint in your configurator rule to limit the range of values that Configurator displays at run time for an integer or decimal feature.

Example

Assume you import a PIM model named zCZ_CAR4DRSDN into Configurator. The model has an option class named Interior Accessories, and it has floor mats and carpet. You can add an integer feature that allows your user to enter a value of 0 to 4 for floor mats, and a decimal feature that allows the user to specify the carpet's pile depth with a value of 0.25 to 0.75.

Try it.

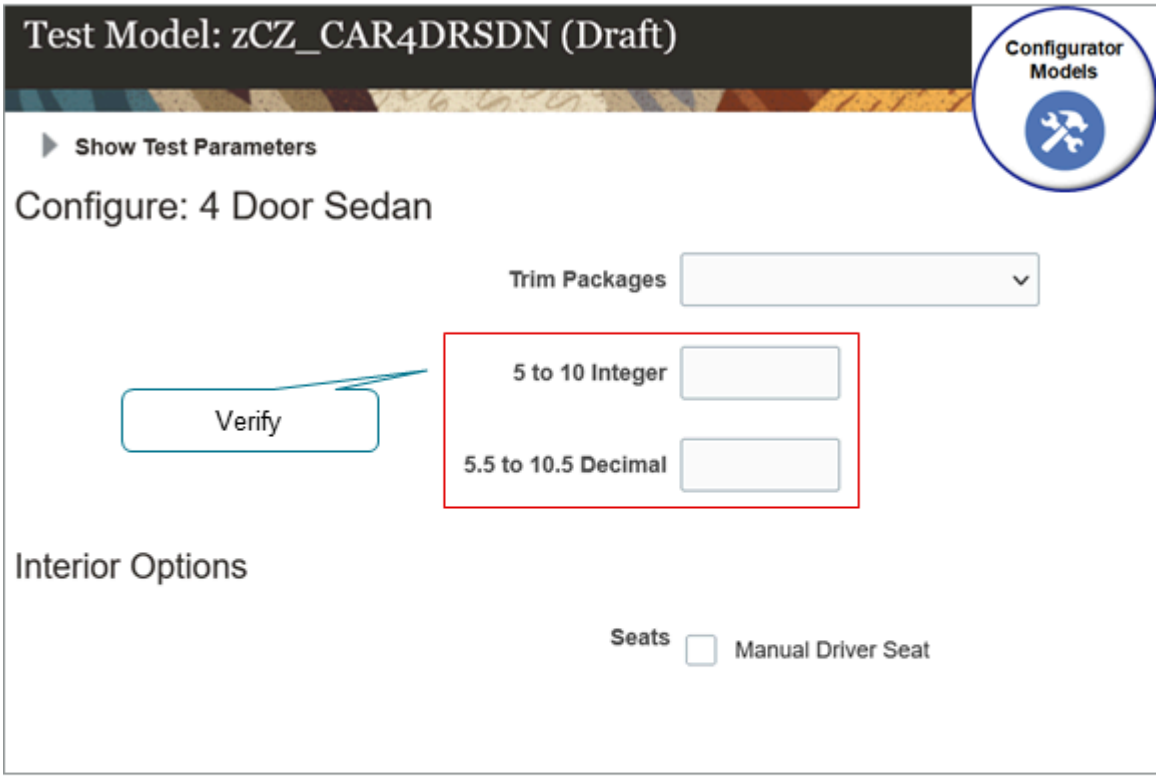
1. Open the workspace that has your model.
2. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
3. Click **Actions > Create > Integer Feature**.
4. In the dialog that displays, set the values, then click **OK**.

Attribute	Value
Name	Floor Mats
Minimum	0
Maximum	4
Domain Ordering	System Default

5. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
6. Click **Actions > Create > Decimal Feature**.
7. In the dialog that displays, set the values, then click **OK**.

Attribute	Value
Name	Carpet Depth
Minimum	0.25
Maximum	0.75
Domain Ordering	System Default

8. Click **Test Model**, the notice your new features.



9. Verify that you can enter only a whole number with a value of 0 to 4 in the Floor Mats integer feature.
10. Verify that you can enter a decimal number that's in the range of 0.25 to 0.75 in the Carpet Depth feature.

Domain Order

Use domain order to specify the values that Configurator uses to finish a user's configuration for an integer feature or decimal feature.

A domain is the entire set of values that an attribute can contain. A range is a set of values within the domain.

Assume you need to configure a window. Different types of windows and different types of wall surfaces require different tolerances for their rough openings, so you have a range of rough opening dimensions throughout your house.

You need to calculate the maximum width for each window. You can use domain ordering with a decimal feature in increasing minimum or decreasing maximum to calculate the actual dimensions of the widow for each opening.

Here are the values that you can use.

Domain Order	Description
System Default	Configurator uses its own default method to determine the value. Use System Default to optimize run time performance. Use it when you don't need to specify the value.
Binary Search, Increasing Minimum	Configurator does several binary searches to split the domain until it finds a single value, or until it doesn't find any value. Configurator examines the upper half of the domain first.
Binary Search, Decreasing Maximum	Same as Binary Search, Increasing Minimum, except Configurator examines the lower half of the domain first.

Here are the values that you can use only for an integer feature.

Domain Order	Description
Linear Search, Minimum to Maximum	Configurator uses a value that it finds in the specified domain in increasing order, beginning with the value that you specify for the feature in the Minimum attribute.
Linear Search, Maximum to Minimum	Configurator uses a value that it finds in the specified domain in decreasing order, beginning with the value that you specify for the feature in the Maximum attribute.

Related Topics

- Chunk Your Large Option Classes
- Option Features
- Boolean Features

Boolean Features

Use a Boolean feature to allow your user to select a true or false value.

- For example, ask your user whether they want tinted windows when they configure a car.
- Configurator displays a Boolean feature at run time as a check box.
- The value for a Boolean feature can be true (contains a check mark), false (doesn't contain a check mark), or empty.
- You can use a configurator rule to set the default value for a Boolean feature.

Example

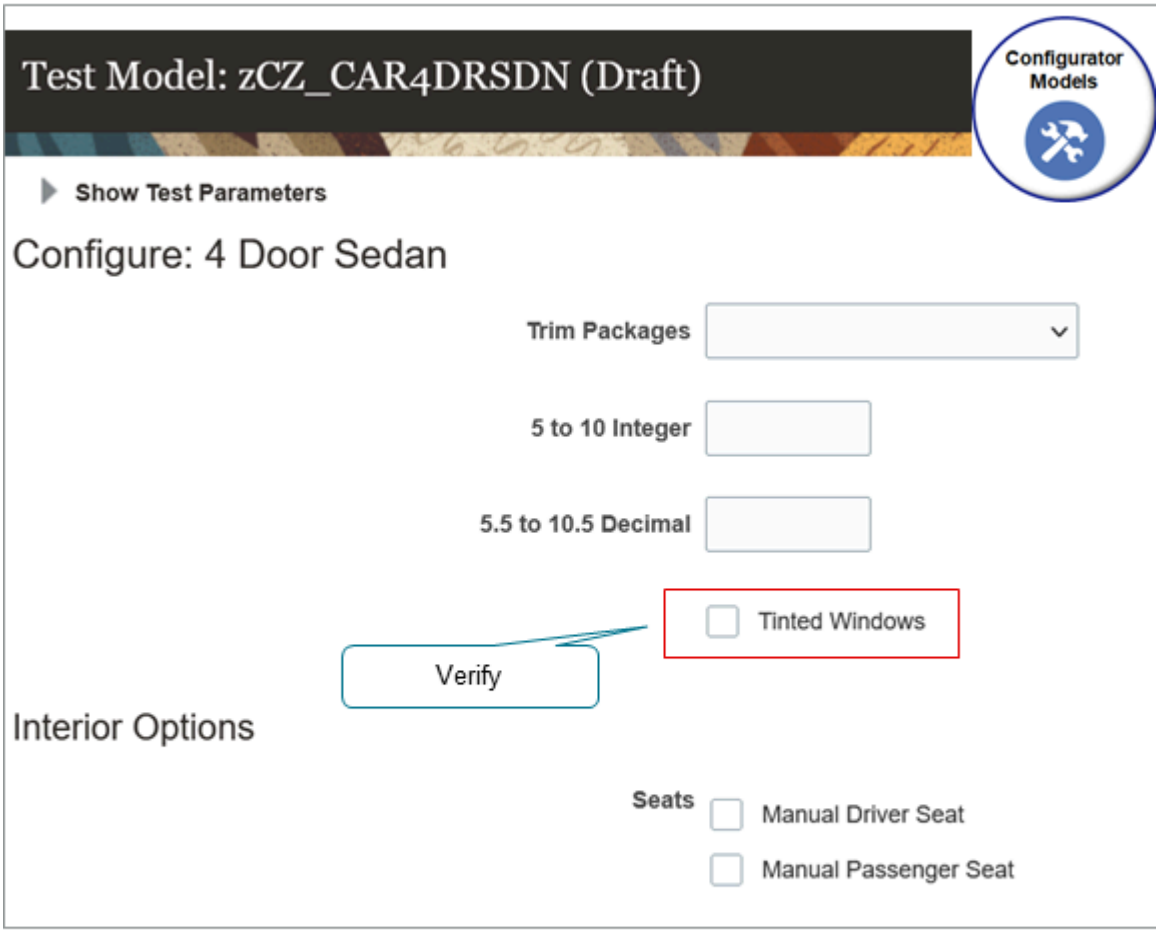
Assume you import a PIM model named zCZ_CAR4DRSDN into Configurator. You need to add a Boolean feature that allows your user to say whether they want tinted windows.

Try it.

1. Open the workspace that has your model.
2. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
3. Click **Actions > Create > Boolean Feature**.
4. In the dialog that displays, set the values, then click **OK**.

Attribute	Value
Name	Tinted Windows
Domain Ordering	Prefer True

5. Click **Test Model**, then verify that the Test Model page displays your Tinted Windows feature as a check box.



The value for the Boolean feature doesn't contain a check mark, by default. To display a checkmark, you must write a rule that results in Configurator setting the Boolean feature to true.

Domain Order

Use the Domain Ordering attribute to specify how Configurator sets the default value at run time for a Boolean feature.

Here are the values that you can use for a Boolean feature.

Domain Ordering	Description
Prefer False	Configurator sets the Boolean feature to False, by default. If another rule prevents Configurator from setting it to False, then Configurator will set it to True. For example, a *negates* rule for the Boolean feature might prevent Configurator from setting it to True or to False.
Prefer True	Configurator sets the Boolean feature to True, by default. If another rule prevents Configurator from setting it to True, then Configurator will set it to False.

Related Topics

- Chunk Your Large Option Classes
- Option Features
- Integer and Decimal Features

Text Features

Use a text feature to allow your user to enter text.

- For example, enter a name, address, or other details.
- Enter any alphanumeric value.

Configurator also uses a text feature to store a calculated value.

Example

Try it.

1. Open the workspace that has your model.
2. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
3. Click **Actions > Create > Text Feature**.
4. In the dialog that displays, set the values, then click **OK**.

Attribute	Value
Name	Monogram
Maximum Length	6 The maximum length is 32,000 characters. This is not equivalent to 32 KB, which equals 32,768 bytes. There is no relationship between Kilobytes and the maximum length of a text feature.

5. Click **Test Model**, then verify that the Test Model page displays your Monogram text feature, and that you can enter no more than six characters.

Rules

You can use a configurator rule to set the value for a text feature.

Attribute	Value
Type of Rule	Description
Default Rule	Use a default rule to set the default value that Configurator displays for a text element. Configurator applies the default rule at runtime when the user manually selects an option and enters a value. Configurator can also apply a default rule when it starts the configuration at runtime. A text feature doesn't have a range of values that you assign through minimum and maximum values, so you can't use Domain Ordering to set the default value. If your text feature must have a value, then you can use a default rule to set it.
Constraint Rule	Use a constraint rule to set the value for a text feature when the user manually selects another option and sets a value for that option. For example, if the user sets Boolean feature x to True, then the constraint rule sets the value for text feature y to value z.
Extension Rule	Use an extension rule to set the value for a text feature according to your specific needs.

Use the `Value()` method in your rule to get the value from a text feature.

If you use a rule to compare values between text features, and if these values might change, then you can only test for equality. Comparisons are case-sensitive. You must use one of these:

- `Equals()` function
- `NotEquals()` function
- = (equals) operator
- < (less than) operator
- > (greater than) operator

Here's an example rule that sets the default value of the Last Name text feature to *Enter your last name here.* You must set the rule's class to Default.

```
'UserLogin'.'Last Name'.Value() = "Enter your last name here."
```

Here's an example rule that compares the values that your user enters in the Email Address text feature to the value that the user enters in the Confirm Email Address text feature. If the values in these two features are equal, then the rule sets the Success option under the Addresses Match option feature.

```
Equals( 'UserLogin'.'Email Address'.Value() , 'UserLogin'.'Confirm Email Address'.Value()) // See if the
 values in the UserLogin text feature and the Confirm Email Address text feature are equal. If so, then the
 condition is true.
 REQUIRES
```

```
'UserLogin'.'Do addresses match?'.'Success. Addresses Match.'.State() // If condition is true, then set the
 Success feature option to Addresses Match.
```

Here's another example rule. If the user selects the Success option, then the rule sets the value of the Address Check Result text feature to *We accept your address*.

```
'UserLogin'.'Do addresses match?'.'Success. Addresses match.'.State() // Success option is selected.
 REQUIRES
'UserLogin'.'Address Check Result'.Value() = "We accept your address." // Set the value of the Address Check
 Result text feature.
```

Use Text Features as Navigation Aids

You can use a text feature as a navigation aid when you have a large configuration that displays more than page at run time. You can display text on each page that helps your user track where they are in the flow. You can then use the text feature in a rule to populate text elements that don't depend on the model's structure. You can conditionally display text at runtime. For example, display details in the text element that describe the options the user has selected.

Related Topics

- Chunk Your Large Option Classes
- Integer and Decimal Features
- Boolean Features

Supplemental Attributes

Use a supplemental attribute to simplify the decisions that your user makes to configure an item. Use it to narrow down the options that your user has to choose from.

Assume you sell a car, and choosing the engine that your customer needs is one of the most important choices your user can make, but it involves technical knowledge that the user might not have. Instead of asking the user what size engine they want, you can add a Primary Vehicle Usage option feature to your model that asks how they plan to use the car. You then add a supplemental attribute such as Fuel Economy to each option in the option feature, and set the attribute's value to help determine what engine your customer needs.

Option Feature for Primary Vehicle Usage

Option	Value of the Fuel Economy Supplemental Attribute
Office Commuting	High
Family Driving	Medium
Off Road	Low
Ride Sharing	High

For example, if you're using the car to commute to and from your office, you don't need a lot of power but you probably are concerned with economy. If you do a lot of off road driving, you're more concerned with power instead of economy.

Assume your model also has an Engine Type feature, so you add the Fuel Economy supplemental attribute to each option and set the attribute's value.

Engine	Value of the Fuel Economy Supplemental Attribute
2.0L, 259 horsepower	Medium
2.4L, 179 horsepower	High
3.6L 300 horsepower	Low

Engines that have a lower horsepower generally get better fuel economy.

But that's only part of the puzzle. How will you use this information to narrow the next question, which is to choose the engine?

You can create a rule that compares the fuel economies of the Primary Vehicle Usage and the Engine Type. Here's the pseudocode.

```
If the fuel economy that the user selects for the Primary Vehicle Usage meets the
fuel economy of the Engine Type, then display the engine type as an option. Don't display any
engine types that don't meet the Primary Vehicle Usage's fuel economy.
```

Consider a demonstration. Go to learning.oracle.com, search for, then open the Add Supplemental Attributes to Simplify the Guided Selling Process presentation. View the demonstration that starts at 1:51.

Note

- You can add a supplemental attribute only on the feature of a configurator model, such as an option feature.
- You can't add the same supplemental attribute to the same feature more than one time.
- You can add a supplemental attribute to a:
  - Supplemental structure only in the Configurator Models work area
  - PIM structure only through ADF Desktop Integration (ADFDI)
- A value set determines the values that you can use for a supplemental attribute. The value set for the values of the supplemental attribute must exist before you add the supplemental attribute.

Value Sets

Your supplemental attribute must have a value set. Here's how you can identify the value sets that you can use with your supplemental attribute.

1. Go to the Setup and Maintenance work area, then click **Tasks > Search**.
2. Search for, then open the Manage Value Sets task.
3. On the Manage Value Sets page, set the value, then click **Search**.

Attribute	Value
Module	Product Model

4. Examine the search results. Your supplemental attribute can reference any value set in the results.

   If you don't see the value set that you need, then click **Actions > Create** and create a new one. Make sure you set the Module attribute for your new data set to Product Model. Note that you can use only the Format Only validation type for a value set that you use with a supplemental attribute.

Add a Supplemental Attribute to Your Model

1. Go to the Configurator Models work area, then open the workspace that has your model.
2. In the Structure Hierarchy area, click the Engine Type option feature.
3. In the Details area, click Supplemental Attributes, click Add, then set the values.

Attribute	Value
Name	Select the attribute that you need to add. If the list doesn't have the value that you need, click Create to create a new one. See the Create a New Supplemental Attribute subtopic below for details.
Value	You must set a value that's compatible with the attribute's data type. For example, if the Data Type attribute on the line says Number, then you must set the Value to a number. If your attribute has predefined values, then select it. For example, assume you have a Color attribute, you have a data set for the Color attribute, and the data set contains the values Silver, Blue, and Gold. You can set the Value attribute to any one of these colors.
Description Value Set Data Type	Configurator automatically populates these attributes when you set the Name attribute. You can't change them, but they do help you to understand you can use the attribute that you're adding.

Create a New Supplemental Attribute

1. Click the **down arrow** in the Name attribute, then click **Create**.
2. In the dialog that displays, set the values.

Attribute	Value
Name	Enter a unique value.
Value Set	Select the value set that you need to use with your new supplemental attribute.
Default Value	Select a default value from the value set, if it has one. Configurator will apply this default value each time it uses this supplemental attribute anywhere in your model. If you don't set a value, then Configurator won't use any default value when it uses this supplemental attribute. Instead, the supplemental attribute will be empty.

Create Your Rule

The Configurator Model work area automatically creates some of the code for you when you create the statement rule, such as the node path and the attribute name, but you might need to add to this code or code it yourself in some situations to meet your specific requirements. You can write the rule in the Constraint Definition Language (CDL) or in an extension rule that you write in the Groovy language.

Here's the format that you use in CDL.

```
COMPATIBLE
&X OF 'nodePath1'.'featureName1',
&Y OF 'nodePath2'.'featureName2'
WHERE
&X.suppAttrs["supAttrName1"]=&Y.suppAttrs["supAttrName2"];
```

where

- The `&X OF` line identifies the node and feature that your user uses to specify their preference.
- The `&Y OF` line identifies the node and feature that you use to identify options that meet the user's preference.
- The `WHERE` clause filters the results so they contain only the options from the Y line that meet the requirements from the X line.
- The `X.suppAttrs` line identifies the supplemental attributes that you're comparing between featureName1 and featureName2.

For example:

```
COMPATIBLE
&X OF 'zCZ_CAR4DRSDN'.'Primary Vehicle Usage?',
&Y OF 'zCZ_CAR4DRSDN'.'zCZCARMECHOPT'.'zCZCARENGINE'.'zCZ_CARENGCAP'
WHERE
&X.suppAttrs["Fuel Economy"]=&Y.suppAttrs["Fuel Economy"];
```

where

- `zCZ_CAR4DRSDN` is the root node of the configurator model.
- `Primary Vehicle Usage?` Is the name of a feature on the root node.
- `'zCZCARMECHOPT'.'zCZCARENGINE'` is the node path to the zCZ_CARENGCAP feature.
- `Fuel Economy` is the name of the supplemental attribute that you're comparing between the Primary Vehicle Usage feature and the zCZ_CARENGCAP feature.

User Interfaces

You can use a supplemental attribute to modify the default label for some elements in the user interface.

Use the `suppAttrs['suppAttrName']` method in a user interface expression.

where

- *`suppAttrName`* specifies the name of the supplemental attribute

For example, display the value of the color supplemental attribute:

```
#{amn.suppAttrs['color']}
```

Related Topics

- Overview of Using Spreadsheets to Manage Supplemental Structures
- Chunk Your Large Option Classes

Transactional Attributes

You can specify whether to display a transactional attribute.

If your model item is part of an item class that includes a transactional attribute, then:

- The snapshot includes the item class and the value set.
- The model tree displays the transactional attribute below the node that has the imported item.
- The details pane for the transactional attribute displays the item class that provides the definition of the transactional attribute, the value set that provides the run time values for the transactional attribute, and the format details for the transactional attribute.

You specify the item class when you create an item in the Product Information Management work area. If you enable the Hidden attribute for a transactional attribute on the Edit Item Class page in the Setup and Maintenance work area, then the host application doesn't display the transactional attributes that are part of that item class. However, the Review page in the host application does display them even if you enable the Hidden attribute.

Your host application usually sends transactional attributes to a downstream application. For example, Order Management Cloud is a host application, and it uses a transactional attribute to send shipping details to Oracle Shipping. Your users might not need to see and use this attribute. You can display or not display it on the Review page.

1. Go to the Configurator Models work area.
2. Click **Tasks > Manage Models**.
3. On the Manage Models page, search for and open your model.
4. On the Edit Configurator Model page, click **User Interfaces**, then enable or disable the Display Hidden Transactional Attributes on the Review Page option.

For details, see *Manage Transactional Attributes*.

Manage Structures

Add a Supplemental Structure

You can add a supplemental structure to your configurator model.

1. Import your PIM (Product Information Management model, add it to a workspace, then open the configurator model for editing. For details about how to import a PIM structure, see *Create and Maintain Configurator Models*. Configurator uses your login ID to automatically lock the model. You can edit only a locked model.
2. Click **Tasks > Manage Models**.
3. On the Manage Models page, search for your model, then click a **draft** or a **version**.
4. On the Edit Configurator Model page, expand **Structure Hierarchy**, then notice that the tree has one root and more than one branch.

5. A supplement structure has different features. Add one now:
    - On the Edit Configurator Model page, in the Structure Hierarchy area, click the **root**.
    - You can add a supplemental structure only on the root of an item structure.
    - Click **Actions > Create**.
    - Select the type of feature that you need, depending on the kind of data that you will model in this structure.

If You Will Model	Then Click
A list of options	Option Feature Option
Integer values	Integer Feature
Decimal values	Decimal Feature
True or false values	Boolean Feature
Plain text values	Text Feature

    - Notice that the Configurator adds a branch for your new feature to the tree.
    - In the dialog that displays, enter your details.
    - In the Details area, modify the feature's attributes, as necessary.
6. Repeat step 3 to add more features, as necessary.
7. Click **Save**.
8. As an option, click **Test Model** to test your changes.

Related Topics

- Chunk Your Large Option Classes
- Manage Transactional Attributes

Chunk Your Large Option Classes

If an option class has more than 25 options, then Configurator uses the Item Selection Table with Header template to display items from the option class, by default. You can use it to help avoid a performance problem that might happen when Configurator attempts to display an option class that has too many options.

This template allows your users to:

- Display your items in a scrolling table. The table loads items while you scroll it instead of loading all items into the table the first time you open it.
- Use the Query By Example control to filter the items that Configurator displays in the table.
- Use a wild card in Query By Example to filter the items.

If this template doesn't meet your needs, then you can set up Configurator so it uses a different template.

You can also use a feature, such as an option feature, to separate a large option class into chunks, avoid the performance problem, and improve your user experience.

Assume you have an option class named Seat that contains thousands of options for different types of seats. You notice that you can chunk the options into categories:

- Seat Width
- Seat Height
- Seat Thickness
- Back Width
- Back Height
- Back Thickness
- Cording
- Color
- Fabric Type
- Foam Type
- Reclining Style
- Automatic
- Heat Styles
- Monogram

Here's how you can implement that.

1. Create an option feature named Seat Width. For details, see *Option Features*.
2. Add options to Seat Width, such as 14 inch, 16 inch, 17 inch, and so on.
3. Create the next option feature, such as Seat Height, then add its options.
4. Continue creating features for all of the categories. Use different types of features, as necessary. For example, the Monogram feature includes text, so use a text feature for it.
5. Select the **summary** on the Overview tab. This will modify the model's user interface so it displays all of the option classes and options, and a bread crumb trail so the user can see all the options they've selected.
6. Create a statement or extension rule that uses your user's response at run time as an input to each option feature, and then processes the input to select the desired seat. Here's an example statement rule.

```
'Chair'.'Cushion Width' > 30 AND 'Chair'.'Cushion Width' <= 50
REQUIRES
'Chair'.'Seat Cushion OC'.'MEDIUM_SEAT_CUSHION';
```

   In pseudocode:

```
If the chair's width is between 30 inches and 50 inches, then select MEDIUM_SEAT_CUSHION.
```

Related Topics

- Add a Supplemental Structure
- Option Features
- Integer and Decimal Features
- Text Features

Use Node Properties to Affect Runtime Behavior and Results

Each node on a configurator model has properties. You can use them to affect the choices and results that Configurator displays at runtime.

You can use these values for a variety of purposes, such as in a configurator rule that has a system attribute, or in a display condition, which is a condition that affects the values that each user interface displays.

The values of some properties are available only at run time. For example, the value of the Quantity property is available only after you select an item.

Here's a summary of the properties that are available for each type of node. For example, an Option Class is a type of node. Yes means the property is available. No means it isn't.

Property	Option Class	Option Feature	Option	Standard Item	Boolean Feature	Decimal Feature	Integer Feature	Model (Single Instance)	Model (Multiple Instances)
ChangedByA	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DefinitionMa	Yes	No	Yes	Yes	No	No	No	Yes	Yes
DefinitionMa	Yes	Yes	No	No	No	No	No	No	No
DefinitionMa	No	No	No	No	No	Yes	Yes	No	No
DefinitionMir	Yes	No	Yes	Yes	No	No	No	Yes	Yes
DefinitionMir	Yes	Yes	No	No	No	No	No	No	No
DefinitionMir	No	No	No	No	No	Yes	Yes	No	No
Description	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DetailedSelec	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
DisplayName	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DisplayNamel	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Excluded	No	No	No	No	Yes	No	No	No	No
HasChildren	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
HasTransAttr	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Property	Option Class	Option Feature	Option	Standard Item	Boolean Feature	Decimal Feature	Integer Feature	Model (Single Instance)	Model (Multiple Instances)
InErrorMode	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
InputRequire	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
InputRequire	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IsBound	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IsBoundQuar	Yes	No	Yes	Yes	No	No	No	Yes	Yes
IsMinSelectio	Yes	Yes	No	No	No	No	No	No	No
LogicState	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
MaxQuantity	Yes	No	Yes	Yes	No	No	No	Yes	Yes
MaxSelected	No	Yes	No	No	No	No	No	No	No
MaxValue	No	No	No	No	No	Yes	Yes	No	No
MinQuantity	Yes	No	Yes	Yes	No	No	No	Yes	Yes
MinSelected	No	Yes	No	No	No	No	No	No	No
MinValue	No	No	No	No	No	Yes	Yes	No	No
Name	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Proposed	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Quantity	Yes	No	Yes	Yes	No	No	No	Yes	Yes
Selected	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
SelectedCour	Yes	Yes	No	No	No	No	No	No	No
SelectionStat	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
Valid	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes

Property	Option Class	Option Feature	Option	Standard Item	Boolean Feature	Decimal Feature	Integer Feature	Model (Single Instance)	Model (Multiple Instances)
Value	No	No	No	No	No	Yes	Yes	No	No

Reduce Start Times When You Have Large Option Classes

Enable a placeholder item for an option class, and Oracle Configurator will replace the optional items in the option class with a single placeholder item.

Using a placeholder reduces the amount of time it takes Configurator to start when your configurator model has large option classes.

Also, if the model doesn't have any user-defined attributes or supplemental attributes, then Configurator won't attempt to load them into memory at run time.

Assume you already have a model item named zCZ_CAR4DRSDN, the model item contains an option class component named zCZ_CARINTOPT, and you want to reduce the overall start time for the model.

Try it.

1. Get the privileges that you need to edit items in the Product Information Management work area.
2. Go to the Product Information Management work area.
3. On the Product Information Management page, click **Tasks > Manage Items**.
4. On the Manage Items page, search for the zCZ_CAR4DRSDN item.
5. In the search results, click your **item**.
6. On the Edit Item page, click **Structures**, then click **Primary**.
7. On the Item Structure page, click **View**, then make sure the Component Order Management option contains a checkmark.
8. Select the **row** that has zCZ_CARINTOPT in the Item attribute, then click **Actions > Edit**.
9. On the Edit Components dialog, set the value, then click **OK**.

Attribute	Value
Use Placeholder	Contains a checkmark.

10. On the Edit Item Structure page, click **Done**.

Manage Attribute Values for an Option Class That You Have Enabled as Placeholder

If the Use Placeholder option contains a check mark on the option class, then Configurator won't apply some of the constraints that you specify for the components that are in that option class. It won't apply constraints that you specify for the item in Product Information Management for these attributes:

- Decimal Flag
- Default Quantity
- Minimum Quantity

- Maximum Quantity

Note

- If you enable this placeholder for more than one instance of the same option class in your model structure, then all the instances of this option class must have the same value for each of these attributes.
- This feature uses values from the option class for these attributes, so all items that are in the option class will use values from the option class at runtime regardless of each item's definition for these attributes.
- If you enable the placeholder, and then disable it, then you must change the value for each of these attribute for the option class and for all the items that are in the option class so that the values are consistent with each other.
- To avoid inconsistent behavior, you must make sure that the definition of each instance of the same option class in your model structure is the same.

Guidelines

- We recommend that you use the fully qualified path for the item in your order import.
- Each item in your option class must be a standard item. Your option class must not include another option class and it must not reference another configurator model.

If the maximum quantity for your option class is greater than one, and if you:

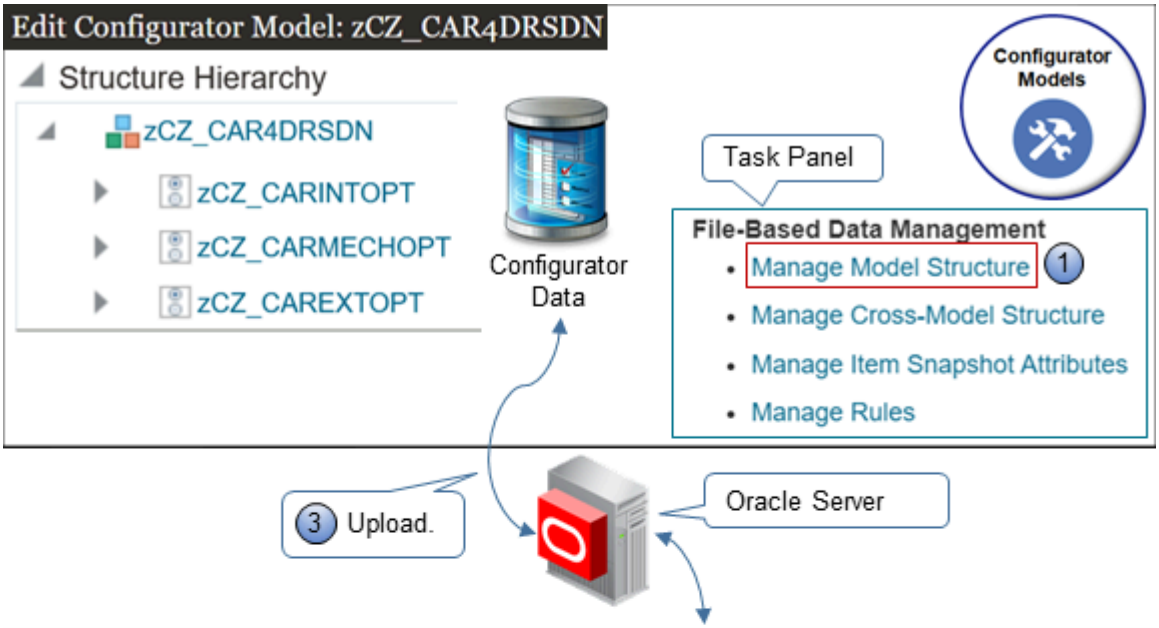
- **Enable the placeholder**. Configurator doesn't create a new page for the option class at runtime.
- **Don't enable the placeholder**. Configurator creates a new page for the option class at runtime.

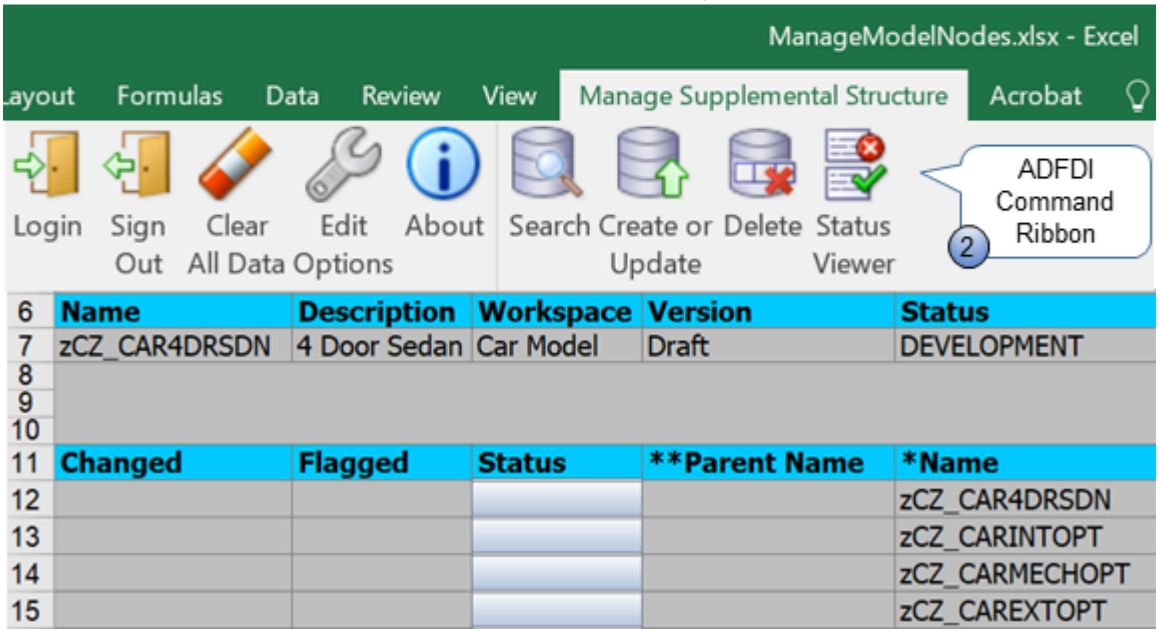
Use Spreadsheets to Manage Supplemental Structures

Overview of Using Spreadsheets to Manage Supplemental Structures

Search and download configurator data from Configurator, then manage it in a spreadsheet in Microsoft Excel.

You can use Application Development Framework Desktop Integration (ADFDI) to manage your supplemental structure, such as updating a large set of data or to move data between your test environment and your production environment.





6	Name	Description	Workspace	Version	Status
7	zCZ_CAR4DRSDN	4 Door Sedan	Car Model	Draft	DEVELOPMENT
8					
9					
10					
11	**Changed**	**Flagged**	**Status**	****Parent Name**	***Name**
12					zCZ_CAR4DRSDN
13					zCZ_CARINTOPT
14					zCZ_CARMECHOPT
15					zCZ_CAREXTOPT

How it works:

1. Download a spreadsheet from the Configurator models work area.
2. Use the ADFDI command ribbon in the worksheet to search, download, create, read, update, and delete configurator data in Excel.
3. Upload your changes to the Oracle server.

The server displays your changes in the Configurator models work area.

You can use Excel to manage:

- Option features and their options
- Boolean features
- Decimal features

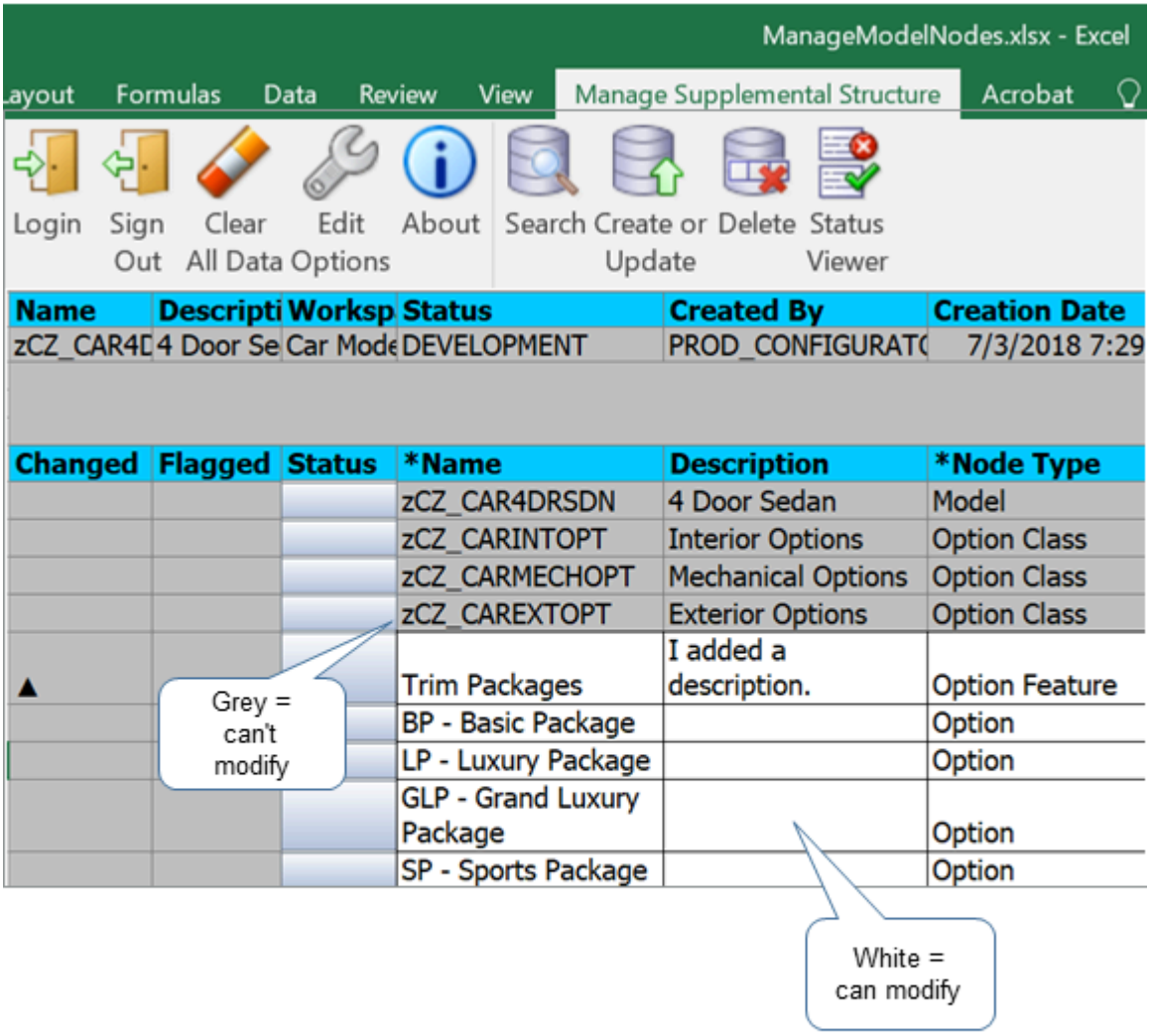
- Integer features
- Text features

Here's a description of some of the work that you can do.

Worksheet	What You Can Do
Option feature and its options	• Create an option feature and its options. • Add a new option to an option feature that already exists. • Delete an option from an option feature that already exists. • Update an option feature's attributes. • Update an option's description. • Create, update, or delete more than one option feature and its options in a single model, or across more than one model.
Snapshot	• Manage the dates that determine when the attribute values are in effect for each snapshot.
Supplemental attribute	• Create or update a supplemental attribute and its values. • Associate a supplemental attribute and its values with an option feature's options. • Remove a supplemental attribute and its values. • Remove the association that a supplemental attribute and its values have with an option feature's options.

Example

Here's an example that includes part of the data for a configurator model.



You can't modify the rows that contain data from Product Information Management (PIM), such as Interior Options. You can modify data that you added in Configurator, such as the description for the Trim Package.

Related Topics

- Add a Supplemental Structure
- Chunk Your Large Option Classes
- Supplemental Attributes
- Use Spreadsheets to Manage Your Rules

Use Spreadsheets to Manage Your Supplemental Structures

Get the details that you need so you can use a spreadsheet to manage your supplemental structure.

Summary of the Setup

1. Install ADF Desktop.
2. Download your spreadsheet.
3. Manage your model.
4. Migrate your data.

Install ADF Desktop

1. Close all instances of Microsoft Excel on your local computer.
2. Sign into Oracle Applications. Sign in with the same user and password that you use when you sign in to use the Configurator Models work area. You might need to sign in each time you connect through ADFDI.
3. Go to the Home page, then under Tools, click **Download Desktop Integration**.
4. In the status bar at the bottom of your browser, allow the download, then wait for the adfdi-excel-addininstaller.exe file to finish downloading.
5. Open adfdi-excel-addin-installer.exe on your local computer.
6. In the ADF Desktop Integration Installer dialog, click **Developer Options**.
7. In the Developer Options dialog, click **Enabled**, click **Install**, wait for the installation to finish, then click **Close**.
8. Open the Control Panel on your local computer, click **Programs and Features**, then verify the list that displays includes your new installation.

   For example, make sure it displays Oracle ADF 11g Desktop Integration Add-In for Excel.
9. Open Excel, then, in Microsoft Office Customization Installer, click **Install**.
10. Click **File > Add-Ins**, then verify that the Add-Ins submenu contains ADF Desktop Integration.

    In some Excel versions, you might need to enable the add-in. For details about installing add-ins, see the documentation for *Microsoft Excel*.
11. Close Excel.

Download Your Spreadsheet

For this example, assume you need to manage a model named zCZ_CAR4DRSDN.

1. Make sure each model that you want to manage meets these requirements:
   - Is in Draft status.
   - Contains at least one participant in a workspace that isn't released, and at least one of these workspaces is in the In Development status.
   - Another user hasn't locked the model.

2. Download a spreadsheet.

   There are different ways to do this.

What You Want	Manage a Supplemental Structure	Manage Option Features for More Than One Model	Manage Supplemental Attributes and Snapshots	Manage Rules
An Empty Spreadsheet	Go to the Manage Models page, then click **Actions > Download Spreadsheet for Managing Model Structure**.	Go to the Manage Models page, then click **Actions > Download Spreadsheet for Managing Cross-Model Structure**.	Go to the Manage Snapshots page, then click **Actions Download Spreadsheet for Managing Item Snapshots**.	Not Available
A Spreadsheet That I Can Search	Click **Tasks > Manage Model Structure**.	Click **Tasks > Manage Cross-Model Structure**.	Click **Tasks > Manage Item Snapshot Attributes**.	Click **Tasks > Manage Rules**.

   For this example, click **Tasks > Manage Model Structure**.
3. In the dialog that displays, save the file to your desired location.
4. Use Microsoft Excel to open the file that you just saved.
5. In Excel, in the Connect dialog, click **Yes**.

   You use this dialog to connect to the server that hosts your Oracle application.
6. In the Login dialog, enter your user name and password, then click **Sign In**.

   Sign in with the same user and password that you use to access the Configurator Models work area.

Here are the file names that you can download.

- ManageModelNodes.xlsx
- ManageCrossModelOptionFeature.xlsx.
- ManageItemSnapshotSupplAttrValAssoc.xlsx
- ManageStatementRules.xlsx

You can also manage rules. See *Use Spreadsheets to Manage Your Rules*.

Manage Your Model

1. Examine the menu bar at the top of the page, and notice that Excel has a new tab named Manage Supplemental Structure, and that the tab includes commands that you use with ADFDI, such as Login, Logout, and Search.
2. Click **Search** on the ADFDI command ribbon.
3. In the Search Models dialog, enter zCZ_CAR4DRSDN, then click **Search**.
4. In the search results, click the **row** that has your model, then click **OK**.

   Notice that ADFDI populates the spreadsheet with your model's details.
5. Insert the new rows or cell values that accomplish your update.
6. Click **Create or Update** on the ADFDIcommand bar.

7. Go to the Configurator Models work area and confirm that the upload uploaded your changes.

Migrate Your Data

You can migrate data between environments.

Assume you need to migrate your rules from model x in your test environment to model y in your production environment.

1. Sign into your test environment, search for model x, then download the rules from model x.
2. Sign into your production environment, then make sure the production environment has all the workspaces and models that you reference in the rules that you are migrating.
3. Search for model y, then download the rules from model y.
4. Copy the data from model x's spreadsheet. Make sure you only copy cells that you can update.
5. Paste your data into model y's spreadsheet.
6. Click **Create or Update** to upload your data to production.

Guidelines for Using Spreadsheets to Manage Your Supplemental Structures

Use these guidelines to help you when you use a spreadsheet to manage your supplemental structure.

- Download the worksheet only one time. For subsequent uses, open the xls file directly on your local computer.
- Save and move the xls file to any location on your laptop, or copy it to another computer.
- To make sure the data in the worksheet is up to date with the data on the server, refresh the search each time after the worksheet finishes processing your action.
- If you must process more than 10,000 records, then separate data into batches of 10,000 records or less for each batch.
- Each workbook contains address details, such as the URL, so it can connect to a specific environment. If you use more than one development environment, then don't use the same workbook in these environments. Instead, download a separate workbook into each environment.
- Don't open more than one workbook at the same time. For example, if you must manage a model structure and a snapshot, then open the workbook for the model structure, modify the model structure, and close the workbook. Then open the workbook for your snapshot. Opening more than one workbook might cause problems in the cache, clipboard, and other areas.
- Be aware of session time out settings. For example, if your session times out after 15 minutes and you're still working in the workbook, you might not be able interact with the server. Commands to the server won't respond. Close the workbook, open the workbook, and sign in.
- Keep your Application Development Framework Desktop Integration (ADFDI) plug-in up to date. The workbook usually informs you when an update is available.
- If Excel prompts you to save the file, then don't save changes in the spreadsheet because this data might become old compared to the data on the server. Instead, download data from the server immediately before each action.
- Don't use CTRL+S, and don't use Save in the File menu in Excel.
- You can't use the Undo command in Excel to undo changes while in Connected mode.

- You can't select a row and then click Delete to delete it while in Connected mode. Instead, use the Delete command on the Application Development Framework Desktop Integration ribbon.
- Don't use the Clear or Clear All command in Excel. Instead, use the Clear All Data command on the Application Development Framework Desktop Integration ribbon.
- You must use Microsoft Excel. You can't use any other spreadsheet application.

Use the Spreadsheets

- Each spreadsheet has a different set of columns. Make sure you enter the correct value for each column.
- A column that has a grey background means you can't edit it. Its read only.
- A column that has a white background means you can edit it.
- Examine the Changed column to see what you've modified. It displays a value in each row that you modify.
- A single asterisk (*) in a column means you must enter a value in that column.
- A double asterisk (**) in a column means you might need to enter a value in that column, depending on values that you set in other columns.
- If you want to copy data from a row, make sure you don't copy the value from the Key column. The Key must be unique.
- You can rearrange, resize, or hide columns to meet your preferences.
- Don't delete any column. Don't copy and paste any column. If you delete a column, or copy and paste a column, then you might encounter an error.
- To create a new record, add data to an empty row. Add values to attributes that display with a white background or a grey background when you create a new record.
- If the Status cell of a row contains Insert Failed, click No when you're prompted whether to discard the pending changes from your update, then examine the Status Viewer to get details about the failure.

Here's a summary of the elements that you can use.

Spreadsheet Element	Description
Login Logout	Sign into or sign out of a session with the Oracle server.
Clear All Data	Click Clear All Data at the end of each session to clear data from the spreadsheet. This action helps to make sure the spreadsheet is up to date with data on the server during the next session.
Edit Options	This value references the server address. You can modify it to use the spreadsheet with a different server. In most situations, don't modify the value in the Edit Options dialog. Instead, use a different spreadsheet for each server.
Search	Search data on the Oracle server.
Create or Update	Upload data that you created or updated in the Oracle spreadsheet. You upload to the Oracle server.
Delete	Select one or more data rows in the spreadsheet, then click Delete to delete them from the server. This action happens immediately while in Connected mode and you can't undo it.
Status Viewer	View messages that describe the result of your actions. For example, if the Status Viewer displays No error, then your action was successful.

Spreadsheet Element	Description
	If an error happens, for example you don't include a required attribute, then the Status Viewer displays an error message.
Changed Column	Displays an icon that indicates you modified a value in the row but haven't yet uploaded your modification to the server.
Flagged Column	Double-click in the Flagged column to delete the row. The Flagged column displays an icon that indicates you plan to delete the row but haven't yet uploaded your deletion to the server. Click **Delete** in the ribbon to upload your deletion.
Status Column	Displays the status of your action, such as Row Inserted Successfully, Update Failed, or Insert Failed. Scan the Status column after each action to make sure the server successfully processed your request. If the Status is empty after a request or displays Row Updated Successfully, then the update was successful. If an error happens, then the Status column displays a summary of the error. Use the Status Viewer to view error details.
Key Column	Displays a value that uniquely identifies the record. You can't modify this value. Make sure you don't select it when you copy a row. You can hide the Key column.

Use Connected Mode or Deferred Mode

The spreadsheet works in Connected mode or Deferred mode, but these modes are transparent to you. You don't need to take an action to use one or the other. For example, if you download data to the spreadsheet from the server, edit a row, then upload the edit, then the spreadsheet works in Connected mode. It maintains an active connection to the server.

An active connection doesn't exist in Deferred mode. For example, assume you open your spreadsheet but don't connect to the server. You intentionally cancel out of the sign in dialog that displays. Instead, you copy data into the Oracle spreadsheet from some other source, such as your own spreadsheet file. You sign into the server and are now in Connected mode. You click Create or Update to upload your changes.

Use Your Own Spreadsheet

You can maintain your own spreadsheet that has your data, then copy and paste that data into the Oracle spreadsheet.

- Make sure that your own spreadsheet has the same columns and that they're in the same sequence that the Oracle spreadsheet uses.
- Make sure you copy only data rows.
- Don't overwrite header rows in the Oracle worksheet.

Use the ManageCrossModelOptionFeature.xlsx File

You must set the Node Type to OPTION_FEATURE or OPTION.

Use these sections in the worksheet:

Section	Description
Manage Cross-Model Option Feature.	Create and manage your option features.

Section	Description
Supplemental Attributes	Enter data in this section to create an association between a supplemental attribute that already exists and an option feature that you manage in the Manage Cross-Model Option Feature section.
Status and Error Messages	Get a status update and view error messages.

Troubleshoot

- If an error happens, then the Status column displays a summary of the error status. Use it to read the summary.
- Use the Status Viewer to read the error message.
- If an error happens, then the spreadsheet displays the Download dialog. Notice the text *Do you want to discard the pending changes?*. Click a value.
  - **Yes**. Discard the change that causes the error. The server doesn't update data and the spreadsheet doesn't display details about the error.
  - **No**. The server doesn't update data, but the spreadsheet does display details about the error in the Status column and in the Status Viewer.
- Use the Search command to restore the values in your spreadsheet after you encounter an error.

Here are some more troubleshooting details.

Trouble	Shoot
You receive a message. `Unable to execute the command Create or Update while a cell is in edit mode.`	Step out of the cell you're editing, then retry the update.
You receive a message. `Maximum: cannot convert the input value to the expected data type (BigDecimal).` Some messages in the Status Viewer includes details. • A prefix that identifies the attribute name. In this example, the error happens in the Maximum attribute. • A parenthetical that suggests a correction. In this example, (BigDecimal) is the parenthetical, and it suggests that you use BigDecimal as the data type value for the Maximum attribute.	Make sure the value in the Maximum attribute contains a BigDecimal data type.
You receive a message. `Required property UniqueAttribute missing.`	Make sure each required attribute contains a value.

Trouble	Shoot
You receive a message. `View row with key Oracle x is not found.`	Refresh your data. You might encounter this error if you perform an action, don't refresh your data, then do another action. For example, assume you. 1. Download data to the Manage Model Structure worksheet. 2. Download data to the Manage Pricing Charges worksheet. 3. Make, then upload a change on the Manage Rules worksheet. 4. Navigate back to the Manage Model Structure worksheet but don't refresh your data. 5. Make, then upload a change on the Manage Model Structure worksheet. To fix this problem, refresh your data after you navigate back to the Manage Model Structure worksheet, then make and upload your changes.

5 Rules

Overview of Creating Rules for Configurator

Create statement rules and extension rules to meet your own, specific customization requirements.

Oracle ACT (Advanced Constraint Technology) is a tool that uses artificial intelligence to help you solve complex problems that can include a wide variety of possible combinations.

With some technologies, you can use simple If/then rules to evaluate conditions and take action in a specific sequence, but these rules can lock you into a fixed set of guided flows. Instead, you can use Oracle Configurator to transform a configuration model into a constraint satisfaction problem. Oracle ACT then uses your statement rules and extension rules to filter constraints and make sure each configuration is valid. Oracle ACT uses algorithms to filter constraints, and these algorithms examine each variable and its set of values to validate your configuration.

Use Oracle ACT to:

- Navigate through a problem and make specific settings throughout the configuration session at run time, where each setting refines the range of possible configurations. You can select only one option or many at any point during the session, and Oracle ACT will automatically create a complete and valid configuration, saving you time and reducing errors.
- Dynamically create an optional substructure at run time, then use that substructure to scale and represent a large and complex product. Assume your model has tens of thousands of optional components. Oracle ACT loads only the components that it needs to make your configuration valid. It doesn't load the tens of thousands of components it doesn't need, helping to make sure you can scale and manage your implementation.
- Solve unique features, such as a user guided search that transforms selections into constraints and uses them to guide the search to a preferred configuration.

You don't need to do anything to enable Oracle ACT. Configurator comes predefined to automatically use Oracle ACT with your statement rules and extension rules.

Related Topics

- Overview of Creating Statement Rules
- Overview of Using Constraint Definition Language
- Overview of Creating Extension Rules

Statement Rules

Overview of Creating Statement Rules

Create a statement rule to extend your model. You can implement your own logic, accumulate values, and make sure each option in your model is compatible with other options in the model.

Here are some typical ways that you might use a statement rule.

- Set default choices or values.
- Automatically select an option according to other selections that your user makes.
- Prevent you from making selections that won't work each other.
- Specify how many instances of a model Configurator creates at run time.
- Calculate and set the value for a numeric option.

Here are some example statement rules.

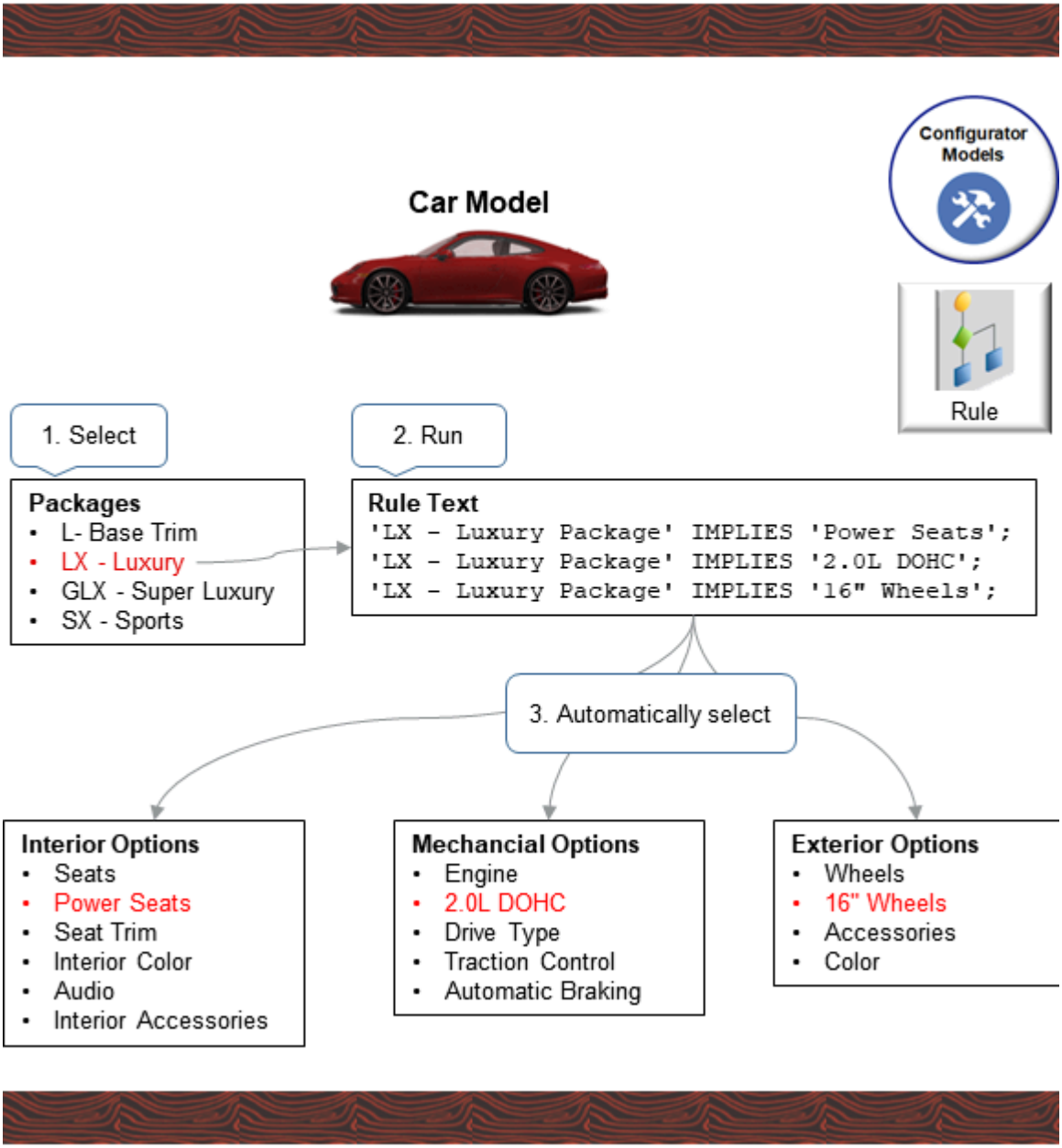
Description	Code

Set the value of a text feature when you select an option.	`'UserLogin'.'Do addresses match?'.'Success. Addresses match.'.State() // Success option is selected.`
	`REQUIRES`
	`'UserLogin'.'Address Check Result'.Value() = "Your address is accepted." // Set value of text feature.`

Select an option according to a choice that your user makes.	`'Task Chair'.'Cushion Width' > 30 AND 'Task Chair'.'Cushion Width' <= 50`
	`REQUIRES`
	`'Task Chair'.'Seat Cushion OC'.'SEAT_CUSHION_12345';`

Set a default value.	`Weight <= 3000`

Assume you need to automatically select options according to a choice that your user makes in a supplemental structure.



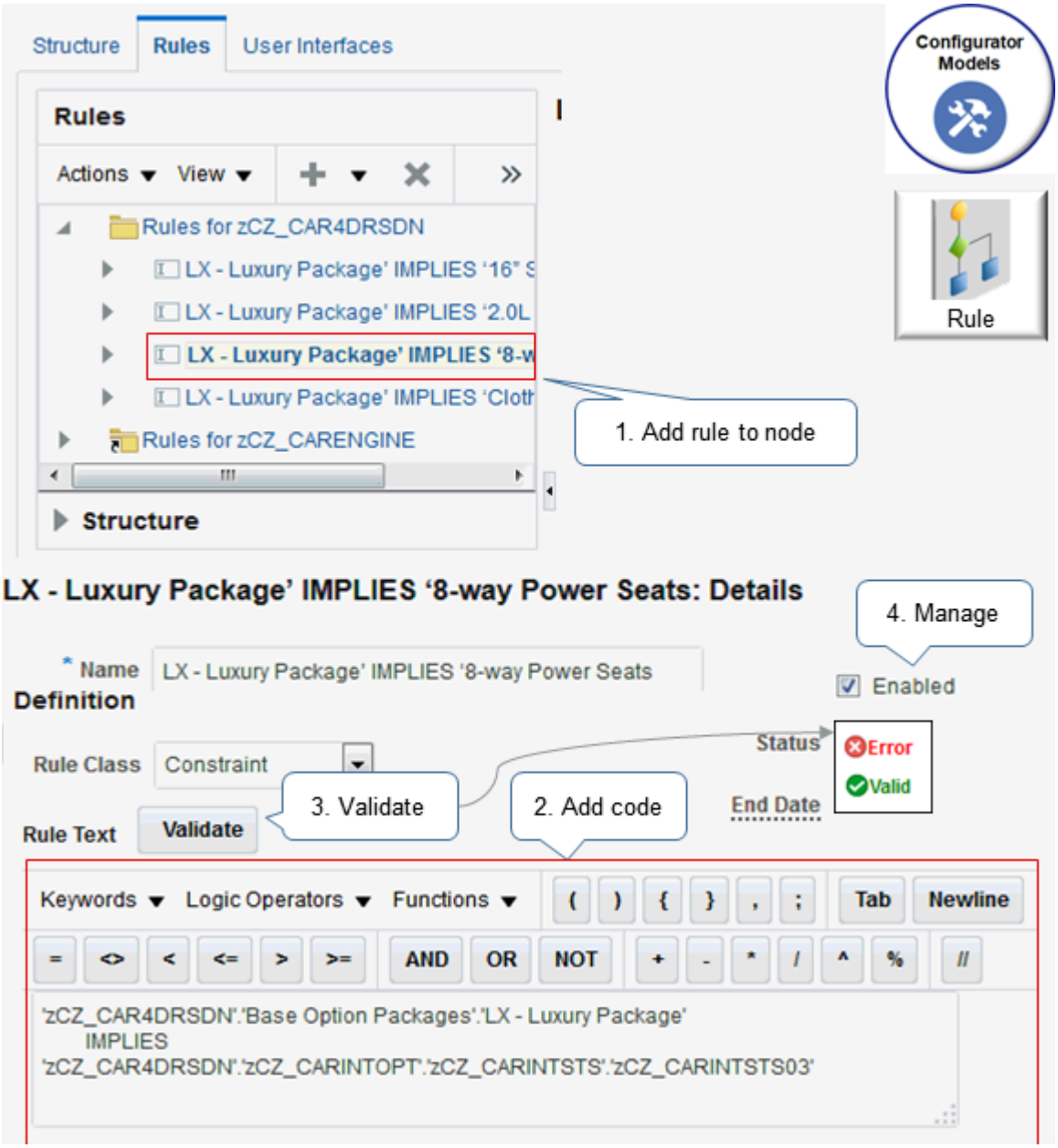
Note

1. You add the Trim Packages supplemental structure to your Car model, and you add options to Trim Packages, such as LX Luxury Package. At run time, your user selects the LX Luxury Package option.
2. You add a statement rule to your model. For example, this code says that if you select LX Luxury Package, then set Interior Options to Power Seats:

'LX Luxury Package' IMPLIES 'Power Seats';

- At run time, the rule automatically selects other options in the model according to the LX Luxury Package that your user selected:
  - Set Interior Options to Power Seats
  - Set Mechanical Options to 2.0L DOHC
  - Set Exterior Options to 16" Wheels

Here's how you do it.



1. Use the Rules area of the Configurator Modeling work area to add your rule to a node on the model. Each statement rule is a part of the model, just like a supplemental structure or user interface.

2. Use the code editor to add your code. Use the code editor's features to help make sure your code is correct, such as inserting keywords, logical operators, functions, and so on. You use the Constraint Definition Language (CDL) to code each rule.
3. Click Validate to make sure your code doesn't have any errors that will prevent Configurator from compiling it, such as syntax errors.
4. Manage your rule. Verify its status after you validate, enable it, and end date it.

Guidelines

- Configurator automatically imports rules when you import your item from the Product Information Management work area into Configurator. For example, when an option class that has options that are mutually exclusive.
- A statement rule is different from a condition that you create for the user interface. A statement rule makes sure your configuration is valid, while a condition specifies how to display or whether to enable an element in a user interface depending on the value of an attribute.
- You can have an implicit rule in a supplemental structure, such as the minimum number or maximum number of selections in an option feature.
- You can't share a statement rule with another model, but you can manually copy it into another model.
- Configurator ignores rules that aren't valid when you test the model.
- You must disable or delete each rule that isn't valid before you release your workspace.

Manage Your Rule

Use the Rules area to manage your rules. You can create and delete rules, organize them into folders, and into a tree. Use these attributes for each rule.

Attribute	Description
Enabled	If you disable the rule, then Configurator won't apply it to your model.
Status	Contains one of: • Modified. You edited the rule text or modified one of rule's attributes since the last time you compiled the rule. • Error. There's an error that's preventing Configurator form compiling the rule. • Valid. You can compile and test the rule.
End Date	If you set the end date for an object that the rule references, then the End Date attribute displays that object's end date. You can't modify the End Date attribute on the Rules tab, but you can modify the object's end date, and the End Date on the Rules tab will display your modified date.

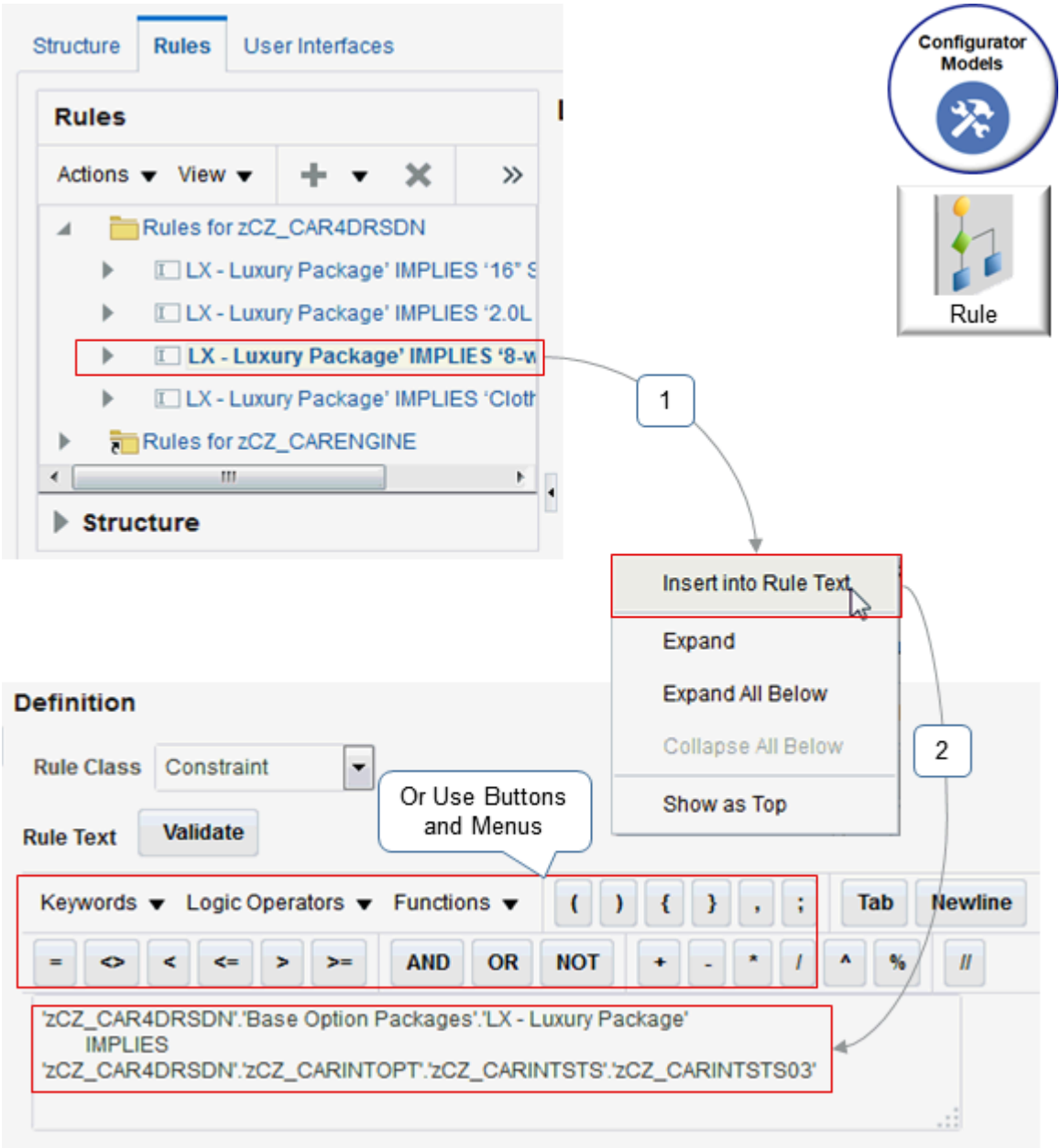
Related Topics

- Set the Rule's Class
- Specify Your Rule's Logic
- Accumulate Values
- Make Sure Options Are Compatible

Create Statement Rules

The rules editor provides you with a great deal of flexibility when you create your rules. You can use various buttons, menus, and actions to create your rule text, or you can manually enter code into the Rule Text window.

For example, you can right-click a node, click Insert into Rule Text, and the editor will automatically add code that is syntactically correct into the rule text.



Its critical that you use these buttons, menus, and actions correctly. Viewing a demonstration and taking training is probably the most efficient way to learn how. To view a demonstration, go to *Customer Connect: SCM Use Statement Rules with Oracle Configurator*. Watch the demonstration that starts at 8:00.

Here's what you'll do:

- Examine a configurator model for a tablet computer. The model has some guided selling questions, option classes, and a child model with its own structure that includes accessories that you can add to the tablet.
- Examine a constraint that automatically sets the default value for several choices and constrains you from modifying those values.
- See how changing the rule's class from Constraint to Default automatically sets default values but allows you to modify them.
- See how changing the rule's class from Default to Search Decision doesn't set any default values, requires you to set all the required values, and then uses autocomplete to finish the configuration when you click Finish and Review.

You can also take the Configurator Modeling training that's available through *Oracle University*. You will get hands-on experience in creating your rules.

Summary

You use the rule editor in the Configurator Models work area to create a statement rule. Here's a summary of how it's done.

Step	Action	Description
1	Go to the Edit Configurator Model page, Rules tab, then click **Actions > Create > Statement Rule**.	Use the folder in the Rules area to organize and manage your rules. You can create more than one statement rule and work on them separately.
2	In the Create Statement Rule dialog, enter a name for your rule, add a description, then select the rule's class.	You can modify the rule class, as necessary. See *Set the Rule's Class*.
3	In the Definition section, enter your rule's CDL text.	See *Specify Your Rule's Logic*.
4	Use the menus to insert CDL syntax into the rule text.	Using the menus helps to make sure that you only insert elements that have the correct syntax.
5	Use Structure area to reference a node on your model.	Each of you CDL statements can reference one of your model's nodes. To make sure that your reference is syntactically correct, use the Structure area to search for and select the node, then click **Insert Into Rule Text**.
6	Click **Validate**.	Validate makes sure your syntax is correct, and makes sure your node references are correct.
7	Examine the Status indicator.	Use the Status indicator to see whether you modified the rule and to see whether it's valid or has an error. Examine any error messages, as necessary.

Step	Action	Description
		You can leave your rule in an Invalid status, and Oracle Configurator will ignore it when you test the model.
8	Click **Save and Compile**.	You have to save and compile before test. If you rule has an error, you can click **Save** to save your changes.
9	Click **Test Model**.	Make sure your model behaves as you expect it to.

Test Your Statement Rule

1. Click **Test Model**. You don't have to be on the model's Rules tab to test it.
2. In the Test Model dialog, set the User Interface attribute to an interface that includes the nodes that your rule references. The dialog automatically selects the interface that you most recently tested, by default.
3. Select the various configuration options that you need to test. Make sure you select all possible choices and configurations that your users might make. Navigate through the configuration, and verify that the selections you make affect other selections as you expect them to.

Related Topics

- Edit Statement Rules
- Overview of Creating Statement Rules
- Overview of Using Constraint Definition Language

Edit Statement Rules

Use the editor to create your statement rule.

Go to the Edit Configurator Model page, then use the Rules tab.

- Use the tree on the Rules pane to access and manage the rules that you define. Examine the icons to quickly get a picture of your rules' statuses.
- In the Structure section, select your model's node, then click **Insert into Rule Text** to insert a reference to your model's node directly in your rule's text.

Here are some of the editor's features that you might find useful.

Feature	Description
Enabled option	Disable a rule while you're working on it or to prevent Oracle Configurator from applying the rule the model.
Status indicator	The status values are:

Feature	Description
	• **Modified**. You edited the rule text or changed some part of it since the last time you compiled the rule. • **Error**. There's an error that's preventing the rule from compiling and running, or the rule has a syntax error. • **Valid**. The rule is valid, and you can compile it and test it.
End Date attribute	A read only attribute that displays the date when the rule will become invalid because the end date has expired for one of the rule's participants.
Definition section	An area that controls how you define the rule.
Rule Class	Use this attribute to change the rule's class, as necessary. If the class is a Default or Search Decision class, then you can click **Reorder** to change the sequence that you want Configurator to use when it applies default values or search decisions. See *Set the Rule's Class*.
Validate	Make sure your rule's syntax is valid.
Text pane	Enter your statement's text in CDL format. Use the toolbar and text buttons to help make sure that the text you insert has the correct syntax.
Insertion menus	Use these menus: • **Keywords**. CDL keywords that you can use to create your statement. • **Logic Operators**. CDL operators that you can use to define your statement's logic. See *CDL Operators*. • **Functions**. Use this set of menus to enter CDL functions: o Logic o Arithmetic o Trigonometric o Set o String See *CDL Functions*. You can tear off a menu that you use frequently and leave it positioned over the editor while you work.
Text buttons	Use these buttons to insert CDL elements directly into your rule text. You can insert: • Characters to delimit each of your statements • Comparison operators • Boolean operators • Mathematical operators

Related Topics

- Create Statement Rules
- Overview of Creating Statement Rules
- Overview of Using Constraint Definition Language

Set the Rule's Class

Each statement rule has a class that you can set to tell Oracle Configurator when and how to apply the rule at run time.

The class specifies:

- The rule's behavior
- Whether the rule is required
- When to apply the rule at run time

Constrain Values

If you set the rule's class to Constraint, then Configurator applies a constraint at run time when you manually select an option or enter a value.

You can include logic, compare values, or make sure options are compatible in your constraint.

You can use a relational operator in your constraint. Here's an example that uses the equal (=) operator and the greater than (>) operator:

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

Note

- The left side of the expression can push numeric details to the right side.
- The right side of the expression can push numeric details to the left side.

A constraint must always be true. For example, if you a selection that violates a constraint, then Configurator displays a message informing you that Configurator can't do the action that the selection requires.

You can't specify the sequence that Configurator uses to run your constraints at run time.

Set Default Values

If you set the rule's class to Default, then the rule can use a default value to help guide you toward a solution.

Assume you sell flower bouquets and you prefer that your customers purchase My Summer Bouquet instead of My Fall Bouquet, and to purchase the Crystal Vase instead of the Glass Vase. These rules and this sequence will get you there:

1. My Bouquet Implies Summer
2. My Bouquet Implies Crystal Vase
3. Fall Bouquet Implies Crystal Vase

If you select My Bouquet, then Configurator will automatically select Summer Bouquet and Crystal Vase by default. If you then select Fall Bouquet, then Configurator will automatically select Crystal Vase.

You can create more than one constraint as a default. Assume your model contains a numeric feature named Weight and it has a range of 1,000 to 5,000. You prefer to sell a solution where the value of this item is less than or equal to 3,000, so you create a statement rule and assign the rule to the Default class:

```
Weight <= 3000
```

Configurator will reduce the range for the weight item at run time:

```
Range: 1000 to 3000
```

If you make some selections that result in a Weight that's outside this range, then Configurator overrides the default.

Note

- Configurator applies a default rule at run time when it starts the configuration, and before you make any selections.
- Configurator applies the default values in the same sequence that you specify them in your rule.
- Configurator might apply a default rule when you manually select an option or enters a value.
- The default isn't a constraint. For example, if the default automatically adds a checkmark to an option feature, then Configurator doesn't prevent you from removing the check mark.
- You can express a default rule as an assignment function.
- If your choices or if a constraint causes a conflict, then the default might fail.

Specify Search

If you set the rule's class to Search Decision, then Configurator applies the rule:

- During autocomplete
- After it applies all user selections, all constraints, and all defaults
- Before it applies the constraint's search decisions
- In the same sequence that you specify search decisions in your rule

You can express a search decision rule as an assignment function, logical expression, or numeric comparison.

Related Topics

- Overview of Creating Statement Rules
- Specify Your Rule's Logic
- Accumulate Values
- Make Sure Options Are Compatible

Specify Your Rule's Logic

Use a logic operator in your statement rule to enforce a relationship, such as Requires, Negates, Implies, and so on.

Consider this pseudocode:

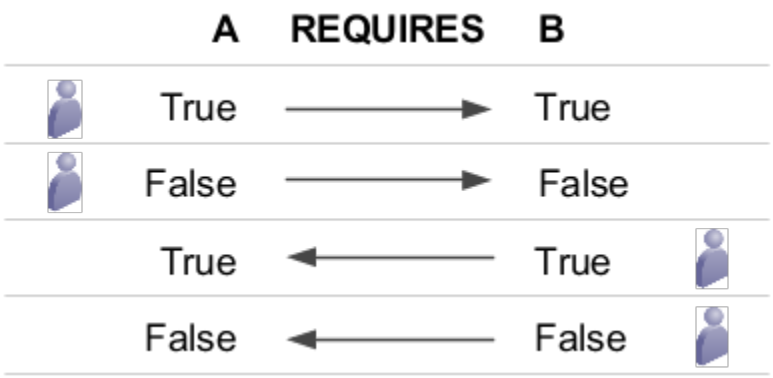
```
If you select option a in the x configured item, then you must also select option y.
```

Note

- You can propagate logic from operand *x* to operand *y*, or from operand *y* to operand *x*.
- If your logic doesn't propagate the value for a relationship, then the rule doesn't modify the option's logic on the other side of the rule.
- The term *true* means the configuration includes the option.
- The term *false* means the configuration doesn't include the option.
- We use the terms *true* and *false* here only to document how the rule works. You don't use *true* and *false* in your code.

Requires

If you select an option, then Oracle Configurator applies the same value to the other option.

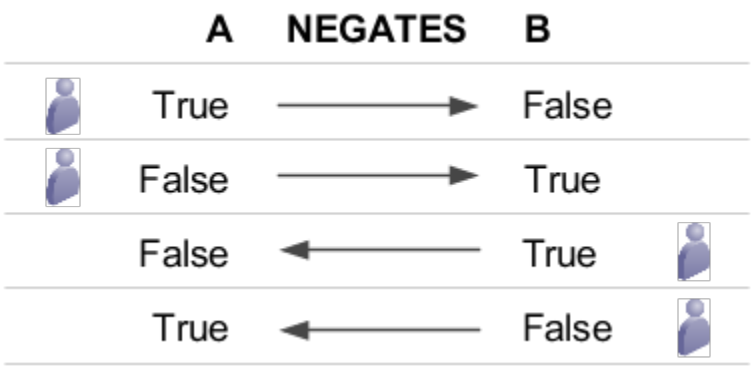


If you:

- Select option *a*, then Configurator also selects option *b*.
- Deselect option *a*, then Configurator also deselects option *b*.
- Select option *b*, then Configurator also selects option *a*.
- Deselect option *b*, then Configurator also deselects option *a*.

Negates

If you select an option, then Configurator applies the opposite value to the other option.



If you:

- Select option *a*, then Configurator deselects option *b*.
- Deselect option *a*, then Configurator selects option *b*.
- Select option *b*, then Configurator deselects option *a*.
- Deselect option *b*, then Configurator selects option *a*.

Implies

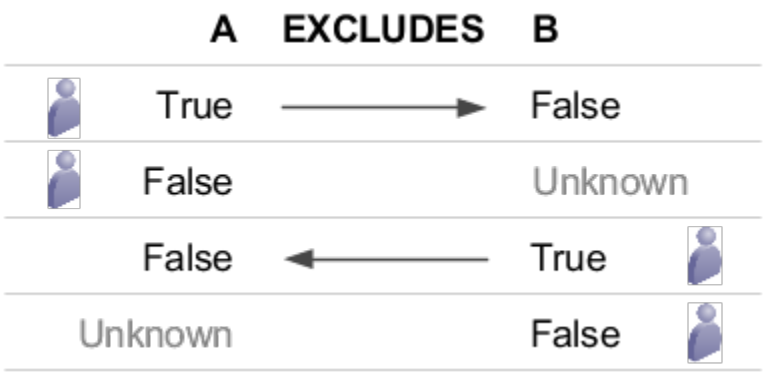
Here's how an Implies rule works.

If you:

- Select option *a*, then Configurator also selects option *b*.
- Deselect option *a*, then Configurator deselects option *b*, and you can select or deselect option *b*.
- Select option *b*, then Configurator deselects option *a*, and you can select or deselect option *a*.
- Deselect option *b*, then Configurator also deselects option *a*.

Excludes

Here's how an Excludes rule works.



If you:

- Select option *a*, then Configurator deselects option *b* and prevents you from selecting *b* because the rule excludes *b* from the configuration.
- Deselect option *a*, then option *b* becomes available and you can select or deselect option *b*.

- Select option *b*, then Configurator deselects option *a* and prevents you from selecting option *a*.
- Deselect option *b*, then option *a* becomes available and you can select or deselect option *a*.

Logic States and Rule Variables

Configurator attempts to narrow the values that you can set for each model's node at run time. You can specify the minimum value and the maximum value for each node. If you don't:

- Select or exclude an option, then Configurator doesn't bind the variable.
- Make a selection, enter a value, or click Finish and Review, then Configurator doesn't bind the variable.
- Make a selection, or if Configurator hasn't excluded an option at run time, then the value of the SelectionState attribute and the DetailedSelectionState attribute is Selectable.
- Configurator might also exclude an option through a default, search decision, or autocomplete.

Related Topics

- Overview of Creating Statement Rules
- Set the Rule's Class
- Accumulate Values
- Make Sure Options Are Compatible
- Constrain Values That Your User Can Select

Accumulate Values

You can use the Add or Subtract keywords in your statement rule to add to or subtract from a variable's numeric value.

For example, if you select the 512 MB RAM option, then allow you to add 512 to the RAM option, then display the total in the Total RAM Selected attribute.

Note

- You can accumulate a value only in the Constraint class. You can't accumulate in the Default class or the Search Decision class. If you create a statement rule in the Default class or Search Decision class, and if you use the add operator or the subtract operator in the rule's text, then Oracle Configurator displays a message when you validate the rule. For example:

  ```
  The rules of the RULE_CLASS rule class can't contain the ADDTO or SUBTRACTFROM accumulator operators.
  ```

- All the accumulation that you specify on the same target node acts as a constraint on that node.
- All the add operators and the subtract operators in the model act as a single constraint on the node. It is the sum of all expressions that you specify against it in the model minus the sum of the SubtractFrom expressions.
- If the target node is involved in any other constraints, then the equality constraint that Configurator creates as a result of the constraint's addition and subtraction expressions must be satisfied along with all the others. The equality constraint is bidirectional, so it can affect the values of the participants that are on the left side of the rule.

Guidelines

- If your model contains more than one statement rule that adds to or subtracts from the same node, and if some other referenced model contains that node, then Configurator will create a single constraint that equates the target to a sum of all the terms that you express in the individual rules in that model.
- If you set up an addition rule or a subtraction rule against a target that resides in more than one parent model in a referenced model hierarchy, then you must satisfy each of the equality constraints that your rule creates. The addition and subtraction terms don't accumulate across more than one model that's referencing another model.

Use the Add or Subtract Keywords

Each statement that accumulates a value must contain a numeric expression. You must include the ADD keyword and the TO keyword in each statement.

Consider an example where you use ADD and TO to accumulate a value:

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

Compare Decimal Values

If your rule compares two decimal values, then Configurator might not consider them as being equal even if they're in the Java double data type. Configurator sets the decimal tolerance value to 0.000000001, which is a scale of 9, and then converts the decimal comparison to a mathematical equation.

Assume you have the expression `a=b`. Configurator might convert it:

```
absolute value of a minus b is less than or equal to the tolerance
```

This equation provides an accurate result as long as the scale is less than or equal to the number of decimal digits that you specify in the tolerance.

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

Plan carefully when you write a rule that has a decimal operand and an option class or collection. Take these actions when *x* accumulates to *y* and *y* is an option class that has more than one option, or when *y* is a collection:

If	And	Then
x resolves to a decimal	Option 1 and Option 2 are each an integer	Use the Round() function on *x*.
x resolves to a decimal	Option 1 and Option 2 are each a decimal	Don't take any action on *x*.
x resolves to a decimal	Option 1 is a decimal and Option 2 is an integer	Use the Round() function on *x* to meet the most limiting restriction because Option 2 an integer.
x is an integer	Option 1 and Option 2 are each an integer	Don't take any action on *x*.

Try It

1. Create a statement rule.
2. Enter the statement rule's text. Use this syntax for the addition or subtraction operator:

   `ADD` *n* `TO` *x*

or

`SUBTRACT` *n* `FROM` *X*

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

Related Topics

- Overview of Creating Statement Rules
- Set the Rule's Class
- Specify Your Rule's Logic
- Make Sure Options Are Compatible

Make Sure Options Are Compatible

Use the Compatible keyword in a statement rule to make sure an option that your user selects is compatible with another option.

Here's an example.

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

This rule iterates over all the items of the:

- Tint option class in the Glass child model
- Color option class in the Frame child model of the Window model.

If the Stain attribute in an item that's in the Color option class equals the Stain attribute in an item that's in the Tint option class, then the color and tint are compatible with each other, and Oracle Configurator uses the color that you select to set the stain to TRUE.

Consider this pseudocode:

```
If &color.userAttrs["Paints_AG.Stain"] and &tint.userAttrs["Paints_AG.Stain"] equal Clear, and if you select the
White color, then Configurator will select the Clear stain.
```

Guidelines

- You can compare children that are in the same option class or that are in different option classes.
- You can compare options only in the Constraint class. You can't compare options in the Default class or the Search Decision class.
- You can't compare options to constrain a selection from only a subset of the participating features.
- You must include at least two iterators in the code that you write for your compatibility statement.
- Only one of the rule's participants can have a value in the Maximum Selections attribute that's greater than 1 when you compare options.
- If you create a statement rule that compares options, and then change the value of the Maximum Selections attribute in any of the rule's participants, then Configurator will display an error when you compile the model.

Syntax

- Use COMPATIBLE at the beginning of a compatibility statement.
- You must include COMPATIBLE and two or more identifiers in each compatibility statement.
- You use the same format for `COMPATIBLE. . .OF` that you use for `FOR ALL. . .IN`.
- You must include a matching identifier in the OF clause for each identifier that you include in the COMPATIBLE clause.
- Use the WHERE clause to specify the set of combinations that you need for the conditional expression.

Related Topics

- Overview of Creating Statement Rules
- Set the Rule's Class
- Specify Your Rule's Logic
- Accumulate Values

Use Explicit Statements

Use an explicit statement to express a relationship between participants that you explicitly identify.

- Use it to apply the rule to participants and to the configurator model that contains these participants.
- You must identify each node and attribute that participates in the rule.
- You specify the node's location in the model.

Use these keywords to identify an explicit statement:

- CONSTRAIN
- COMPATIBLE
- ADD...TO
- SUBTRACT...FROM

Here's an example that includes a single expression. It uses the logical IMPLIES relationship.

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

Related Topics

- Use Iterators to Apply Rules to More Than One Object
- Constrain Values That Your User Can Select

Use Iterators to Apply Rules to More Than One Object

Use an iterator to apply a rule to more than one object in your configurator model.

- Express a relationship between participants in a node.

- Apply a rule to different options of an option feature when these options have the same attributes.
- Apply a rule to different children of an option class when these children have the same attributes.
- Use the node's attributes to specify participants in the rule. This helps to apply a consistent set of rules when you frequently modify the model's structure or the model's attributes.
- Express a relationship between a combination of participants.

You can use a local variable in an iterator statement to iterate over one or more constraints.

You use these keywords.

- FOR ALL IN
- WHERE

Use More Than One Iterator

Consider an example that uses an ADD TO statement. If the value of the UDA2 attribute in feature *x* is equal to the UDA2 attribute in feature *y*, then Oracle Configurator uses the numeric value of *x* to set the values in *y* for all the options of *x* and *y*.

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

For another example, see *Make Sure Options Are Compatible*.

Use the For All, In, and Where Keywords

You can use the FOR ALL, IN, and WHERE keywords in an iterator statement.

Keyword	Description
FOR ALL IN	Use these keywords to begin the two clauses of an iterator statement: • Use the IN keyword to specify the source to use for the iteration. • You can include only a literal collection or a collection of model nodes, such as OptionsOf, in the IN clause. • All instances of a model use the same iteration.
WHERE	Use this keyword to begin the clause of an iterator statement when you need to remove iterations that don't match with the WHERE: • Your conditional expression in the WHERE clause must be static. • If you use the COLLECT operation in a WHERE and in an IN clause, then your operands must be static. • Configurator evaluates each compatibility statement starting at the top of your code, and then moves down. It evaluates the first rule it encounters, and then the second rule, and so on. It doesn't prioritize or give precedence according to an expression's AND or OR operator.

Here's an example where the result provides three values, *a*, *b*, and *c*, for one option, *d*.

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

Here's another example:

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

Note

- If the UDA3 attribute in the `a` option class contains a value that's less than 5, then the result provides as many values to the numeric feature `d` as there are children in option class `a`.
- The result contains only as many values as there are children in the criteria that you specify in the WHERE clause.
- This example uses a set of curly brackets ({}) to enclose a collection. The collection is `OptionsOf(a)`.

In both examples, Configurator expands a single statement into one or more constraints or contributions without explicitly repeating each one. You can also use the iterator variable in the left side of a statement that accumulates a value.

Use the COLLECT Operator

You can use the COLLECT operator in an iterator statement.

- Use a function that aggregates a value, such as Min(...), Max(...), Sum(...), or AnyTrue(...), to accept a collection of values as an operand.
- Use COLLECT to specify the range of values for the collection that Configurator sends to the aggregation function. In many cases, you can also use FOR ALL to achieve this purpose.
- You can use COLLECT with a complex expression and a WHERE clause to filter out elements that the collection uses as the source for its range of values.
- COLLECT is an operator that returns a collection, so you can use it inside a collection literal. Configurator flattens the collection literal, which allows it to concatenate each collection.
- You can use only one iterator with COLLECT.
- You can't use COLLECT to put dynamic variables in the IN clause or the WHERE clause because using dynamic variables in these clauses is a collection that's unknown, and it might cause a problem when you test your rule.

Here's an example that uses a COLLECT operator and a FOR ALL iterator with a single value. The value contains the maximum value of the collection for children of the `a` option feature.

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

Here's another example that has the same result.

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

If you need to limit the values that you aggregate, then you must use COLLECT. Consider an example:

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

Each iteration of the FOR ALL clause and the WHERE clause results in an error for every element of the `{Option11, Option32, OptionsOf(Feature1)}` collection that doesn't contain the UDA1 attribute. You can use the COLLECT operator to prevent the error. For example:

```
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 Operator with DISTINCT

Here's an example that uses the COLLECT operator with the DISTINCT keyword to collect distinct values from an attribute.

```
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()}
```

This example prevents your user from selecting options that have different values for the Shape attribute from the optionClass3 option class. optionClass3 has a value of zero for the Minimum Quantity attribute, and it has no limit for the Maximum Quantity attribute.

Related Topics

- Overview of Using Constraint Definition Language
- CDL Data Types
- CDL Syntax and Other Details
- CDL Expressions

Constrain Values That Your User Can Select

Use the constrain keyword to apply a constraint at run time when you manually select an option or enter a value.

- You can use the constrain keyword at the beginning of a constrain statement.
- A constrain statement uses an expression to constrain a relationship.
- You don't have to have a constrain keyword in a constrain statement.

You can include only one of these operators in each constrain statement:

- Requires
- Negates
- Implies
- Excludes

Here's an example of a constrain statement that uses the constrain keyword.

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

Here's an example that doesn't use constrain.

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

Here's an example constrain statement that uses FOR ALL...IN with an iterator.

```
CONSTRAIN F1 DEFAULTS &var1
```

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

Here's what it means in pseudocode:

```
If your user selects one option in the F1 feature, then select all other options in the F1 feature.
```

Related Topics

- Specify Your Rule's Logic
- Use Iterators to Apply Rules to More Than One Object
- Constrain Values That Your User Can Select

Specify Sequences for Defaults and Search Decisions

Autocomplete applies default rules and search decision rules according to the sequence that you specify in your rule.

Oracle Configurator will assign a sequence number and place the rule at the end of the rule's list.

Assume you have five rules in the Default class. If you create a new rule in the Default class, then Configurator sets the sequence number for the new rule to 6 and displays it at the end of the list of default rules.

You can modify the sequence that Configurator uses when it applies your rules at run time:

1. Go to the Configurator Models work area, then open the Rules tab.
2. Click **Reorder Default Rules** or click **Reorder Search Decision Rules**.
3. Click an arrow to move your rule up or down.
4. Click **Save**.

If your rule applies to more than one model or to more than one participant, then Configurator might not apply the rule in the sequence that you specify. Configurator will use the sequence of the set of rules that apply to the same combination of models or instances.

However, Configurator will apply the rules as a group after it applies all the rule's participants, and after it applies all of the rules that are in the class that you set up in various instances of the same combination of models or instances.

Related Topics

- Overview of Creating Statement Rules
- Set the Rule's Class
- Specify Your Rule's Logic
- Accumulate Values
- Make Sure Options Are Compatible

Insert Attributes Into Your Rules

You can insert various types of attributes into your rule text.

Use this format:

- ***nodePath*** is the path to a node on your model.
- ***attrName*** is the name of an attribute of the node.

You use a different procedure for each type of attribute.

Type of Attribute	Format	Example
Custom attribute that you create on the model	***nodePath*** **.userAttrs[" *attrGroupName* . *attrName* "]** Assume you create an attribute named Color: • Open the rule editor. • Expand the Structure area, then click the node, such as MY_THEATER. • In the Attributes area, click **Item**, then notice all your custom attributes. • Click your Color attribute, then click **Insert into Rule Text**. Your user's actions and your rules don't affect the attribute's value, so the attribute value is static at run time.	**'Home Theater System'.'Speaker System'.userAttrs["PhysicalAttributes.Color"**
Supplemental	***nodePath*** **.suppAttrs[" *attrName* "]** Do the same steps that you do to add a custom attribute, except click **Supplemental**. Your user's actions and your rules don't affect the attribute's value, so the attribute value is static at run time.	**'Home Theater System'.'Speaker System'.suppAttrs["color"]**
Transactional	***nodePath*** **.transAttrs[" *attrName* "]** Select the attribute in the Structure area of the rule editor, then click **Insert Into Rule Text**. Your user's actions and your rules affect the attribute's value, so the value of a transactional attribute is dynamic at run time.	**ADD 'Custom Window'.'Frame'.transAttrs["Linear Length"]/5 TO 'Custom Window'.'Frame'.'Track'.Quantity()** Here we have an example that includes a window and window track. Each track is 5 feet long. This example divides the linear length of the window by 5, then adds it to the number of tracks that the window needs.
System attribute, such as: • Name • Value • Quantity • State • Options • SelectedCount	***nodePath*** **. *attrName* ()** Do the same steps that you do to add a custom attribute, except click **Supplemental**.	**'Home Theater System'.'Speaker System'.'5.1'.Quantity()**

Guidelines for Using Transactional Attributes in Your Rule

You can map a constraint to all occurrences of a transactional attribute in a set of nodes. For example, if node x has a value of 1, then prevent the user from selecting or entering a value 1 for the Weight transactional attribute:

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

You can't use a path to reference a value for a transactional attribute. You must reference the value as a literal in a conditional expression. For example:

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

You can't use the `Selection` system attribute with a transactional attribute. This restriction applies even if the transactional attribute has a value set that Configurator displays at run time, such as the values that it displays for an option feature. However, you can use `Selection` with an option feature.

For background details, see *Transactional Attributes*.

Related Topics

- Overview of Creating Statement Rules
- Set the Rule's Class
- Specify Your Rule's Logic
- Accumulate Values
- Make Sure Options Are Compatible

Use Integer Values and Decimal Values in Configurator Rules

Consider how your configurator rule uses integer values and decimal values.

You must make sure that the result of multiplying integer values in your rule doesn't exceed Java's integer limit. Constraint Definition Language (CDL) uses Java's Double data type to store a decimal value. Double has a much higher limit than Java's Integer data type. In some cases, a rule that you create in CDL might result in a value that exceeds the integer's limit.

The result of a mathematical operation in a configurator rule depends on each operand's data type. If each of your operands are an integer, and if the value after the operator is a constant integer, or if it references an attribute that contains an integer value, then the result will be an integer.

Scenario

Consider a scenario.

```
IntegerFeature (min = 0, max = 2,147,483,647)
DecimalFeature (min = 0, max = 999,999,999,999,999)
```

Assume you have this logic in your rule.

```
IntegerFeature * 400,000 = DecimalFeature
```

You might assume that you can specify any value for IntegerFeature as long as that value remains in IntegerFeature's range, and then store the result of your multiplication in DecimalFeature.

However, IntegerFeature and the constant value of 400,000 are each an integer, 400,000 is after the operator, so the result of the multiplication must be an integer, not a decimal, and it must not exceed IntegerFeature's maximum value. At runtime, if you set IntegerFeature to a value that exceeds 5,368, you'll have an error.

Runtime Math	Result
5,368 multiplied by 400,000 equals 2,147,200,000.	2,147,200,000 is less than IntegerFeature's 2,147,483,647 maximum, so you're good to go.
5,369 multiplied by 400,000 equals 2,147,600,000.	2,147,600,000 exceeds IntegerFeature's 2,147,483,647 maximum, so you'll have an error.

Solutions

Convert an Operand to a Decimal

For example, change 400,000 to 400,000.0:

```
IntegerFeature * 400,000.0 = DecimalFeature
```

Another way is to multiply one of the operands by a very small value, such as 1.00000001. Multiplying by a small value will have a negligible affect on the result, but will help to avoid the problem. For example:

```
IntegerFeature * 1.00000001
```

Use a Decimal Feature or Supplemental Attribute

If you can't use a simple constant or a decimal value after the operator, then consider using a decimal feature instead of an integer feature.

If you can't use a decimal feature, then use a supplemental attribute that contains an integer instead of using the constant value. For example:

```
IntegerFeature * m1.suppAttrs["IntegerAttribute"] = DecimalFeature
```

Other Alternatives

Code	Description
`(IntegerFeature * SQRT(1)) * m1.suppAttrs["IntegerAttribute"] = DecimalFeature`	The SQRT function will return a decimal.
`(IntegerFeature * 1.00000001)) * m1.suppAttrs["IntegerAttribute"] =DecimalFeature`	The 1.00000001 constant is a decimal.

Modify the Sequence of Your Operands

If the solutions that we describe above don't work for you, then you can modify the sequence of your operands. The problem happens only when the value after the operator is static, so you can move that value to before the operator:

```
m1.suppAttrs["IntegerAttribute"] * IntegerFeature = DecimalFeature
```

Here's a more complex example:

```
Add &x.Quantity() * &x.suppAttrs["IntegerAttribute"] To DecimalFeature FOR
```

```
ALL &x IN {OptionClass.Options()}
```

You can rewrite it to avoid the problem:

```
Add (&x.Quantity() * SQRT(1)) * &x.suppAttrs["IntegerAttribute"] To
DecimalFeature FOR ALL &x IN {OptionClass.Options()}
```

Or, rewrite it this way:

```
Add &x.suppAttrs["IntegerAttribute"] * &x.Quantity() To DecimalFeature FOR
ALL &x IN {OptionClass.Options()}
```

The problem happens only when you multiply integer operands and the result overflows into a decimal. If the overflow doesn't happen, then the sequence that you use for the operands doesn't matter.

Use Spreadsheets to Manage Your Rules

You can use Application Development Framework Desktop Integration (ADFDI) with a Microsoft Excel spreadsheet to manage rules across models and environments.

Here are some of ways that you can manage your rules.

What You Do	Description
Search	Manage rules for a draft model. Search for the model by its name, across workspaces, or by the name of the workspace that contains the model.
Create	Create a rule for the model that you searched for, download it to an ADFDI spreadsheet, add details to the rule, then upload it.
Update	Update a rule that's in the model that you searched for, download it to an ADFDI spreadsheet, modify the rule, then upload it.

Use these attributes in the spreadsheet.

Attribute	Can Update?	Description
Model	Yes	Name of the model that contains your rule. You can't create a new model this way.
Workspace	Yes	Name of the workspace that contains a draft of the model. You can't create a new workspace this way.
Name	Yes	Name of the rule. If you enter a new name, then your upload will create a new rule in the model, workspace, and folder. You can't change the name of an existing rule this way.

Attribute	Can Update?	Description
		You can't use a rule name that already exists in another draft of the model.
Description	Yes	Description of the rule.
Folder Path	Yes	Name and path of the folder that contains your rule. If you enter a new path, then the upload creates a new folder under the top Rules folder for the model. Use this format for a new subfolder: `subfolder/sub-subfolder/...`
Type	Yes	Type of the rule. You can use only the Statement Rule type.
Rule Class	Yes	Class that the rule is currently in. To change it, select one of: • Constraint • Default • Search Decision
Enabled	Yes	Specify whether to enable the rule. Select Yes or No.
Rule Text	Yes	CDL text that defines the rule. Make sure you enclose each node name with single quotation marks (' '). If you begin your rule with a node name, then add another set of single quotation marks immediately before the name. For example: `''MyNodeName'`
Rule Status	No	Oracle Configurator validates your rule text and displays the result in the Rule Status column: • Modified • Valid • Error Configurator does the same validation on a rule that you upload and on a rule that you create in the Configurator Models work area. If validation fails, the upload will still upload your rule but it will be in an error status.

Attribute	Can Update?	Description
Status	No	ADFDI validates data in the spreadsheet and displays the result in the Status column. If the Status is Insert Failed, then click **No** when you're prompted to delete the changes from your update, then click **Status Viewer** to examine details about the error.
End Date	No	Date when Configurator will no longer apply this rule.
Error Message	No	Describes why your validation failed.

ADFDI and Configurator updates your rule after every upload.

To see you changes, you might need to close and then open the Rules page.

Try It

Assume you already imported the zCZ_CAR4DRSDN model from the Product Information Management work area, and Configurator created a workspace for it. For details, see *Manage Your Workspace*.

You now need to use ADFDI to manage rules in the CAR4DRSDN participant of your model.

1. Go to the Configurator Models work area.
2. Open the zCZ_CAR4DRSDN Model Creation Workspace.
3. In the Workspace Participants area, click **CAR4DRSDN**.
4. On the Edit Configurator Model page, click **Tasks**, then notice it has a File-Based Data Management area. You use the tasks in this area to manage your model through ADFDI.
5. Click **Manage Rules**.
6. In the save dialog that your browser displays, save the ManageStatementRules.xlsx file to any location. For example, assume you already created a folder named `C:my_rules`. Download it to the `my_rules` folder.
7. Open Windows File Explorer, navigate to the `my_rules` folder, then open ManageStatementRules.xlsx.
8. In Microsoft Excel, In the dialog that displays, click **Yes** to connect to the ADFDI plug-in. Use the same user name and password that you use to log into Oracle Applications and to access the Configurator Models work area.
9. Make sure you're in the Manage Rules worksheet, then enter the values in the first row immediately under the column headings.

Attribute	Value
Model	CAR4DRSDN
Workspace	zCZ_CAR4DRSDN (V1) Model Creation Workspace
Name	Rule 1
Description	Rule 1
Folder Path	Empty
Type	Statement Rule

Attribute	Value
Rule Class	Default
Rule Text	`''CAR4DRSDN'.'CAREXTOPT'.'CARWHEELS'.'CARWHEELS01' EXCLUDES 'CAR4DRSDN'.'CAREXTOPT'.'CARWHEELS'.'CARWHEELS02''` You must use single quotation marks (' ') to enclose this text string. Don't use double quotation marks (" ").
Enabled	Yes

For the Type, Rule Class, and Enabled attributes, click the cell, then use the down arrow to select the value.

10. Click **Create or Update**.
11. Go back to your workspace in the Configurator Models work area.
12. Click the Rules **tab**.
13. Verify that the tree includes the rules that you added from the worksheet.

For details about how to use ADFDI, see *Overview of Using Spreadsheets to Manage Supplemental Structures*.

Related Topics

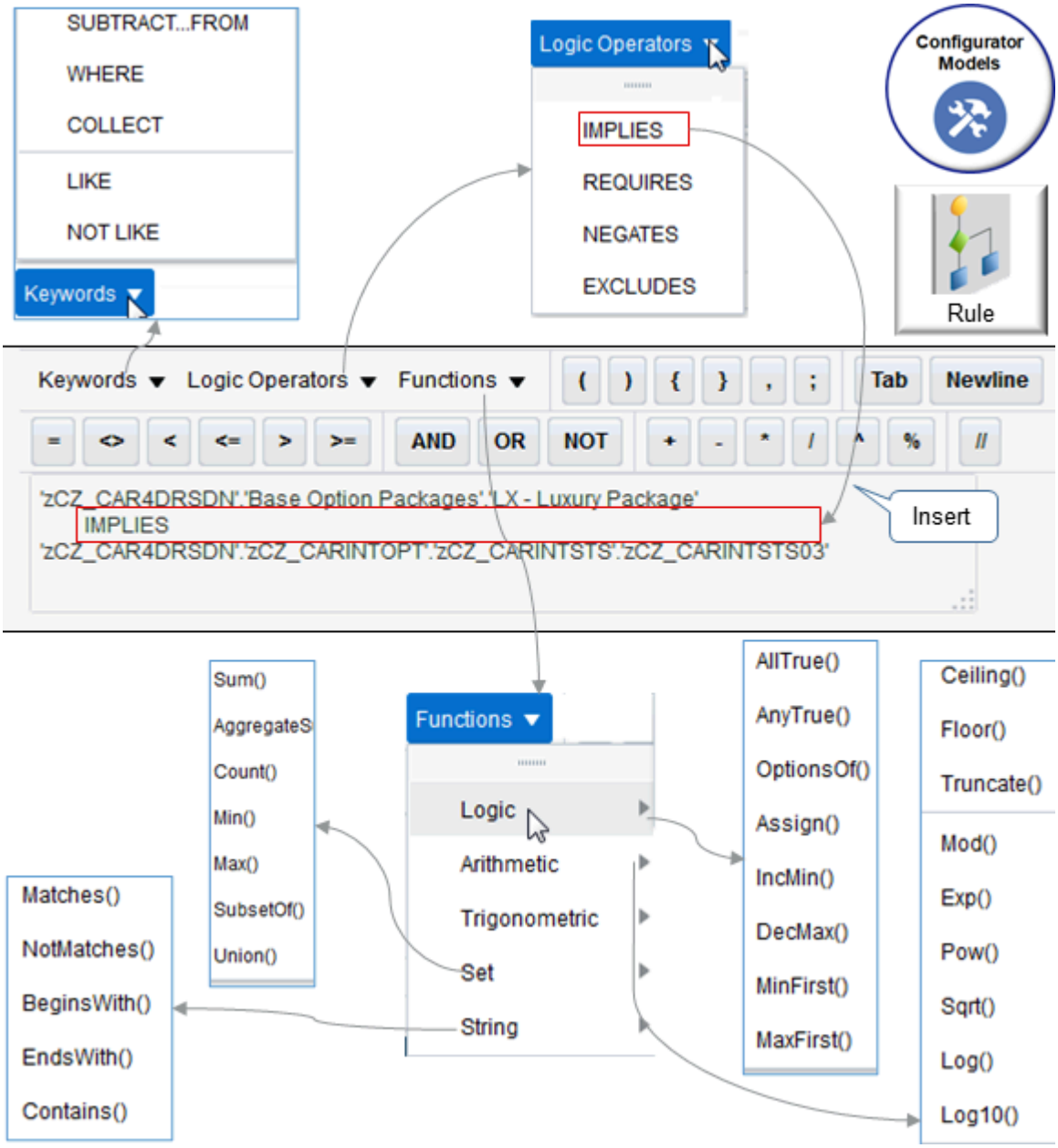
- Overview of Creating Statement Rules
- Create Statement Rules

CDL Reference

Overview of Using Constraint Definition Language

You use the Constraint Definition Language (CDL) to create your statement rules, and you use the CDL editor in the Configurator Models work area to edit your CDL code.

Here are just some of the keywords, operators, and functions that you can insert into your code.



You can specify these relationships:

- Logical
- Numeric
- Compatibility
- Comparison

Each rule contains:

- Rule Definition
- Rule Statements
- Data Types

Rule Definition

Each statement rule has a name, rule text, and other attributes, such as the rule's class.

Rule Statement

Each rule statement specifies the rule's intent, such as:

```
If the user selects option a, then add a value of 10 to the x integer feature
```

Note

- You use the semicolon (;) to separate each statement.
- You can use an explicit statement and an iterator statement.
- Oracle Configurator parses each rule statement as a token. Everything in CDL is a token, except white space characters and comments.
- Each statement includes one or more clauses.
- Each clause includes keywords and one or more expressions.
- A keyword is a predefined token that determines the CDL format. CONSTRAIN, COMPATIBLE, REQUIRES, IMPLIES, LIKE, and NOT are each an example of a keyword.
- An expression contains the operator and the operands.
- An operator is a predefined keyword, function, or character that involves the operands in each logical, functional, or mathematical operation. REQUIRES and the plus sign (+) are each an example of an operator.
- An operand can be an expression, a literal, or an identifier. The literal or identifier can be a singleton or a collection.
- Each node that you reference in a statement is a participant.
- A literal is a token that contains a specific type of data, such as Numeric, Boolean (True or False), or Text. The literal in CDL is similar to the constants that you use in other programming languages.
- An identifier identifies a node or attribute in your model. It uses a sequence of letters and digits that starts with a letter.
- An identifier can also identify a parameter. You can use it in an iterator statement. You use an ampersand (&) to prefix the name of each parameter.
- CDL uses separators to help you scan and read your code. CDL also uses separators to establish boundaries between tokens and to group them to help maintain the code's structure. Each separator is a single character, such as the semicolon between statements or the parentheses that enclose an expression.

Related Topics

- CDL Data Types
- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Expressions
- CDL Functions

CDL Functions

You can include a function in an expression.

- The function does an operation on its arguments.
- The function returns a value that you can use to evaluate the entire statement.
- You can use the value that the function returns in other parts of the expression.
- You can use the result as an operand of another operator or function. The result's data type must match the data type that you use in the other operator's or function's argument.

Note

- A function includes one or more arguments.
- You must enclose the function's arguments in parentheses.
- If your expression has more than one function, then you must use a comma to separate each function.
- An argument can be an expression.

This example includes operations for the Round function. Feature-x and Feature-y must be numeric features:

```
Round (13.4)
Round (Feature-x / Feature-y)
```

You can use these types of functions.

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

Arithmetic Functions

Function	Description
Abs(*x*)	• Takes a single number as an argument and returns the positive value of that number. For example, `Abs(-12345.6)` returns a value of 12345.6. • The argument's value can be -infinity to +infinity. • The return value can be 0 to infinity.
AggregateSum(*x*)	• You can use it in a rule that's in the Constraint, Default, or Search Decision class, but only as a subexpression.
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.
Exp(*x*)	• Takes a single number between -infinity and +infinity and returns a value between 0 and +infinity. • Returns *e* raised to the *x* power.

Function	Description
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 that's greater than 0 and less than +infinity and returns a number between -infinity and +infinity. • Returns the logarithmic value of *x*. • If *x*=0, then an error happens.
Log10(*x*)	• Takes a single number that's greater than 0 and less than +infinity and returns a number between -infinity and +infinity. • Returns the base 10 logarithmic value of *x*. • If *x*=0, then an error happens.
Max(*x,y,z...*)	• Returns the largest of its numeric arguments.
Min(*x,y,z...*)	• Returns the smallest of its numeric arguments.
Mod(*x,y*)	• Returns the remainder of *x* divided *y*, where *x* and *y* are each a number between -infinity and +infinity. • If *y* is 0, then you're attempting to divide by 0 and Oracle Configurator returns an error. • If *x*=*y*, then the result is 0. For example, `Mod(7,5)` returns 2. • This is a binary function.
Pow(*x,y*)	• Returns the result of *x* raised to the power of *y*. For example, `Pow(6,2)` returns 36. • 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. • This is a binary function.
Round(*x*)	• Takes a single decimal number as an argument and returns the nearest integer. • If the left side of a numeric rule is a decimal number, and if this number is part of a bill of materials that you import, and if that bill of materials can use a decimal quantity, then you can't use the Round(x) function. You can't use it because the right side of the equation accepts a decimal quantity, so you don't need to round the value. Round(x)is available when the item on the bill of materials accepts only integer values.
RoundDownToNearest(*x,y*)	• Rounds the first argument to the nearest smaller multiple of the second argument. For example, `RoundDownToNearest(433,75)` returns a value of 375. • *x* is a number between -infinity and +infinity • *y* is a number that's greater than 0 and less than +infinity. • It returns a number between -infinity and +infinity. • This is a binary function.
RoundToNearest(x,y)	• Returns a number between -infinity and +infinity. For example, `RoundToNearest(433,10)` returns 430.

Function	Description
	• *x* is a number between -infinity and +infinity • *y* is a number that's greater than 0 and less than +infinity. • This is a binary function.
RoundUpToNearest(x,y)	• Rounds the first argument up to the nearest multiple of the second argument. For example, `RoundUpToNearest(34.1,0.125)` returns 34.125. • *x* is a number between -infinity and +infinity, • *y* is a number that's greater than 0 and less than +infinity. • Returns a number that's between -infinity and +infinity. • This is a binary function.
Sqrt(*x*)	• Returns the square root of *x*. • Takes a single number between 0 and +infinity and returns a value between 0 and +infinity. • If you provide a negative value for *x*, then Configurator displays an error.
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.

The term *infinity* means its value has no limit. It can be infinite in the positive direction or in the negative direction.

- -infinity means that the value can be any value that's zero or smaller than zero.
- +infinity means that the value can be any value that's zero or greater than zero.

Trigonometric Functions

Function	Description
Sin(*x*)	• Takes a single number *x* that's between -infinity and +infinity and returns a value that's between -1 and +1.
ASin(*x*)	• Takes a single number that's between -1 and +1 and returns a value that's between -pi/2 and +pi/2. • Returns the arc sine of *x*. • If you provide a value for *x* that's outside the range between -1 and +1, then Configurator displays an error.
Sinh(*x*)	• Returns the hyperbolic sine of *x* in radians. • Takes a single number that's between -infinity and +infinity and returns a value that's between -1 and +infinity. • If you provide a negative value for *x*, and if the result exceeds the double, then Configurator displays an error. For example, `sinh(-99)` will work but `sinh(999)` will create an error.
Cos(*x*)	• Takes a single number that's between -infinity and +infinity and returns a value that's between -1 and +1. • Returns the cosine of x.

Function	Description
ACos(*x*)	• Takes a single number that's between -1 and +1 and returns a value that's between 0 and pi. • ACos(x) returns the arc cosine of *x*. • If you provide a negative value for *x*, or if you provide a value for *x* that's outside the range between -1 and +1, then Configurator displays an error.
Cosh(*x*)	• Takes a single number that's between -infinity and +infinity and returns a value that's between -infinity and +infinity. • Returns the hyperbolic cosine of *x* in radians. • If you provide a negative value for *x*, and if *x* exceeds the maximum value of a double, then Configurator displays an error. For example, **`cosh(-200)`** will work but **`cosh(-2000)`** will create an error.
Tan(*x*)	• Takes a single number *x* that's between -infinity and +infinity and returns a value that's between -infinity and +infinity.
ATan(*x*)	• Takes a single number that's between -infinity and +infinity and returns a value that's 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* that's between -infinity and +infinity and returns a value that's between -1 and +1.

Logical Functions

Function	Description
AllTrue	• A logical AND expression. • Accepts one or more logical values or expressions. • If all of the arguments are true, then it returns a value of **`true`**. • If all of the arguments are false, then it returns a value of **`false`**. • If all of the arguments aren't true, and if all of the arguments aren't false, then it returns a value of **`unknown`**.
AnyTrue	• A logical OR expression. • Accepts one or more logical values or expressions. • If any of the arguments are true, then it returns a value of **`true`**. • If all of the arguments are false, then it returns a value of **`false`**. • If all of the arguments aren't true, and if all of the arguments aren't false, then it returns a value of **`unknown`**.
Not	• Accepts a single logical value or expression. • If the argument is false or unknown, then it returns a value of **`true`**. • If the argument is true, then it returns a value of **`unknown`**.

Text Functions

Each of these text functions compares two operands of text literals.

Function	Description
BeginsWith	If the first character of the first operand matches the first character of the second operand, then it returns `true`.
Contains	If the value in the first operand contains the value in the second operand, then it returns `true`.
EndsWith	If the last character of the first operand matches the last character of the second operand, then it returns `true`.
Equals	If the value in the first operand equals the value in the second operand, then it returns `true`.
Matches	If the value in the first operand matches the value in the second operand, then it returns `true`.
NotEquals	If the value in the first operand doesn't equal the value in the second operand, then it returns `true`.

Note

- Don't use a text function in the body of a constraint or statement that accumulates a value unless it evaluates to a constant string. Configurator will validate this condition.
- You can use a text function can with static text. For example, in the WHERE clause of an iterator.

Option Function

Function	Description
OptionsOf	Takes the name of an option class or a feature as an argument and returns its options.

Related Topics

- CDL Data Types
- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Expressions
- CDL Operators

CDL Expressions

You use an expression in your statement rule. It has two operands and an operator, or two functions and their arguments.

Here's an example of a simple mathematical expression. It has the `2` operand, the `*` (multiplication) operator, and the `frame.border` operand:

```
2 * frame.border
```

This example has the `window.frame.width` operand, the - (minus) operator, and the `frame.border` operand:

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

This example has a Window model. You need to calculate the size of the glass to put in a window frame. You need a gap of one 1/2 inch between the glass and frame, and the frame border is 1 inch wide:

```
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;
```

This example uses a BOOLEAN value:

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

This example uses an INTEGER or DECIMAL value:

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

Related Topics

- CDL Data Types
- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Functions
- CDL Operators

CDL Operators

An operator is a predefined token that consists of Unicode characters. You use it to specify the operation to perform at run time between operands.

Operators That You Can Use

Operator Type	Operator	Description
Logical	AND	• Requires two operands. • If each operand is true, then it returns `true`.
Logical	OR	• Requires two operands. • If either operand is true, then it returns `true`.
Logical	NOT	• Requires one operand. • If the operand is true, then it returns `false`.

Operator Type	Operator	Description
		• If the operand is false, then it returns **true**.
Logical	REQUIRES	• Requires two operands.
Logical	IMPLIES	• Requires two operands,
Logical	EXCLUDES	• Requires two operands.
Logical	NEGATES	• Requires two operands.
Logical and Comparison	LIKE	• Requires two text literal operands. • If the values in each operand match each other, then it returns **true**. If not, it returns **false**.
Logical and Comparison	NOT LIKE	• Requires two text literal operands. • If the values in each operand don't match each other, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	=	• Requires two operands. • If the values in each operand match each other, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	>	• Requires two operands. • If the value in the first operand is greater than the value in the second operand, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	<	• Requires two operands. • If the value in the first operand is less than the value in the second operand, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	<>	• Requires two operands. • If the value in the first operand is equal to the value in the second operand, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	<=	• Requires two operands. • If the value in the first operand is less than or equal to the value in the second operand, then it returns **true**. If not, it returns **false**.
Logical, Arithmetic, and Comparison	>=	• Requires two operands. • If the value in the first operand is greater than or equal to the value in the second

Operator Type	Operator	Description
		operand, then it returns **`true`**. If not, it returns **`false`**.
Arithmetic	*	• Performs multiplication on numeric operands.
Arithmetic	/	• Performs 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.
Text	+	• Concatenates text strings.
Other	Assign(node)	• Assigns a node to a point in a sequence. • If you specify the Domain Ordering attribute on the node's details page, then Oracle Configurator assigns the node. If you don't specify this attribute, then Configurator implicitly assigns the node according to the operator type. • You can use this operation only with a rule that's in the Default class or the Search Decision class.
Other	IncMin()	• Similar to the **`Assign(node)`** operator, but **`IncMin()`** overrides any explicit or implicit operations that assign the node and instead attempts to use a binary search with increasing minimum to assign the node. • You can use this operator with an integer or decimal, including items and options that have a quantity. • If you don't explicitly specify a system attribute, such as State, Quantity, or Value, then Configurator applies this operation to the node's default system attribute. • If you use this operation on an item, then you can specify the RelativeQuantity attribute as an alternative. • You can use this operation only with a rule that's in the Default class or the Search Decision class.
Other	( )	• Groups an expression.

Operator Type	Operator	Description
Other	,	• Separates arguments.
Other	.	• References a part of a model.
Other	-	• Changes a positive value to a negative value or changes a negative value to a positive value.

Operator Data Types

Operator	Data Type
Arithmetic	Integer Decimal
Logical	Boolean
Comparison	Boolean

You can use the result of each operator as an operand of another operator. Each of these operators must use the same data type. For example:

- If operator *x* is a decimal, and if operator *y* is a decimal, then you can use the results of *x* as an operand of *y*.
- If operator *x* is a decimal, and if operator *y* isn't a decimal, then you can't use the results of *x* as an operand of *y*.

Operator Precedence

Configurator uses this precedence when it processes operators:

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

Precedence	Operator	Description
10 (left)	`DEFAULTS, EXCLUDES, NEGATES, IMPLIES, REQUIRES`	Logic operators

If two or more operators have the same precedence, then Configurator processes them from left to right as they appear in the rule.

LIKE and NOT LIKE Operators

You can include the LIKE operator and the NOT LIKE operator to compare text.

- You can use them only on static context. For example, you can use the WHERE clause of an iterator.
- You can use LIKE or NOT LIKE with a run time participant only if the operator evaluates to a constant string. Configurator validates this condition when you save the model.

Use LIKE to Get a Boolean Value

Consider this example:

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

Here's how that works:

If the Text of a.attr Contains	Then Configurator Returns
`eig`, such as `a.attr =weight` or `eight`	`true`
`Rein`	`false`

In this example, if the user selects option A and option B, and if the value of their attributes is A1B1, then the rule selects all options in C:

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

If the user selects option A and option B, and if the value of their attributes isn't A1B1, then the rule selects all options in C:

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

Concatenate Text Operator

You can use the + (plus sign) to concatenate text.

- You can use it only with static text. For example, you can use it in an iterator's WHERE clause.
- You use it in the body of a constraint or contributor statement only if it evaluates to a constant string. Configurator validates this requirement when you save the model.

Collect Operator

You can use the Collect keyword to create a collection and then provide the collection to an operator that can use 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})`.

Related Topics

- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Expressions
- CDL Functions

CDL Data Types

Get details about the data types that you can use in CDL.

You can use these data types:

- Integer
- Decimal
- Boolean
- Text
- Node

A variable's data type might not work with the type that Oracle Configurator expects. You can't explicitly convert a data type to another data type, but Configurator can implicitly convert a data type. Here are the data types that Configurator will implicitly convert.

If Your Data Type Is	Then Configurator Implicitly Converts It To
Integer	Decimal
In one of these types of nodes: • Standard Item • Option Class • Model • Option Feature • Option • Boolean Feature	Boolean Integer Decimal Node
In an Integer Feature node	Integer Decimal
In a Decimal Feature node	Decimal

If your rule has the wrong data type, then Configurator will display an error message. An Invalid Collection message means Configurator can't implicitly convert your data types so they work correctly with each other.

You can use the Text data type only with static text. You can't use a text literal, reference, or expression with the Constrain, Add, or Subtract keywords. Configurator validates this usage when you save the model.

Related Topics

- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Expressions
- CDL Functions
- CDL Operators

CDL Syntax and Other Details

Get more details to help you code in CDL, such as details about elements, notation, separators, and terminology.

CDL Elements

Element	Description
Comment	Include comments in your rule at your discretion to help explain the rule. Comments aren't tokens so Oracle Configurator ignores them.
White space	White space includes spaces, line feeds, and carriage returns. Use white space to improve your code's readability. White space isn't a token so Configurator ignores it.
Case Sensitivity	These are case sensitive: • Values that identify objects in a model. • Parameters. Also, you can't enclose a parameter in quotes, such as (" ") or (' '). • Text literals. These aren't case sensitive: • Keywords • Keyword operators • The constants E and PI and the scientific E • The keywords TRUE and FALSE • All keywords, constant literals, and so on
Quotation Mark	You must enclose these in single quotation marks (' '):

Element	Description
	• The name of a node that contains a white space. If the name doesn't have a white space, then you don't need to enclose it. • Text that Configurator might interpret as a keyword or operator.

Notation

Symbol	Description
-- or //	Use a double hyphen (--) or a double slash (//) to begin a single line comment that extends to the end of the line.
/* */	Use a slash asterisk (/*) to start a comment that spans more than one line of code. Use an asterisk slash (*/) to end a comment that spans more than one line of code.
&lower case	Use lower case text and prefix it with an ampersand to identify the name of a parameter or to identify the iterator for a local variable.
UPPER CASE	Use upper case text for a keyword, for the name of a predefined variable, or for the name of a parameter.
Mixed Case	Use mixed case for the name of node that you create or for the name of a rule that you create.
;	Use a semicolon to indicate the end of one statement and the beginning of the next statement.

Separators

You use a separator to separate keywords and expressions from each other. They help you maintain each of your token's structure.

Separator	Description
(	The open parenthesis indicates the beginning of a function argument or the beginning of an expression.
)	The close parenthesis indicates the end of a function argument or the end of an expression.
,	The comma separates arguments or elements.
;	The semicolon separates statements.
.	The dot separates identifiers.

Terminology

Term	Description
Clause	A segment of a statement rule that includes a keyword and an expression.
Collection	A set of operands, inside a parentheses and separated by commas.

Term	Description
explicit statement	Statement that creates a relationship between participants that you explicitly reference. You apply the rule to the participants and the model that contains those participants.
expression	A subset of the statement that contains operators and operands.
formal identifier	A variable that you define in the scope of an iterator statement to represent an iterating identifier.
iterator statement	An iterator is a statement that works like a query to iterate, or repeat, over one or more relations or constraints.
nonterminal	The kind of symbols that we use to present CDL that includes names.
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 isn't 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 isn't within a collection.
statement	The entire sentence that expresses the rule's intent. A CDL rule can include more than one statement, and each statement can include clauses that contain expressions separated by semicolons.
terminal	The kind of symbols that we use to present CDL that includes the names, characters, or literal strings of tokens.
Token	The result of translating characters into a format that Configurator can use. All text except white space characters and comments are tokens.
unicode	A 16 bit, character encoding that allows you use to characters from Western European, Eastern European, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi and all other major languages, and encode them in a single character set.

Style That This Book Uses

Style	Description
Italics	Italics text represents a variable. This book doesn't use angle brackets (< >), square brackets ([ ]), or curly brackets ({ }) to indicate a variable. We use this style to avoid confusion. Code often uses these brackets as part of the syntax for other reasons.
[]	A set of square brackets encloses an optional clause.
>	Represents the Microsoft DOS command line prompt.
$	Represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX.
%	The percent sign by itself represents the UNIX command line prompt.
name()	A set of parentheses at the end of text means the text contains the names of a method or a function. For example, the text Round(), means the Round function. We don't use this style in the code itself or in code examples in this book. If you see a set of parentheses added at the end of text in code, then that means its part of the code.

Style	Description
. . .	A vertical ellipsis in code represents code that we have removed from the example because it isn't relevant to the topic.
...	A horizontal ellipsis represents statements or commands that we have removed from the example because they aren't relevant to the topic.

For formatting and print purposes, the example CDL code in this book doesn't always have a carriage return. If it doesn't, assume the carriage return is implied. When you code your CDL, you must press the Enter key at the end of each line to insert a carriage return.

Related Topics

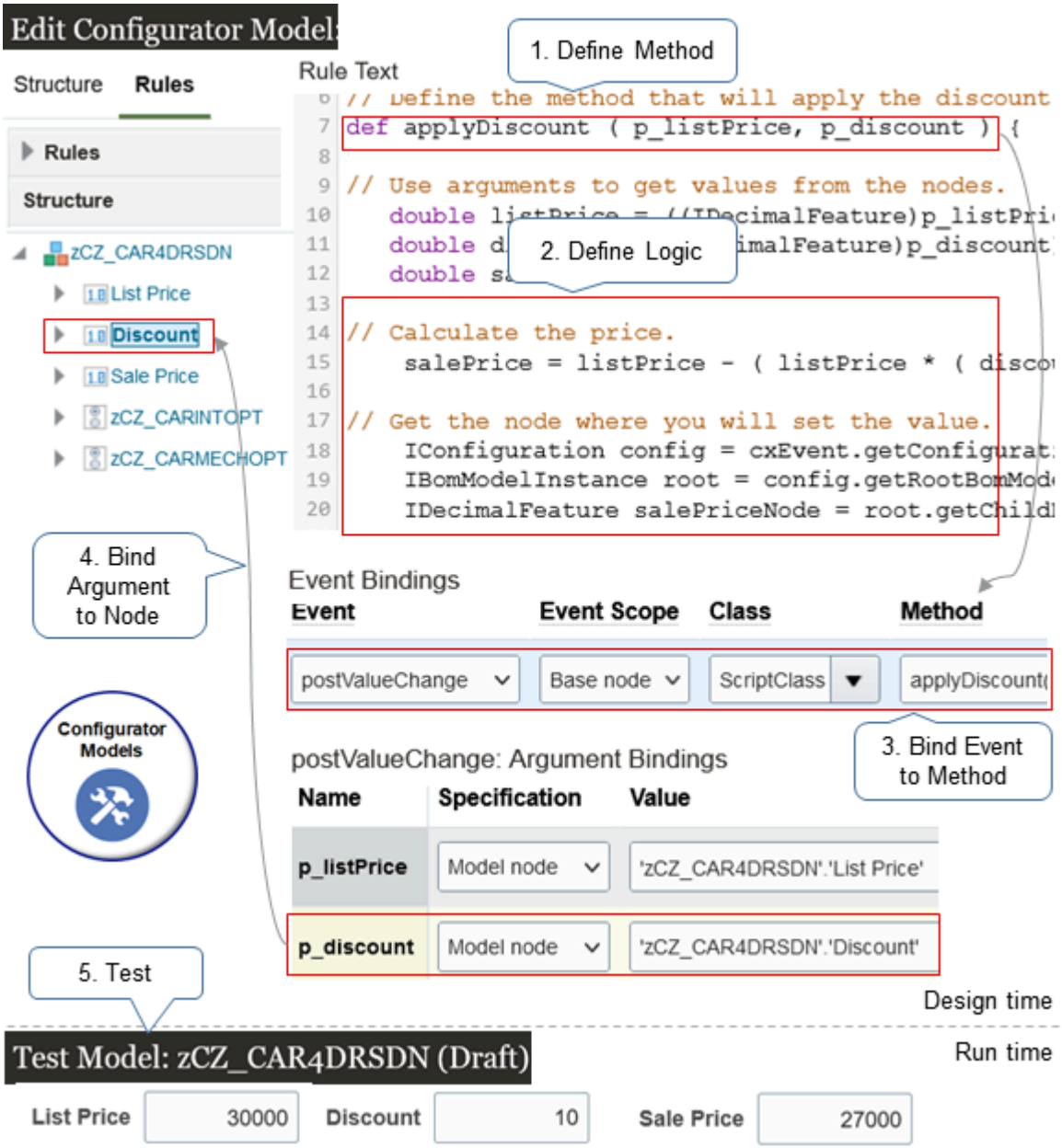
- Use Iterators to Apply Rules to More Than One Object
- CDL Syntax and Other Details
- CDL Expressions
- CDL Functions
- CDL Operators

Extension Rules

Overview of Creating Extension Rules

Use the Groovy scripting language to write your own, custom extension rule.

Consider an example where you add the zCZ_CAR4DRSDN model to a new workspace. You add features and an extension rule that allows your user to specify the list price and the discount on the car, then automatically adjust the sale price according to the discount that you enter.



Here are some of the important tasks that you typically do.

Step	Task	Description
1	Define a method.	Define a Groovy method that controls your rule's logic. For example, create a method named **`applyDiscount`**, and include arguments that you will use to store pricing details, such as **`p_listPrice`** and **`p_discount`**.
2	Define the rule's logic.	Specify the calculations that your rule will do, such as using the list price and the discount to

Step	Task	Description
		calculate the sale price, and specify the nodes in the model that are part of the calculation.
3	Bind an event to your method.	Create a relationship between your Groovy method and an event. Oracle Configurator will call the method when that event happens. For example, if your user changes the value in a decimal node, such as the discount, then Configurator raises the postValueChange event, and will use that event to call the applyDiscount method that applies the discount. You must bind at least one event to your rule.
4	Bind the method's arguments to a node.	Create a relationship between each argument and the source that Configurator will use to get the argument's value. For example, the **`applyDiscount`** discount method includes the **`p_discount`** argument. You bind **`p_discount`** to the node where you enter the discount.
5	Test.	Test your rule in a simulated run time environment before you release the rule.

This is a simple example that you can examine to help understand how to create an extension rule. Users don't typically specify pricing. Configurator usually gets pricing details from Oracle Pricing.

Related Topics

- Guidelines for Writing Extension Rules
- Create an Extension Rule
- Guidelines on How to Be Groovy

Guidelines for Writing Extension Rules

Use these guidelines to help you write an extension rule.

Identify Your Event

An event is an action or condition that happens on your model at run time. For example, you modify the value of an attribute. Oracle Configurator can run your extension rule when a predefined event happens at run time. You can use these events to start the rule.

Use This Event	To Call Your Extension Rule Immediately After the User
postConfigInit	Starts the session. Use this event to set values in your model as soon as your user starts configuring the item.

Use This Event	To Call Your Extension Rule Immediately After the User
postConfigSave	Clicks **Save**.
postValueChange	Changes a value or makes a selection in a node. For example, you modify the value of a decimal feature.
preAutoComplete	Clicks **Finish** or clicks **Finish and Review**. This event happens immediately before Configurator does the autocomplete at run time. Use this event to set values in your model before you review the configuration, or before you save and finish the configuration.

Set the Base Node

You must specify at least one node as the base node.

If the base node contains an item, and if you end date that node, then Configurator won't run the extension rule on or after that end date. To see the end date for the rule, go to the Edit Configurator Model page, click Rules, then examine the End Date attribute in the Details section.

Configurator automatically adds the path that locates the node in the model when you click Set as Base Node.

- It adds this path to the Base Node attribute in the node's Detail section.
- You can also manually enter the path in the Base Node attribute. If you do, and if the value that you enter contains a space character ( ), then you must use single quotation marks (' ') to enclose the entire text that you enter.
- If you need to copy the path for some reason, click the Structure tab. In the Details area, copy the value in the Path attribute, then navigate back to the node's detail area and paste the value into the Base Node attribute. If the node's name is unique in the model, then you can type the node name into the Base Node attribute, save the model, and Configurator will add the entire path for you.

If the model can have more than one instance of the base node, then Configurator enables the Instantiation Scope attribute, and sets its default value to Instance. The instantiation scope is a filter that determines how to run the rule. It determines:

- When to run the rule
- Whether to create a separate instance of the rule for each instance of the base node, or to create a single instance of the rule for all of the base node's instances

Enter the Rule's Text

- Save the rule while you work on the rule's text, even if it isn't complete or correct.
- Click **Undo** or click **Redo** to undo or redo text edits.
- Use **Next** and **Previous** to navigate text.
- Click **Find and Replace** to find and replace text.

- Enter a value in **Go to Line**, then click the **magnifying glass** to go to a line.
- Use the scroll bars, Click **Maximize**, and click **Collapse Pane** to help navigate your text.
- Enter unformatted text, and the text editor will format it for you, such as adding color to help identify keywords, literals, and comments.
- Green brackets and parentheses means you correctly closed the phrase, which is good. Red means you haven't, and that's bad. Fix your code until its green.

Bind Your Events

- Click **Validate**, and Configurator will populate the values that you can select in the Class attribute and the Method attribute with the classes and methods that you have defined in your rule's text. Click Validate at any time, and Configurator will update these values according to the classes and methods that you have added or removed from your script.
- If you haven't clicked **Validate** at least one time, then the lists for these attributes are empty.
- Set these attributes when you bind the event:

Attribute	Description
Event	Select the event that will start the method.
Event Scope	Create a filter that specifies when to apply the event binding. Set it to: o **Global**. Use the binding when you modify a value on any node. o **Base Node**. Use the binding only when you modify a value on the base node. If the scope applies to any extension rule in the model, then Configurator will run all the methods that you have bound to that event.
Class	Select the class that contains the method you're binding. If you didn't define your method in a class, then select ScriptClass.
Method	Select the method that you're binding to the event. If you didn't define your method in a class, then select `run()`, which runs all methods that aren't in a class.

- You can bind more than one method in an extension rule. If you do, then the sequence that you use to create them doesn't affect how Configurator runs the rule.
- You must bind at least one event to each rule.

Bind Your Arguments

- Configurator automatically adds each method argument that you define in the rule's text to the Argument Bindings area.
- Configurator lists each argument with its own Name, Specification, and Value attributes in the Argument Bindings area.

- Use the Specification attribute to specify how the method gets the value for the argument.

Specification	What You Do
Literal	Enter a numeric or string value, without quotation marks.
Model Node	a. Make sure you select the argument's **row** in the Argument Bindings area. If you don't do this, then Configurator won't populate the Value attribute on this row. b. Expand the **Structure** section on the Rules tab. c. Expand the **model** in the Structure section. d. Right-click the **node** that will provide the value for the argument, then click **Set as Argument Value**. e. Verify that Configurator automatically populated the Value attribute on this row. If the node's name is unique in the model, then you can enter the node name directly into the Model Node attribute, and Configurator will automatically add the path to the Model Node attribute when you save the model.

- Make sure you bind each argument that Configurator displays in the Argument Bindings section.

Validate Your Rule

- Click **Validate** to validate your rule's text and your bindings. Note that this validation is different than the validation that you do for a statement rule. The statement rule validates only the rule's text.
- Click Validate at any time to validate your code. Validating doesn't save or affect your work.

Here are some of the more common validations that Configurator does for you. It makes sure:

- The rule's text isn't empty.
- Your Groovy syntax is correct.
- Your Groovy script doesn't have any annotations.
- The rule has at least one base node.
- The rule binds at least one event.
- If the Groovy method that you bind to an event has arguments, then validation makes sure you bind all the arguments.

If you modify an event binding, then validation makes sure your modification is correct. If it isn't correct, then validation adds an error icon to the event. For example, modifying the method's name makes the binding incorrect. If you have an incorrect binding, the Configurator ignores it when you test the model.

More

If you modify the name of a node that the rule references, then Configurator:

- Displays a warning that identifies the original name of the node that you modified.
- Changes the rule's Status attribute to Error.

You must change the node's name to its original value, or update your script so it uses the new node name.

You can test a model that contains a rule that isn't valid. You can't release a model that contains a rule that isn't valid. You can also disable a rule that isn't valid, and then release the model.

Related Topics

- Create an Extension Rule
- Guidelines on How to Be Groovy

Guidelines on How to Be Groovy

You use the Groovy scripting language when you write your rule text, and you use interfaces in your script to import objects from your configurator model so your script can access them.

Be Groovy

Here are some important Groovy features that you can use when you write your extension rule.

- Use the `def` keyword to declare variables without declaring the type of variable.
- Define methods and variables outside of a class and use them anywhere in your script. Groovy will include them in its `ScriptClass` class, and will run them in the `run` method.

Define Methods in Classes When You	Don't Define Methods in Classes When You
○ Prefer to use inheritance and encapsulation in your code. ○ Prefer to build more complex logic, such as you might normally do in the Java programming language. ○ Define a lot of methods. Adding them to a class makes it easier to select them from the Class attribute and the Method attribute when you bind your events.	Need to write and test code quickly. You can simply bind every event to `ScriptClass` and `run`.

For more, see *Groovy*.

Access Configurator Objects

You use various interfaces to access objects during a runtime configuration. Here are some of the objects that you typically use.

What Your Code Needs to Do	Interface That You Use to Access It	Code That You Use to Create the Object	Example That Uses the Object
Get the event that you specify when you bind the method.	`ICXEvent`	No code needed. The script automatically creates the `cxEvent` object when you run the rule.	Get the current configuration that the event references: `IConfiguration config = cxEvent.getConfiguration()`
Get the rule's base node.	`ICXEvent`	`def baseNode = cxEvent.getBaseNode()`	Test whether your user selected the base node: `if (baseNode.isSelected()) ...`

What Your Code Needs to Do	Interface That You Use to Access It	Code That You Use to Create the Object	Example That Uses the Object
Get the configuration that's active during the runtime session.	`IConfiguration`	`IConfiguration config = cxEvent.getConfiguration()`	Get the root node of the configuration that's active during the runtime session: `def root = config.getRootBomModel()`
Get the root node that's active during the runtime session.	`IBomModelInstance`	`IBomModelInstance root = config.getRootBomModel()`	Get a child node of the model. Use the item name to search the model, such as the AS54888 item, and start searching at the root node: `def childItem = root.getChildByName("AS54888")`

Oracle Configurator automatically initializes each object that's involved with the event that you bind. This object is an instance of the `ICXEvent` interface named `cxEvent`.

Manage Node Values

You can get and set the values for various objects on each node. Here are some examples.

Type of Feature	Java Interface	Code That Gets the Value	Code That Sets the Value
Integer	IIntegerFeature	`int getValue()`	`void setIntValue(int value)`
Decimal	IDecimalFeature	`double getValue()`	`void setDecimalValue(double value)`
Option	IOptionFeature	`IOption getSelectedOption()`	`void select()`
Boolean	IBooleanFeature	`boolean isSelected()`	`void toggle()`

Many more interfaces are available to meet your specific needs. For example:

- Get and set logic states
- Access properties
- Access options
- Override contradictions
- Manage logical contradictions
- Manage exceptions

The `oracle.apps.scm.configurator.runtime.core` package contains the classes that you can use to interact with Configurator. For details, go to *https://docs.oracle.com/*, then search for Java API Reference for Oracle Configurator.

Java API Reference for Oracle Configurator only includes objects that Configurator supports in an extension rule. Some Groovy classes contain objects and interfaces that Configurator doesn't support. If your script references a class or

interface that Configurator doesn't support, then you will encounter an error when you validate your code. The error identifies the reference that isn't correct.

Groovy can determine the feature's data type at run time, so you don't need to declare the type in your script.

Related Topics

- Guidelines for Writing Extension Rules
- Create an Extension Rule

Create an Extension Rule

Use Groovy to code your extension, bind a Groovy method to an event, bind the method's arguments to a source, then test your work.

Summary of the Setup

1. Create your model and supplemental features.
2. Create the extension rule.
3. Bind your groovy method to an event.
4. Bind your method's argument to a source.
5. Test your model.

Assume you need to create a model named zCZ_CAR4DRSDN, allow you to specify a value of 20,000 to 60,000 for the list price, then apply a 0% to 10% discount.

This topic includes example values. You might use different values, depending on your business requirements.

Create Your Model and Supplemental Features

1. Go to the Configurator Models work area.
2. Create a new workspace and set the Effective Start Date to tomorrow. For details, see *Manage Your Workspace*.
3. On the Manage Workspaces page, in the search results, click your new **workspace**.
4. On the Workspace page, click **Actions > Select and Add > Models**.
5. In the dialog that displays, search for your model, select it, then click **OK**.

Attribute	Value
Model	zCZ_CAR4DRSDN

6. In the Workspace Participants area, click **zCZ_CAR4DRSDN**.
7. On the Edit Configurator Model page, select the zCZ_CAR4DRSDN model's **root node**, then click **Actions > Create > Decimal Feature**.
8. In the Dialog that displays, set the values.

Attribute	Value
Name	List Price
Minimum	20,000

Attribute	Value
Maximum	60,000
Domain Ordering	System Default

9. Click **Apply and Create Another**, then set the values.

Attribute	Value
Name	Discount
Minimum	0
Maximum	10
Domain Ordering	System Default

10. Click **Apply and Create Another**, then set the values.

Attribute	Value
Name	Sale Price
Minimum	0
Maximum	100,000
Domain Ordering	System Default

11. Click **OK**, then click **Save**.

Create the Extension Rule

Create a rule that applies the discount to the list price, then puts the result in the sale price.

1. On the Edit Configurator Model page, click **Rules**.
2. Click **Actions > Create > Extension Rule**.
3. In the dialog that displays, set the Name attribute to Apply Discount, then click **OK**.
4. In the Structure area, expand the **tree**.
5. Right-click the Discount **node**, then click **Set as Base Node**.
6. In the Rule Text area, enter this Groovy script:

```
// Import objects from the configurator model.
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 method that will apply the discount.
def applyDiscount ( p_listPrice, p_discount ) {

// Use arguments to get values from the nodes.
 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 where you will set the value.
 IConfiguration config = cxEvent.getConfiguration()
 IBomModelInstance root = config.getRootBomModel()
 IDecimalFeature salePriceNode = root.getChildByName("Sale Price")

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

7. Click **Save**, then click **Validate**.

Bind Your Method to an Event

1. In the Event Bindings area, click **Create**, then set the values.

Attribute	Value
Event	postValueChange
Event Scope	Base Node
Class	ScriptClass
Method	applyDiscount Note that the Method attribute also displays applyDiscount's arguments, such as `p_listPrice` and `p_discount`.

2. Click **Save**.

Bind Your Method's Argument to a Source

1. In the Argument Bindings area, click the **row** that has this value.

Name	Specification	Value
p_listPrice	Model Node	Empty

2. In the Structure area, right-click the List Price **node**, then click **Set as Argument Value**.
3. In the Argument Bindings area, verify that the row now has this value.

Name	Specification	Value
p_listPrice	Model Node	'zCZ_CAR4DRSDN'.'List Price'

4. In the Argument Bindings area, click the **row** that has this value.

Name	Specification	Value
p_discount	Model Node	Empty

5. In the Structure area, right-click the Discount **node**, then click **Set as Argument Value**.
6. In the Argument Bindings area, verify that the row now has this value.

Name	Specification	Value
p_discount	Model Node	'zCZ_CAR4DRSDN'.'Discount'

7. Click **Save**, click **Validate**, then click **Save and Compile**.

Test Your Model

1. Click **Test Model**.
2. In the dialog that displays, set the values, then click **OK**.

Attribute	Value
User Interface	Default
Root Quantity	1

3. On the Test Model page, verify that the page displays your new decimal features and displays them in the same sequence that you used when you added them to your model:
   - List Price
   - Discount
   - Sale Price
4. In the List Price attribute, enter `30,000`.
5. Verify that you can only enter values starting with 20,000 and up to 60,000. These are the minimum and maximum values that you specified when you created the List Price decimal feature.
6. Notice that the Sale Price attribute is empty.
7. In the Discount attribute, enter `10`.
8. Verify that you can only enter values starting with 0 and up to 10. These are the minimum and maximum values that you specified when you created the Discount decimal feature.
9. Click **Enter** in the Discount attribute, then notice that Oracle Configurator updates the value in the Sale Price attribute from empty to 27,000.

   The discount is 10% and the list price is 30,000. 10% of 30,000 equals 3,000, and 30,000 minus 3,000 equals 27,000.
10. Enter `20` in the Discount attribute, then notice that the Sale Price attribute now contains 24,000.

    Configurator applies this change because you bound the extension rule to the event that happens when you change the value of the Discount attribute.
11. Enter `20,000` in the List Price attribute.

Notice that Configurator doesn't update the Sale Price. That's because you didn't bind the rule to an event that happens when you change the List Price's value.

In an actual implementation, you would probably also bind the rule to the List Price. We don't do it here to illustrate how the logic works. To test your knowledge, you can set up that bind and see if it works.

12. Click **Finish**.

Related Topics

- Guidelines for Writing Extension Rules
- Guidelines on How to Be Groovy

Get External Data for Your Extension Rules

You can use an extension rule to call a web service that's outside of Oracle Fusion Applications.

For example, its probably more efficient to get data from a web service instead of maintaining it in your model, such as to get data about product specifications from your suppliers, or to get data from a large set of government regulations.

Summary of the Setup

1. Connect the web service.
2. Create the extension rule.

Connect the Web Service

1. Sign into Oracle Applications with administrator privileges.
2. Go to the Setup and Maintenance work area, then click **Tasks > Search**.
3. Search for, then open the Manage External Service Details for Extensions task.
4. On the Manage Connector Details page, click **Actions > Add Row**, then set the values.

Attribute	Description
Target System	Select the application that will provide the data. You can select an application that you have registered as a trading community partner, or select an Oracle application.
Connector Name	Enter the name that you will use to refer to the service from your extension rule. For this example, enter `VisionCorp`.
Connector URL	Enter the URL to the external web service. For example, enter `http://visioncorp04.com:7011/services/VisionWebServicePort`.
User Name	If the service that you're calling requires a user name, then enter it.

Attribute	Description
Password	If the service that you're calling requires a password, then enter it.
Invocation Mode	Select Synchronous.

5. Click **Save and Close**.

Create the Extension Rule

Assume you already created a Groovy script and you now need to call the web service in that script.

1. Sign into Oracle Applications with the privileges that you need to manage Oracle Configurator.
2. Go to the Configurator Models work area.
3. Open the model that has your script.
4. On the Edit Configurator Model page, click **Rules**, the open your extension rule for editing.
5. In the Structure area, select a **base node**, such as the root node of the model.
6. In the Rule Text area, enter the Groovy script that will call the web service.

```
import oracle.apps.scm.configurator.runtime.core.IRuntimeNode;
import oracle.apps.scm.configurator.runtime.core.ServiceException;
import oracle.apps.scm.configurator.runtime.core.SoapServiceRequest;
import oracle.apps.scm.configurator.runtime.core.SoapServiceRequest.SOAP_PROTOCOL_TYPE;
import oracle.apps.scm.configurator.runtime.core.SoapServiceResponse;
import javax.xml.soap.SOAPBody;
import javax.xml.soap.SOAPMessage;
import javax.xml.soap.SOAPElement;
import javax.xml.soap.SOAPBodyElement;
import javax.xml.namespace.QName;
import org.w3c.dom.Document;
import org.w3c.dom.Element;

public class WebCXmin implements Serializable {
 public WebCXmin() { }

 public String callWebService(IRuntimeNode node) {
 SoapServiceRequest request = new SoapServiceRequest(node.getConfiguration());
 SOAPMessage message = request.getSoapMessage();

 // Create the XML payload.
 QName bodyName = new QName("ws", "executeWebCX");
 SOAPBodyElement bodyElement = request.getSoapBody().addBodyElement(bodyName);
 bodyElement.addNamespaceDeclaration("ws", "http://services.vision/");
 bodyElement.setPrefix("ws");

 QName className = new QName("className");
 SOAPElement classes = bodyElement.addChildElement(className);
 classes.addTextNode("axel.ce.ws.webcx.DSPrimeTestCX");

 QName name = new QName("params");
 SOAPElement params = bodyElement.addChildElement(name);

 QName entry = new QName("entry");
 SOAPElement entries = params.addChildElement(entry);

 QName key = new QName("key");
 SOAPElement keys = entries.addChildElement(key);
 keys.addTextNode("json");

 QName value = new QName("value");
 SOAPElement values = entries.addChildElement(value);
```

```
values.addTextNode("{\"qty\": 2, \"children\": [], \"type\": \"MI\", \"state\": [\"UTRU\", \"SELD\",
\"USLD\"]}");

message.getMimeHeaders().addHeader("Content-Type", "text/xml; charset=utf-8");
message.getMimeHeaders().addHeader("SOAPAction", "executeWebCX");
SoapServiceResponse response;
try {
// Call the web service that you specified in the Setup and Maintenance work area.
response = node.getConfiguration().invokeSoapService("VisionWS", request);
} catch (ServiceException e1) {
throw new Exception(" msg=" + e1.getMessage());
}
def base = node;
def tf = base.getChildByName("TextFeature");
Document doc = response.getSoapBody().extractContentAsDocument();
Element root = doc.getDocumentElement();
tf.textValue = "SOAP Body node: "+root.getFirstChild().getTextContent();
}
}
```

7. Bind the event.

Attribute	Value
Event	Select one: - **postConfigInit**. Call the web service when the runtime configuration starts. - **postValueChange**. Call the web service when the user changes a value on the node that the bind references.
Class	ScriptClass
Method	getCpuHw(String region) Make sure the web service can interpret the value of the `region` argument.

Guidelines

- Your script must import the SoapServiceResponse class and the SoapServiceRequest class. You might need to import other classes depending on your needs.
- This example uses the `http://services.vision` namespace and the `cpuHw` method from the web service. It assumes that the web service defines them.
- You must use the invokeSoapService method from the configuration object to call the web service, and you must send it to the connector name that you register in the payload. In this example, the connector name is `VisionWS`.

Call SOAP 1.1 or SOAP 1.2

You must use the SOAP 1.1 specification to call some SOAP services. You can't use SOAP 1.2 with them because SOAP 1.2 doesn't support the Web Service Definition Language (WSDL). As a workaround, you can add HTTP headers when you call the web service in your extension rule:

```
Content-Type: text/xml
SOAPAction: executeWebCX
```

For example:

```
SOAPMessage message = request.getSoapMessage();
message.getMimeHeaders().addHeader("Content-Type", "text/xml; charset=utf-8");
message.getMimeHeaders().addHeader("SOAPAction", "executeWebCX");
```

If you don't specify these HTTP headers, then your SOAP 1.1 call will fail.

Related Topics

- Create an Extension Rule
- Guidelines on How to Be Groovy
- Guidelines for Writing Extension Rules

FAQ for Model Rules

What happens if I change the node name of a supplemental structure that participates in a rule?

Any rule that references the renamed node won't be valid when you compile the model.

- A warning message identifies the affected rules.
- The warning message identifies the original name of each node that you used when you created the rule.
- Oracle Configurator changes the value of the Status attribute from Valid to Error for each change that made the rule not valid.
- If you update a rule to use the new node name, then the rule becomes valid again.

Related Topics

- Overview of Using Constraint Definition Language

Why can I test, but not release a model that has rules that aren't valid?

You can test a model that has a rule that isn't valid because the testing phase of creating a model provides you the opportunity to make each rule valid. Oracle Configurator won't allow you to release a model that contains a rule that isn't valid because that rule will prevent your user from creating a configuration that's valid.

If you don't fix an invalid rule, then you can disable that rule, and then release the model.

Related Topics

- Overview of Using Constraint Definition Language

6 User Interfaces

Overview of Model User Interfaces

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 user-defined 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 modify 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 select 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 enables 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**

Related Topics

- User Interfaces for Configurator Models
- The Default User Interface
- Generated User Interfaces
- Multichannel User Interfaces
- How You Modify User Interfaces

User Interfaces for Configurator Models

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 modified to suit your requirements.
- 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 can't 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 and saved UIs, allowing modification. They aren't 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. You can't modify these templates.

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 (meaning that only one option at a time can be selected), it's mapped to a radio button group control template.
- If an option class isn't mutually exclusive, it's mapped to a check box group control template.
- A required child model 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.

Related Topics

- The Default User Interface
- Generated User Interfaces
- How You Modify User Interfaces
- Multichannel User Interfaces

The Default User Interface

If you don't define any of your own 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 can't 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's 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.

Related Topics

- User Interfaces for Configurator Models
- Generated User Interfaces
- How You Modify User Interfaces
- Multichannel User Interfaces

Generated User Interfaces

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.

Use 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 don't 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 can't 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 don't 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 doesn't 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 selected for a UI that's 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 select 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 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 don't 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 isn't 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's 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 modifying the generated UI.

Related Topics

- User Interfaces for Configurator Models
- How You Synchronize UIs with Structure
- The Default User Interface
- How You Modify User Interfaces
- Multichannel User Interfaces

How You Synchronize UIs with Structure

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 aren't 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 (meaning that they represent items that are end-dated as of run time), they're automatically filtered out of your UI without further action by you. They aren't 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 attributes 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 you modify the configuration's behavior for a node, then you might need to change the type of control template that renders the node. Assume your option class originally allowed the user to select only one value in an option, but now allows the user to select more than option. You might need to change the control template for the node's page from a radio button group to a check box group.
- If you resequence the nodes in a supplemental structure in the model, and if Configurator explicitly displays them in the UI, then you must resequence them in the existing UI. You don't need to worry about the option nodes of an option feature because resequencing doesn't affect them.

Related Topics

- User Interfaces for Configurator Models
- The Default User Interface
- Generated User Interfaces
- How You Modify User Interfaces
- Multichannel User Interfaces

Multichannel User Interfaces

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

Applicability helps you present the UI that's 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 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 select 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 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.

Related Topics

- User Interfaces for Configurator Models
- The Default User Interface
- Generated User Interfaces
- How You Modify User Interfaces

Templates, Pages, and Navigation Styles

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. You see a template map when creating a UI. The available navigation styles are:

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

Each template map is available in alternate variants:

- 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.

You can't 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.

> **Note:** Don't use the Single Page Navigation for Test UI with Enhanced Selection Controls template on UIs intended for end users. That template is designed for use only with the Test Model operation.

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:

- **Finish**: The configurator engine finishes the configuration, which means that the engine automatically completes the remaining selections that are required for a valid configuration of the model. After the configuration is finished, the configuration session ends, the configuration data is returned to the host application, and the user is returned to the previous location in the host application.
- **Finish and Review**: The configurator engine finishes the configuration, then navigates the user to the Review page, where the user's selections are displayed for review and possible further configuration.
- **Save for Later**: Saves the configuration in the exact state left by the user. The configurator engine doesn't finish the configuration, and the saved configuration may be invalid.
- **Cancel**: Warns the end user about losing any selections made, and returns to the host application without saving any configuration data.

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's 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's 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.

Running Summary

To assist a sales end user in maintaining a comprehensive picture of a complex configuration, you can add a running summary pane to each of the navigation style templates.

The running summary templates enable you to display cumulative sales information for all the items in the current configuration that can be selected and ordered.

Tip: The items listed in the running summary change as you make selections in the model, so including a running summary is a good way to check the behavior of your model when testing it. You can always remove the running summary later, if you want.

The following running summary templates are available:

- Running Summary with Item Description, and Amount
- Running Summary with Item Description, Quantity and Amount
- Running Summary with Item Name, Quantity and Amount
- Running Summary with Quantity and Item Description

The summary also includes the total configured net price, excluding charges such as tax and shipping.

To add a running summary to one of your own user interfaces:

1. On the User Interfaces tab of the Edit Configurator Model page, select one of your UIs.
2. On the Overview subtab, select a template from the **Running Summary Template** list.
3. When the UI is tested, the running summary appears on the right side of the page. The items in the summary are updated as you make selections in your configuration.

Instance Management Table

The configuration of multiple instances of a model can be complex and confusing to the end user. You can use the Instance Management Table template in model user interfaces to provide easier configuration of multiple instances at run time.

The Instance Management Table template is only applicable to model nodes that can have multiple instances. Multiple instances are specified prior to model import, in the Product Information Management work area. Such nodes display the **Instantiability** setting as **Multiple instances** on the Details pane of the model's Structure tab. When you create a user interface, any model nodes that can have multiple instances are rendered by default in the new UI using the Instance Management Table template. You can also create UI page items using the template by adding nodes from the model tree in the Structure pane.

The following variants of the template are available. You can change the variant by editing the page item for the template in the WYSIWYG pane.

- Default (Instance Management Table from UI Template Map)
- Instance Management Table
- Instance Management Table with Facets

At run time, a node rendered with the Instance Management Table template is initially displayed as a caption and an **Add** button. When the end user clicks Add, an instance of the node is created, and the instance management table is updated, adding a row that enables the end user to manage the newly created instance. The columns of the instance management table are described in the following table.

Table Heading	Instance Behavior
Name	The default name of the instance of the node, which can be edited by the end user. Each instance is identified by adding a sequence number. Example: `[1] [2] [3] ...`. Instance numbers remain associated with the instances for which they're created. Example: If the user creates instances `[1] [2] [3]` and deletes instance `[2]`, then the next new instance added is numbered `[4]`, although there are three instances at that moment.

Table Heading	Instance Behavior
Quantity	The number of instances of the item that will be managed together from that row of the table. Example: If the Quantity is 4, then 4 instances can be configured identically, deleted together, or duplicated together. The number of instances that can be entered in this field at that moment is displayed as tip text when you click in it. Note that the total quantity for all the instances in the table is governed by the Minimum Quantity and Maximum Quantity of the associated model node. Example: if the Maximum Quantity is 8, and there are four instances, and the user enters a quantity of 5 for one instance, then the other three instances are each assigned a quantity of 1.
Configure	The edit control to configure the instance. The configuration that you perform on this instance is applied identically to all of the number of instances in **Quantity**.
Delete	The edit control to delete the instance. Instance numbers of deleted instances aren't reused.
Duplicate	The edit control to duplicate the instance. The new duplicate instance keeps the instance number of its original, and prefixes the instance name with `Copy of`. Additional duplicates are prefixed with `Copy 2 of` and so on. Since the instance number of the original is retained in its duplicates, but each duplicate is counted as an instance, a new instance added with the **Add** button is assigned an instance number that reflects the current total number of instances. Example: If the user creates instance `[1]` and makes three duplicates of it, then the next new instance added is `[5]`, since there are four instances at that moment of addition. The control is disabled for all instances when the Maximum Quantity of the associated model node has been reached.

Item Selection Table

In item-based models, some option classes might contain hundreds or even thousands of options. Loading that many options in a UI control can affect the responsiveness and usability of the model. Configurator handles this situation by providing the Item Selection Table UI template.

- When a default user interface is needed, Configurator uses the Item Selection Table with Header template instead of a check box control template for option classes with more than 25 options.
- When you generate a new user interface, Configurator uses the Item Selection Table by default for option classes with more than 25 options. A single-select or multi-select version of the template is used, as appropriate to the option class. You can modify your generated UI to use other templates.
- The Item Selection Table template is available in versions with or without header, standard or enhanced, and single-select or multi-select.
- A query-by-example control is included in the template to enable the end user to search for options.

Predefined Templates

This table lists the UI templates that are predefined for use in your user interfaces.

Template Name	Description
Boolean Feature Control	Check box that indicates logic state of the associated node.
Boolean Feature Control with Facets	Check box that indicates logic state of the associated node with facets.

Template Name	Description
Check Box Group	Check box group for a set of options.
Check Box Group with Facets	Check box group for a set of options with facets.
Check Box Group with Quantity	Check box group with quantity fields for a set of options.
Check Box Group with Quantity and Facets	Check box group with quantity fields for a set of options and two facets.
Choice List	A drop-down list for selecting a single item. Applicable to option classes and option features.
Choice List with Facets	A drop-down list for selecting a single item with facets. Applicable to option classes and option features.
Compact Stack Layout	Renders the UI items of an associated UI region vertically with no spacers between them.
Enhanced Boolean Feature Control	Check box that indicates logic state of the associated node; indicates detailed selection state.
Enhanced Check Box Group	Check box group for a set of options; indicates detailed selection state.
Enhanced Check Box Group with Quantity	Check box group with quantity fields for a set of options; indicates detailed selection state.
Enhanced Item Selection Control	Check box for selecting standard items; indicates detailed selection state.
Enhanced Item Selection Control in Form Layout	Check box wrapped in a form layout for selecting standard items; indicates detailed selection state.
Enhanced Item Selection Control with Quantity	Check box with a quantity field for selecting standard items; indicates detailed selection state.
Enhanced Item Selection Control with Quantity in Form Layout	Check box with a quantity field wrapped in a form layout for selecting standard items; indicates detailed selection state.
Enhanced Item Selection Table with Header, Multi-Select	Table with header for selecting or configuring multiple items, showing quantity, description, and detailed selection state.
Enhanced Item Selection Table with Header, Single-Select	Table with header for selecting or configuring a single item, showing quantity, description, pricing, and detailed selection state.
Enhanced Item Selection Table, Multi-Select	Table for selecting or configuring multiple items, showing quantity, description, pricing, and detailed selection state.
Enhanced Item Selection Table, Single-Select	Table for selecting or configuring a single item, showing quantity, description, pricing, and detailed selection state.

Template Name	Description
Enhanced Item Single-Instance Control	Check box and a button to configure a single instance for an instantiable component or child model; indicates detailed selection state.
Enhanced Item Single-Instance Control in Form Layout	Check box and a button wrapped in a form layout to configure a single instance for a component or child model; indicates detailed selection state.
Enhanced Radio Button Group	Radio button group for a set of options; indicates detailed selection state.
Enhanced Radio Button Group with Quantity	Radio button group with quantity fields for a set of options; indicates detailed selection state.
Enhanced Summary Region	The Enhanced Summary Region UI element displays all options that are selected during a configuration session in a table.
Flow Layout	Renders the UI items of an associated UI region in a start to end flow layout.
Form Layout	Renders the items of the associated UI region in a form layout
Form Layout for Page	Renders the items of the associated page in a form layout
Form Layout with Quantity	Renders the items of the associated UI Region in a form layout and contains a numeric field bound to the Quantity attribute of the associated model node.
Item Instance Management Table	Table for adding, deleting, and configuring instances, and entering quantities for a child model.
Item Instance Management Table with Facets	Table for adding, deleting, and configuring instances, and entering quantities for a child model and facets.
Item Selection Control	Check box for selecting standard items.
Item Selection Control in Form Layout	Check box wrapped in a form layout for selecting standard items.
Item Selection Control with Quantity	Check box with a quantity field for selecting standard items.
Item Selection Control with Quantity in Form Layout	Check box with a quantity field wrapped in a form layout for selecting standard items.
Item Selection Table with Header, Multi-Select	Table with header for selecting or configuring multiple items, showing quantity, description, and pricing
Item Selection Table with Header, Single-Select	Table with header for selecting or configuring a single item, showing quantity, description, and pricing.
Item Selection Table, Multi-Select	Table for selecting or configuring multiple items, showing quantity, description, and pricing.

Template Name	Description
Item Selection Table, Single-Select	Table for selecting or configuring a single item, showing quantity, description, and pricing.
Item Single-Instance Control	Check box and a button to configure a single instance for a component or child model.
Item Single-Instance Control in Form Layout	Check box and a button wrapped in a form layout to configure a single instance for a component or child model.
Radio Button Group	Radio button group for a set of options.
Radio Button Group with Facets	Radio button group for a set of options with facets.
Radio Button Group with Quantity	Radio button group with quantity fields for a set of options.
Radio Button Group with Quantity and Facets	Radio button group with quantity fields for a set of options and two facets.
Running Summary with Item Description and Amount	The Running Summary Region UI element displays all options that are selected during a configuration session in a table showing Item Description and Amount.
Running Summary with Item Description, Quantity and Amount	The Running Summary Region UI element displays all options that are selected during a configuration session in a table showing Item Description, Quantity and Amount.
Running Summary with Item Name, Quantity and Amount	The Running Summary Region UI element displays all options that are selected during a configuration session in a table showing Item Name, Quantity and Amount.
Running Summary with Quantity and Item Description	The Running Summary Region UI element displays all options that are selected during a configuration session in a table showing Quantity and Item Description.
Section Header	Renders the items of the associated UI region in a stack inside a panel header
Section Header with Quantity	Renders the items of the associated UI region in a stack inside a panel header and contains a numeric field bound to the Quantity attribute of the associated model node.
Selectable Image Group	Renders the set of options as a group of selectable images
Selectable Image Group with Header	Renders the set of options as a group of selectable images with header
Spacer	Renders a spacer between UI elements on a page.
Stack Layout	Renders the items of the associated UI region in a stack layout, one below the other.
Stack Layout for Page	Renders the items of the associated page in a stack layout, one below the other.
Choice List for Transactional Attributes	A drop-down list for selecting a transactional item attribute.

Template Name	Description
Layout Area for Transactional Attributes	Form layout area for a transactional attribute.

Related Topics

- User Interfaces for Configurator Models
- How You Modify UI Elements
- The Default User Interface
- Generated User Interfaces
- How You Use Images for Selections

How You Modify User Interfaces

After you generate a UI for a model, you can modify 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 modify a UI, you can:

- Change the location of generated UI items
- Add model nodes to the UI
- Add basic 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 basic UI elements, and the WYSIWYG editor region where you can interactively see the results of your modifications 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 isn't 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's 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's 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 doesn't 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's 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 ease locating 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 select a form or stack layout for the region.

    You can't 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 autocomplete 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 isn't modifiable as part of a UI, since it's part of the structure and rules of the model, rather than its UI.

Adding Basic UI Elements

In addition to model-dependent elements that are automatically inserted into a generated UI, you can add basic UI elements that are independent of model structure to 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.
    - 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
```

- For an **IFrame** element, enter:

  - **URL:** URL of an accessible internal or external web site, with optional URL parameters that generate output displayed inside the frame. Example:

    ```
    https://www.mysite.com:1000/perform?taskID=12345678
    ```

    An IFrame element must be placed in its own layout region in the user interface.

    If the URL is outside the current domain, then you must use Cross-Origin Resource Sharing (CORS) to enable a client application running in one domain to retrieve resources from another domain, using HTTP requests. See the related topic about CORS for details.
  - **Width:** Width of the IFrame, in pixels. Do not add `px` to the number.
  - **Height:** Height of the IFrame, in pixels. Do not add `px` to the number.
  - **Inline Style:** CSS expression.

5. You can change the default associated model node for a basic UI element by clicking the Search and Select control on the **Associated Model Node** field, and selecting a node that's accessible relative to the UI page. Changing the associated model node enables you to affect basic UI elements based on values or events in other parts of the model. Examples include:

   - You can use the value entered for a text feature during a configuration session as part of the value for the URL parameter of an IFrame basic element. Set the text feature as the associated model node of the IFrame, then use the UI expression `#{amn.value}` to access the value of that node. Example:

     ```
     https://www.mysite.com:1000/perform?taskID=#{amn.value}
     ```

   - You can use selection of a value for an option class or option feature during a configuration session to conditionally control the whether an Image basic element appears or not. Set the option feature as the associated model node of the Image, then set the display condition for the Image to be TRUE when the option feature is bound to a value by the occurrence of a selection. Example:

     - **Object**: Associated model node
     - **Attribute**: IsBound
     - **Value**: Equals TRUE

Limitations on Modifying UIs

There are limitations to how you can modify 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 that directly represents a model node. 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 can't modify the configuration summary page.

Related Topics

- User Interfaces for Configurator Models
- Generated User Interfaces
- How You Modify UI Elements
- CORS

How You Modify UI Elements

After you generate a UI for a model, you can modify individual UI elements in it, to better suit the needs of your application.

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 selected 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. Select 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 won't affect UIs that you have 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're not displayed in the UI.

Modifying UI Element Captions

You can use the UI expression language to override the default item display name of a given UI item, replacing it with a combination of static text and all associated model node attributes, with the exception of DisplayName.

The following table provides an example of the effect of using the UI expression language.

Element of UI caption	Example
Associated model node in model structure	Name: CARWHEELS Description: Wheels
Default display name in UI	Wheels
UI expression	`Item Name: #{amn.name} - Item Description: #{amn.description}`
Modified display name at run time	Item Name: CARWHEELS Item Description: Wheels

To override the default display name of a UI element:

1. Select the UI element.

   You can modify the following UI elements:

   - Page Caption of a Page element
   - Header Region Caption of a Header Region element
   - Page Item Caption of a Page Item element
   - List Item Caption of a List Item element, which can be part of a Page Item

   > **Note:** When you override the default display name of UI caption elements, then the node property DisplayName isn't available for the associated model node.

2. Click the Edit control for UI element.
3. In the Edit dialog box, enter a UI expression for the caption of the UI element.

4. Click OK in the Edit dialog, save the UI, and click **Test Model**.
5. In the test UI, the caption appears with the overriding expression text.

Using UI Facets

You can add basic UI text elements in the UI facets of the header of certain UI page items, to augment the information presented to the end user at run time.

A UI facet is an area in the UI item into which you can place additional UI items related to the associated model node for the UI item, or reachable from the associated model node. The following configurator model node types have UI templates that include UI facets:

- Option classes
- Option features
- Boolean features

Templates with facets are available for the following UI page items:

- Boolean Feature with Facets
- Check Box group with Facets
- Check Box group with Quantity and Facets
- Choice List with Facets
- Item Instance Management Table with Facets
- Radio Button Group with Facets
- Radio Button Group with Quantity and Facets

A template with facets contains layout regions arranged in the following default relationship:

```
Header: template with facets
Messages facet: Compact Stack Layout template containing:
 Basic UI text elements (in vertical sequence)
Information facet: Flow Layout template containing:
 Basic UI text elements (in horizontal sequence)
Items or options of the associated model node
```

The layout regions can be changed if they're not suited for the desired behavior.

Example showing two text elements stacked in the messages facet and four text elements in a flow in the information facet:

```
System Memory
 Memory mixing requires firmware version: XXYY123
 Your selection of DIMMs exceeds the maximum of: 64
 Memory (GB) Included: 0 | Minimum Required: 64 | Maximum Allowed: 2048 | Total Configured: 2608
 [0] A8V234 Four 8 GB DIMM
 [0] A8V567 Four 16 GB DIMM
 [0] A8V890 Four 32 GB DIMM
 [18] A8V999 Four 64 GB DIMM
```

To use UI facets for a page item:

1. Select and edit a page item for an associated model node that can use facets.
2. Apply one of the templates that includes facets. After you click OK, the facets are added to the page item.
3. Select the first layout region under the header, which is the messages facet. This region's template is Compact Stack Layout. Don't change the template.

4. From the list in the UI Elements tab in the Resources pane, select the **Text** element, then, from the context menu or the UI Elements toolbar, click **Add to Page**.
5. Edit the basic Text element that you inserted into the facet. Add static text in the **Text** field to provide messages about configuring the associated model node to the end user.
6. You can use the UI expression language to dynamically display information about the associated model node, or nodes reachable from the associated model node. You must use expression language attributes that correspond to the node types for which you can use UI templates with facets.
   For example, assume that the associated model node is an option feature with Minimum Selections of 2 and Maximum Selections of 5, and you enter the following three expressions in Text field of three basic text elements in the messages facet:

   ```
   Number of options you selected: #{amn.selectedCount}
   Maximum number of selections allowed: #{amn.maxSelected}
   Minimum number of options selected?: #{ amn.minSelectionSatisfied}
   ```

   When you test the model and select one option, the following text is displayed in the facets:

   ```
   Number of options you selected: 1
   Maximum number of selections allowed: 5
   Minimum number of options selected?: false
   ```
7. Select the second layout region under the header, which is the information facet. This region's template is Flow Layout. Don't change the template.
8. Repeat the steps described for the messages facet, to add static or dynamic text in basic Text elements in the information facet.
9. Save your changes to the UI, and click **Test Model**.

CAUTION: If you add UI items to facets, then later change the page item's template to a template without facets, then the facets will be removed, and all the UI items that you added to the facets will be lost.

Related Topics

- User Interfaces for Configurator Models
- UI Expression Language
- Use Node Properties to Affect Runtime Behavior and Results

UI Expression Language

You can use the UI expression language to dynamically create values for text parameters of basic UI elements according to the associated model node attributes of any basic element UI item.

Syntax

Each expression includes a reserved keyword that references a model's node. You qualify it with a name that references an attribute on the model's node. For example:

```
The node named #{amn.name} has a maximum quantity of #{amn.maxQuantity}.
```

- You must start each expression with `#{` (pound sign and left curly bracket) and end it with `}` (right curly bracket).

- The `amn` keyword creates a reference to the model's node for a UI basic element.
- Each expression is case sensitive, including the `amn` keyword.
- You must always qualify the `amn` keyword with a reference to an attribute of the model's node:

  ```
  #{amn.referenceName}
  ```

- You must refer to valid attribute. Valid attributes are enumerated in the user assistance for configurator model node properties. If the attribute isn't valid, then Configurator will create an error when you attempt to use it in an expression.
- UI expression attribute references are formed by putting the initial letter of the attribute name into lower case. The prefixes `Is` and `Has` are omitted from expression references.

The following table summarizes attribute references in the UI expression language.

UI Expression Language Reference	Refers To	Example Expression
`name`	The name of the associated model node.	`#{amn.name}`
`description`	The description of the associated model node.	`#{amn.description}`
`quantity`	The quantity of the associated model node.	`#{amn.quantity}`
`boundQuantity`	Attributes named like IsBoundQuantity	`#{amn.boundQuantity}`
`hasChildren`	Attributes named like HasChildren	`#{amn.hasChildren}`
`value`	The value of a transactional attribute, if the associated model node is an attribute under the root model node.	`#{amn.value}`
`suppAttrs['<suppAttrName>']`	The value of the named supplemental attribute.	`#{amn.suppAttrs['color']}`
`userAttrs['<UDAttrGroup>.<UDAttrName>`	The value of the named user-defined attribute.	`#{amn.userAttrs['PhysicalAttributes.Color']}`

Restrictions on the UI Expression Language

The following restrictions apply to UI expression language expressions.

- Expressions are case-sensitive.
- Expression attribute references must be to attributes that are valid for the associated model node. This validation is also applied if you change the associated model node for an element.
- The expression language can be used with all model node system, user and supplemental attributes, but only for the text parameters of basic UI elements. The basic UI elements are: Text, Image, Spacer, and IFrame. The text parameters are: Text, Inline Style, Style Class and URL.
- References to elements of a collection or list aren't valid. Example of invalid reference: `#{amn.children[0].name}`.

- References to transactional item attributes are only valid when the associated model node of a basic UI element is a transactional attribute under the root node of the model. Valid reference expression: `#{amn.value}`.
- If an attribute value of the associated model node isn't available at run time, the displayed value is blank. If an expression can't be evaluate at run time because an attribute value isn't available, the displayed value is `N/A`.

Typical Uses for the UI Expression Language

You typically use the UI Expression Language to:

- Create a Text basic UI element in a user interface page to dynamically display the minimum and maximum values of its associated model node. Example:

  ```
  Min: #{amn.minValue}
  Max: #{amn.maxValue}
  ```

- Create an IFrame basic UI element where the IFrame's URL is dynamically augmented to provide query parameters. Example:

  ```
  https://www.mysite.com:1000/perform?taskID=#{amn.value}
  ```

- Store a central set of Cascading Style Sheets (CSS) in a node's supplemental attribute and use the expression language to generically refer to the CSS from a basic UI element's Inline Style attribute. Example:

  ```
  #{amn.suppAttr['InlineStyle']}
  ```

Related Topics

- User Interfaces for Configurator Models
- How You Modify UI Elements
- Use Node Properties to Affect Runtime Behavior and Results

How You Use Images for Selections

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 images to display for an item or option on a single row at run time.
   - **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.

Related Topics

- User Interfaces for Configurator Models
- The Default User Interface
- Generated User Interfaces
- Multichannel User Interfaces

How You Visualize Configurations

You can define user interface elements that enable end users to visualize how an object changes visibly in reaction to actions they take during a configuration session.

To provide visualization of objects in a configurator model:

- Design the visualization model
- Add an IFrame element and enable it to receive model node changes
- Communicate model node changes to the external visualization tool

Design the Visualization Model

Visualization requires that you provide a model in some external 2D or 3D visualization tool. This visualization model should correspond to all or part of the configurator model for the product that your end users are configuring.

Since many parts of a configurator model can change while users are configuring a product, it's important to determine the scope of the model node changes needed to support the visualization. If a visualization only supports a component of a model, then the scope of the model node changes should be reduced to that component only and not the entire model.

Add an IFrame Element and Enable It To Receive Model Node Changes

Visualization takes place within an inline frame of a user interface. You create the inline frame with an IFrame basic UI element. By default, an inline frame displays an accessible internal or external web site. To visualize configured objects, you enable the IFrame element to receive model node changes.

Consult the related topic about modifying user interfaces for more details on adding an IFrame basic UI element. For adding visualization, there are a few special things to do:

1. Select the location to add the IFrame element. It will be added below the currently selected page element.
2. From the list in the UI Elements tab in the Resources pane, select the **IFrame** element and click **Add to Page**.
3. In the **Add IFrame** dialog box, select the associated model node for the IFrame. Select carefully, because the scope of the node controls which model node changes are passed to the IFrame.
4. Select **Receives model node changes**.

5. Select the scope from which changes to the model should be communicated to the IFrame: **Associated Model Node and Subtree** or **Associated Model Node**.
6. In the **URL** field of the **Contents** group, enter the URL and parameters for the external visualization site that you're integrating with. For example:

   ```
   https://myserver.com/viewer.html?scs=model_data/mountainbike_blue.scs
   ```

   Remember that If the URL is outside the current domain, then you must use Cross-Origin Resource Sharing (CORS).
7. Enter other desired properties in the **Contents** group.

Tip: If a user interface has multiple UI pages, you can add an IFrame to each page, so that the end user can keep the visualized object in view throughout the configuration session. Depending on your use case, you could determine whether to display different components of your model on different UI pages, or the entire model on every page.

Model Node Changes That Are Passed to the IFrame

When end users select items or enter values for items in the IFrame's associated model node or subtree, Configurator passes the changes in the values of certain attributes to the IFrame, as JSON payloads of standard name-value pairs. When these model node change payloads are received by the IFrame, the values are passed to the external visualization tool identified by the IFrame's URL. Be aware that no information is passed back from the IFrame to the configurator model.

These static attributes are passed to the IFrame for every associated model node or subtree node:

- NodeType
- DisplayNamePath
- HasTransactionalAttributesFlag (only for node types imported from the Product Information Management work area)

It's useful to know which attributes are passed to the IFrame, and from there to the external visualization tool, because these attributes can be used in the JavaScript code that matches configurator model nodes with visualization model nodes, as shown in the code fragment provided here.

The NodeType determines which dynamic attributes are passed to the IFrame:

NodeType	SelectionState	Quantity	InstanceCount	Value
OPTION_CLASS_ITEM	Yes	Yes	No	No
STANDARD_ITEM	Yes	Yes	No	No
MODEL_ITEM	Yes	Yes	No	No
ROOT_MODEL_ITEM	Yes	Yes	No	No
COMPONENT_PORT	No	No	Yes	No

Communicate Model Node Changes to the External Visualization Tool

Configurator communicates model node changes in standard JSON payloads of name-value pairs to the IFrame, internally using the JavaScript method `window.postMessage()`. These payload messages must be interpreted by the visualization tool to properly display model node changes in the visualization IFrame.

For example, assume that a configurator model of a bicycle includes an option class and standard items with these display names:

```
Mechanical Options.Brakes
 Basic Alloy Bicycle Brake Set
 Dual Compound Side Pull Bicycle Brake Set
 Delux Cantilever Brake System
```

In the HTML file used by the visualization tool (which you specified in the **URL** field of the IFrame) you might write the following JavaScript statements to match the display names of the configurator model nodes with the corresponding nodes of a visualization model that you have built. The sample code below sets the color of a node of the visualization model when the matching configurator model node is selected by the end user in a configuration session.

```
...
if (displayNamePath.startsWith("Mechanical Options.Brakes")) {
 if (selectionState === "SELECTED") {
 if (displayNamePath.includes("Basic Alloy Bicycle Brake Set")) {
 var color = new Communicator.Color('255','255', '255');
 model.setNodesFaceColor([18, 19, 20, 21], color);
 } else if (displayNamePath.includes("Dual Compound Side Pull Bicycle Brake Set")) {
 var color = new Communicator.Color('100','100', '0');
 model.setNodesFaceColor([18, 19, 20, 21], color);
 } else if (displayNamePath.includes("Delux Cantilever Brake System")) {
 var color = new Communicator.Color('200','100', '100');
 model.setNodesFaceColor([18, 19, 20, 21], color);
 }
 } else if (displayNamePath === "Mechanical Options.Brakes" && selectionState === "SELECTABLE") {
 model.unsetNodesFaceColor([18, 19, 20, 21]);
 }
 }
...
```

Related Topics

- Add a Supplemental Structure
- How You Modify User Interfaces
- CORS

FAQ for 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 aren't tightly connected to the model. You can add, delete, and reorder pages without affecting the model structure. Similarly, you can change the page captions that are generated from node names in the model, because you aren't changing the model structure. However, UI page items are tightly connected to the model, and you can't 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.

Related Topics

- User Interfaces for Configurator Models
- How You Modify User Interfaces

7 Connectors

Overview of Connectors

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

- Connectors provide the basis for making connections from a model to related items outside that model. Use a connector when the configuration of a product requires you to include an item that isn't part of the product's structure.
- For example, when configuring the model for a computer, you might also want to add a warranty as part of the order, although warranty items aren't part of the computer model. A connector provides the means for including the warranty in the configuration.
- You define a connector in a model's structure, with a target that's an item class. Before defining the connector, you must import the items in that item class.
- The connections are created at runtime, as part of the configuration of a model. The items associated with the target item class are connectable to the model, through the connector.

When To Use Connectors

Products are often configured from items in a bill of materials that are related by specific rules to create a defined, limited model structure.

But if you need to configure products that depend on items that aren't part of a single model's structure, you might want to use connectors.

Here are examples of models for which you might use connectors to include separate but related items in a configuration of the model:

Configurator Model	Related Item Connected to Model	Conditions for Connection
Network server	Warranties	Exclude warranties for 12 months. Allow only warranties for 18 or 24 months.
Network server	On-site installation service	Automatically added as a related item on a quote or order.
Desktop computer	Cables	No restrictions.
Car	Trailer	Can optionally add only one trailer to a car.
Car	Warranties	Can add up to four warranties.

Here are some user stories where connectors could be the best solution:

User story 1

Create a quote or order for a configured item with separate related items. You're configuring a network server and have selected the required components of the configuration. Certain components that you have selected exclude warranties of 12 months, and only 18 or 24-month warranties are allowed. You should be able to select the available warranties from within the Configurator user interface. On completing the configuration, you should see the network server and the warranties that you selected as related quote or order lines. Also note that these warranties aren't modeled in the Product Development work area as part of the configured item's component structure.

User story 2

Modify a quote or order for a configured item with separate related items. You have an existing quote or order for which you're re-configuring a network server and have changed the selected components of the configuration. Certain components that you have selected now require an on-site installation service. The on-site installation service should automatically be added as a related item to the quote or order and you should be able to select additional available services from within the Configurator user interface. On completing the configuration, you should see the modified network server and the new on-site installation service as a separate but related line in the quote or order. Also note that these services aren't modeled in the Product Development work area as part of the configured item's component structure.

User story 3

Validate a quote or order on or before submission. You have an existing quote or order which includes a configured network server with an on-site installation service line item and you have removed the on-site installation service line item. Certain components that you had previously selected for the network server require an on-site installation service. Before you submit the quote or order, you select an action to verify it one last time. You will then be notified that an on-site installation service is required and it will be added back to the quote or order as a separate line item. (Note that this validation might also happen automatically when the quote or order is submitted.)

How You Create Connectors

You create connectors in a way that's mostly similar to creating other supplemental structure. But before you create a connector you must import the item class and item that a connector depends on.

Import Items Associated with an Item Class

Because a connector has an item class as a target, you must import the items from the intended target item class before you can create the connector to that target. If your item class includes an item class hierarchy for your related items, then you can import all the items in the hierarchy as well, to enable connections to them.

To import items associated with an item class:

1. Identify the item class associated with the items that you want to connect to your model.
2. On the Manage Snapshots page of the Configurator Models work area, select Import Items Associated with an Item Class from the Actions menu.
3. In the Import Item Snapshots dialog box, select the Organization that the items belong to and the Item Class of the related items that you want to connect to.

> **Note:** To be connectable, an item must have the attribute Order Management Indivisible set to Yes.

4. By default, only the items in the selected item class itself will be imported. If you also want to connect to all of the items associated with all of the child item classes of the selected item class, select Include items associated with child item classes. You can also select a particular child item class to import by itself.
5. Click Submit. A scheduled process is submitted. Make note of the scheduled process request ID.
6. In the Scheduled Processes work area, examine the log file for the *Import Items Associated with an Item Class* process. The log file contains the names of the items imported from the item class you chose. If they haven't been previously imported, then the process imports:
    - The specified item class itself
    - All standard items that are associated with the specified item class
    - If Include items associated with child item classes is selected, then all of the child item classes, and all of their associated items

    After the import, you can query the imported items and item classes on the Manage Snapshots page.

    > **Note:** Each import is limited to a batch of 5000 item snapshots. If that batch size is exceeded, the import process status is set to Warning. Continue to run the Import Items Associated with an Item Class action, with the same parameters, until all your items are imported. Check the list of imported items in the log file of the scheduled process to determine when you're finished. A list less with than 5000 items means there are no more items left to import.

7. You can refresh the item class snapshots in the usual way, to reflect item changes to existing items, or addition of new items associated to the item class, in the Product Information Management work area.

Create Connectors on Models

You add connectors to model structure, similar to the way that you add other supplemental structure. To enable the addition of related items to a mode at runtime, you add connectors to model structure.

1. If you haven't already done so, import all the items associated with the intended target item class.
2. In a workspace that you have locked, open a model for editing.
3. On the Edit Configurator Model page, select the root node of the model. Select Create > Connector. You can only create connectors on the root node.
4. In the Create Connector dialog box, you select a value for the Target of the specified Target Type. Search for and select the target item class. The search list contains all the item class snapshots that have been imported into the Configurator Models work area.

    After a connector is defined, all imported items associated with the target item class become items that are connectable to the model at runtime.

    > **CAUTION:** Targeting an item class or item class hierarchy containing a very large number of items can result in significant impacts to performance. If you target the Root Item Class, you can't select Include items associated with child item classes.

5. Enter values for Minimum Connections and Maximum Connections, to determine the number of instances of this connection that may be added at runtime. These fields represent the minimum and maximum number of actual related items which can be added to a configuration.
6. Enter values for Minimum Quantity and Maximum Quantity, to determine the sum quantity of connected items that may be added at runtime through this connector. These fields represent the minimum and maximum sum quantity of all the related items added at runtime to a configuration.

7. After you create a connector, you can modify the minimum and maximum connections and quantities, but you can't change the target.
8. There's currently no representation of connectors in a user interface. The UI Presentation setting is reserved for future use.
9. After you create the connector, it appears in the model tree, at root level. The connections that can now be made in a runtime configuration session are between related items and the model as a whole.

Related Topics

- Import Items Into Configurator Models
- Refresh Your Snapshot
- Snapshots

How You Use Connectors

You use the connectors that you created in your model structure to make connections, during a runtime configuration session, with items outside the structure of your model.

In this example, users who are ordering a car may want to also order a trailer to use with the car.

Here are the entities used in this example, and the roles they play in adding a trailer to an order for a car:

Entity	Role
Configurator model	The car being ordered. The example model is: CAR4DRSDN 4 Door Sedan
Item class	The class of trailers, created in the Product Information Management work area. The example item class is: Trailers Trailers
Standard items	The set of trailers, created as items of the Trailers item class, in the Product Information Management work area. The example set of standard items is: 2WHL_BSTL_TRLR_CRT 2 Wheel Black Steel Trailer Cart 4WHL_BSTL_TRLR_CRT 4 Wheel Black Steel Trailer Cart CO_TRLR Carry-On Trailer HDHM_CARGO_TRLR Heavy Duty Hitch Mount Cargo Carrier WM_UTIL_TRLR Wire Mesh Utility Trailer with Ramp Gate
Snapshots	The snapshots are the copies of the item class and its standard items, with the same names, imported into the Configurator Models work area.
Connector	The connector provides the basis for connecting the car model to the standard items in the Trailers item class. The example connector is:

Entity	Role
	Trailer Connector
Option feature and options	In the absence of UI templates for connected items, the option feature and its options represent the imported item class and its items. The example option feature has the same name as the item class: Trailers The options have the same names as the items, such as: HDHM_CARGO_TRLR
Text feature	The text feature holds the name of the option feature's option that the user selects at runtime, and represents the connected item. The example text feature is: Selected Trailer Item
Statement rule	Sets the value of the text feature to be the name of the option feature option (representing the connected item) selected at runtime. The example statement rule is: Updated Selected Trailer Item
Extension rule	Adds a connectable item to the configuration, using the name of the selected option feature option to connect to the corresponding standard item. The example extension rule is: Add Trailer

The steps to put these elements of the example together are:

1. Import the snapshots.
2. Create the supplemental structure.
3. Create the statement rule.
4. Create the connector.
5. Create the extension rule.
6. Test the model.

Import the Snapshots

On the Manage Snapshots page of the Configurator Models work area, select Import Items Associated with an Item Class from the Actions menu.

The data shown in this example is hypothetical, for demonstration purposes.

- Search for and select the item class: Trailers.
- Don't select the Include items associated with child item classes option so that the import contains only the items associated with the Trailers item class, and not from its child item classes.

The resulting imported standard items include:

- HDHM_CARGO_TRLR Heavy Duty Hitch Mount Cargo Carrier
- WM_UTIL_TRLR Wire Mesh Utility Trailer with Ramp Gate
- And so on

Create the Supplemental Structure

Since there's currently no predefined user interface support for connectors, you can represent the Trailer item class as an option feature, and the associated items as its options. You can modify the default user interface for the options as desired, for instance by using selectable images for the options. At runtime, the end user can select these options in the user interface and an extension rule adds the actual related items to the running summary.

In the CAR4DRSDN model, create an option feature and its options, with these settings:

- Name: Trailers. For clarity, this is the same name as the item class.
- Minimum Selections: 0
- Maximum Selections: 1
- Create an option that matches the name of each of the standard items in the Trailers item class, such as HDHM_CARGO_TRLR. You can use ADFdi integration to create the options, if there are a large number of them.

In the CAR4DRSDN model, create a text feature with these settings:

- Name: Selected Trailer Item
- Maximum Length: 100 (or whatever is long enough to accommodate the names of your standard items).

Create the Statement Rule

In the CAR4DRSDN model, create a statement rule, with these settings:

- Name: Updated Selected Trailer Item
- Rule Class: Constraint
- In the Rule Text, create a series of REQUIRES statements, one for each option of the Trailers option feature.
    - On the left-hand side of the rule, insert an option of the Trailers option feature.
    - On the right-hand side of the rule, insert the text feature, Selected Trailer Item, and assign the name of the Trailers option as its value.

The CDL statements in the Rule Text field should look like this:

```
Copy'CAR4DRSDN'.'Trailers'.'HDHM_CARGO_TRLR' REQUIRES 'CAR4DRSDN'.'Selected Trailer Item' =
 "HDHM_CARGO_TRLR";
'CAR4DRSDN'.'Trailers'.'CO_TRLR' REQUIRES 'CAR4DRSDN'.'Selected Trailer Item' = "CO_TRLR";
'CAR4DRSDN'.'Trailers'.'WM_UTIL_TRLR' REQUIRES 'CAR4DRSDN'.'Selected Trailer Item' = "WM_UTIL_TRLR";
'CAR4DRSDN'.'Trailers'.'2WHL_BSTL_TRLR_CRT' REQUIRES 'CAR4DRSDN'.'Selected Trailer Item' =
 "2WHL_BSTL_TRLR_CRT";
'CAR4DRSDN'.'Trailers'.'4WHL_BSTL_TRLR_CRT' REQUIRES 'CAR4DRSDN'.'Selected Trailer Item' =
 "4WHL_BSTL_TRLR_CRT";
```

When the end user at runtime selects an option of the Trailers option feature (such as 'HDHM_CARGO_TRLR'), the rule sets the value of the text feature Selected Trailer Item to the name of the trailer standard item (which would be "HDHM_CARGO_TRLR"). This name is used to create a connector instance that specifies that standard item.

The rule code shown in the example is designed primarily for clarity.

Create the Connector

In the CAR4DRSDN model, create a connector under the root node, with these settings:

Field	Value
Name	Trailer Connector
Target Type	Item Class
Target	Trailers
Minimum Connections	0
Maximum Connections	1
Minimum Quantities	1
Maximum Quantities	1

The characteristics defined in the connector node govern the behavior of the items related through the connector at runtime:

- The target type governs which set of related items are connectable to the model. For a connector whose target type is item class, the set of related items is the set of items associated with that item class.
- The minimum and maximum connections govern how many different standard items from the target item class can be added to this configuration. In this example, that means how many different trailers can be added to the configuration of the car. Because you only want to allow one trailer per car, you set Maximum Connections to 1. To make the addition of a trailer to the configuration optional, you set Minimum Connections to zero.
- The minimum and maximum quantities govern the sum of all the quantities of standard items in the target item class.
- In this example, that means how many total trailers can be added to the configuration of the car. Again, because you only want to allow one trailer per car, you set both Minimum Quantities and Maximum Quantities to 1

Create the Extension Rule

In the CAR4DRSDN model, create an extension rule, with these settings:

Field	Value
Name	Add Trailer
Base Node	The text feature Selected Trailer Item. The statement rule sets the value of this feature to the name of the standard item that's selected at runtime.
Instantiation Scope	Instance

Field	Value
Rule Text	Enter the Groovy code sample that follows this table. You can reuse this code for other connectors, with the substitution of your connector name for the name in this example.
Event	postValueChange
Event Scope	Base node
Class	ScriptClass
Method	run()

Enter this Groovy code in the Rule Text field:

```
Copyconfig = cxEvent.configuration
root = config.getRootBomModel()
baseNode = cxEvent.baseNode;
// Retrieve the connector definition from the root model
connPort = root.getChildByName("Trailer Connector");
// Check to see if the connector has any connector instances
if (connPort.getInstanceCount() == 0) {
 // If not then add an instance
 instance = connPort.addInstance();
}
// Retrieve the connector instance
instance = connPort.getInstances().get(0)
// Set its quantity to 1
instance.quantity = 1;
// If the connector instance has an existing connected item defined, then remove it
instance.unsetType();
// Set the connector instance to the new connected item
instance.setType(instance.getAvailableLeafType(baseNode.getTextValue()));
```

The following statement locates the connector you created, by its name:

```
CopyconnPort = root.getChildByName("Trailer Connector");
```

When adapting this example to your own scenario, replace the name of the example connector node (Trailer Connector) with your own connector node name.

The following statement sets the type of the new connector instance, by using the name of the option corresponding to an imported standard item snapshot (in other words, to the standard item):

```
Copyinstance.setType(instance.getAvailableLeafType(baseNode.getTextValue()));
```

Connectors in Action

Connectors provide behavior that's distinct from other configurator model entities.

Runtime Configuration Sessions

Connectors affect how you interact with models during a configuration session.

- The extension rules that you define can drive the runtime selection of related items that belong to an item class. You can implement business logic in your rules to select the items. Otherwise, you would have to add the related items to the item structure underlying your models, which would be onerous to build and maintain, considering that related items are commonly related to multiple models.
- During a configuration session, any related items that you add to the configuration through connectors and rules are added as lines in the running summary pane of the user interface. The name and quantity of each related item is displayed. User interface templates for connected items aren't available, so you can't add connected items to your UI modifications.
- When you select to save a configuration during a session, and later restore the configuration in a new session, connected items are included. Host applications can also save and restore configurations including connected items.
- Be aware that items with decimal or fractional quantities can't be used with connectors. This is governed by the attribute Order Management Indivisible in the Product Information Management work area.

Connections and Quantities

It's helpful to understand some of the design background when you're using connectors.

- Connector nodes are definitions for what are, in effect, typed runtime instances of the connector entity. The name of the connector node becomes the name of the connector instance type. The characteristics defined in the connector node govern the behavior of that connector type's instances at runtime:
- The item class that's named as the target of the connector provides scope for validating the item being connected to. The item class name is used to provide the model with the items it can connect to at runtime, and only those items. The available items from that item class can be limited and become unavailable for selection, depending on your rules governing the availability of these items.
- The minimum and maximum connections govern how many instances of the connector type can be instantiated during the configuration of the model. In functional terms, those values govern the minimum and maximum number of different standard items from the target item class can be added to a configuration. In the example used elsewhere in this chapter, that means how many different trailers can be added to the configuration of the car.
- The minimum and maximum quantities govern the sum of all the quantities of related items. In the example of the car and trailer, that means how many total trailers can be added to the configuration of the car.

Here's an example of the relationship between connections and quantities. Assume that your model represents a family calling plan for cell phone service. A connector targets the item class for cell phone models, in which items are different phones. You would define the minimum and maximum connections and quantities to govern how the phones can be added as related items to the configuration of the plan model.

The plan allows, per family:	Your connector definition specifies:
Up to 3 different phone models, from among: xPhone Solar	Minimum Connections = 1 Maximum Connections = 3

The plan allows, per family:	Your connector definition specifies:
Pixie	
Up to 4 total phones	Minimum Quantity = 1 Maximum Quantity = 4

These different combinations would all be allowed by your connector definition:

xPhone	Solar	Pixie	Total Quantity
2	1	1	4 (Maximum Quantity)
4	0	0	4 (Maximum Quantity)
2	0	1	3
1	0	0	1 (Minimum Quantity)

How You Filter Connectable Items

The target item class for a connector might contain thousands of items, which could strongly affect performance when loading the connector's set of connectable items at runtime. You can define a filtered set of connectable items for a connector to reduce that performance effect.

Define the Set of Connectable Items

To define a set of connectable items for a connector:

1. On the Structure tab of the Edit Configurator Model page, select a connector.
2. In the Details region of the connector, select the Connectable Items tab.
    - The items in this table comprise a subset of the full set of items associated to the connector's target item class. At runtime, these items will be the only ones connectable to your model.
    - If you don't add any items to this table, then at runtime the full set of items associated with the connector's target item class will be considered connectable items.
3. Click Select and Add. In the Select and Add: Connectable Items dialog box, search for items associated to the connector's target item class. You can search on the Name, Description, or Item Status of the items.
4. Select the items to add, then click Apply to continue adding as many items as needed without closing the dialog, or click OK to close the dialog and add the selected items.
5. Save the model.

Select the Connectable Items at Runtime

You can use an extension rule to select the connectable items that you defined. Here's a simple example of one way that you can automatically select connectable items.

1. Define a connector with these values:
    - The Name is MyConnector.
    - The target item class of your choice.
    - The Minimum Connections and Minimum Quantities are 1 or more.
    - The Maximum Connections and Maximum Quantities are equal to or greater than the number of items selected on the Connectable Items tab of your connector.
2. Define an extension rule with these settings:
    - The Base Node is set to the root node of the model.
    - This event binding:

Field	Value
Event	postConfigInit
Event Scope	Global
Class	ScriptClass
Method	run()

    - The following rule text, substituting the name of the first item in your set of connectable items for FIRST_ITEM_NAME. The rule loops through the set of connectable items, beginning with the first one:

```
Copyimport oracle.apps.scm.configurator.runtime.core.IConnectorNodeType;

def baseNode = cxEvent.baseNode;
def connPort = baseNode.getChildByName("MyConnector"); // The name of your connector.
def instance = connPort.getInstances().get(0);
def inst;

for (Object type : instance.definitionLeafTypes)
{
// Add all the items in the set of connectable items, starting with the first one.
 if(type.id == "FIRST_ITEM_NAME") { // The name of the first item in your connectable set.
 instance.setType((IConnectorNodeType)type);
 instance.quantity = 1;
 } else {
 inst = connPort.addInstance();
 inst.setType((IConnectorNodeType)type);
 inst.quantity = 1;
 }
}
```

3. Optionally, create a user interface that includes one of the running summary templates.

4. Save and Compile the model.
5. Click Test Model. If you added the running summary, select the user interface you created that includes it.
6. When the Test Model user interface appears, the items in your set of connectable items are automatically selected by the extension rule when the new configuration is initialized. The items in your set of connectable items appear in:
    - The running summary, if you included that template in a UI that you created.
    - The Review page, if you click Finish and Review. All connected items appear there by default, regardless of whether you added a running summary.

How You Integrate Connectors

You can return configuration selections and items related by connectors to hosting applications.

REST Resources

You can use the connectedItems REST resource to return the connected items from a specified configuration. The connectedItems resource is a child of the `configurations` resource.

- A hosting application can call Configurator REST services to retrieve connected items as independent lines.
- Connected items can be retrieved through the `configurations` resource for integration purposes in other applications, such as a quoting or Order Management.
- You can retrieve an individual connected item's runtime attributes, such as item number, unit of measure or quantity ordered.

Example

The following sample URL uses the `connectedItems` child resource of the `configurations` resource to obtain an output payload containing all of the connected items included in the specified configuration. In this example, there are two connected items in the configuration.

`Copyhttps://vision.com/fscmRestApi/resources/11.13.18.05/configurations/300100185429818_300100185429819/child/connectedItems`These are the objects of our main interest in the output payload:

Example value	Object name	Description
300100185429818	ConfigHeaderId	Configuration header ID.
300100185429819	ConfigRevisionId	Configuration revision ID.
300100185429818_300100185429819	ConfigurationId	ID of the configuration, composed of the configuration header ID and the configuration revision ID. You pass this to the resource to return the connected items.
First example connected item	-	-

Example value	Object name	Description
300100185429862	ConnectedItemId	Connected item ID of the first connected item.
300100181203985	InventoryItemId	ID of the first connected item.
7YR_PTRAIN_WARR	InventoryItemNumber	Item name of the first connected item.
Second example connected item	-	-
300100185429864	ConnectedItemId	Connected item ID of the second connected item.
300100181204014	InventoryItemId	ID of the second connected item.
WM_UTIL_TRLR	InventoryItemNumber	Item name of the second connected item.

Here is the JSON output payload returned for the connected items:

```
Copy{
 "items": [
 {
 "ConfigHeaderId": 300100185429818,
 "ConfigRevisionId": 300100185429819,
 "ConnectedItemId": 300100185429862,
 "ParentConfigLineId": 300100185429821,
 "InventoryItemId": 300100181203985,
 "InventoryItemNumber": "7YR_PTRAIN_WARR",
 "InventoryOrganizationId": 204,
 "InventoryOrganizationCode": "V1",
 "UomCode": "Ea",
 "InventoryItemType": 4,
 "Quantity": 1,
 "links": [
 {
 "rel": "self",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819/child/connectedItems/300100185429862",
 "name": "connectedItems",
 "kind": "item"
 },
 {
 "rel": "canonical",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819/child/connectedItems/300100185429862",
 "name": "connectedItems",
 "kind": "item"
 },
 {
 "rel": "parent",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819",
 "name": "configurations",
 "kind": "item"
 }
 ]
 },
```

```
 {
 "ConfigHeaderId": 300100185429818,
 "ConfigRevisionId": 300100185429819,
 "ConnectedItemId": 300100185429864,
 "ParentConfigLineId": 300100185429821,
 "InventoryItemId": 300100181204014,
 "InventoryItemNumber": "WM_UTIL_TRLR",
 "InventoryOrganizationId": 204,
 "InventoryOrganizationCode": "V1",
 "UomCode": "Ea",
 "InventoryItemType": 4,
 "Quantity": 1,
 "links": [
 {
 "rel": "self",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819/child/connectedItems/300100185429864",
 "name": "connectedItems",
 "kind": "item"
 },
 {
 "rel": "canonical",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819/child/connectedItems/300100185429864",
 "name": "connectedItems",
 "kind": "item"
 },
 {
 "rel": "parent",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819",
 "name": "configurations",
 "kind": "item"
 }
 ]
 }
 ],
 "count": 2,
 "hasMore": false,
 "limit": 25,
 "offset": 0,
 "links": [
 {
 "rel": "self",
 "href": "https://vision.com:443/fscmRestApi/resources/11.13.18.05/
configurations/300100185429818_300100185429819/child/connectedItems",
 "name": "connectedItems",
 "kind": "collection"
 }
 ]
}
```


Related Topics

What's the difference between a connector and a model reference

Model references and connectors both extend the structure of configurator models, but are different entities.

A model reference is a hierarchical link to a child model.

- A model reference is part of the bill of material for the configured product, and appears as the same order line in Order Management. On the other hand, connected related items are outside the bill of material, and can appear as separate lines in a quote or order. Connected related items can be individually fulfilled, and shipped apart from the configured product.
- A model reference is defined in the Configurator Models work area by importing a snapshot of a hierarchical model item structure that was created in the Product Information Management work area and contains child models, which become referenced models to the parent model.
- During a runtime configuration session, you can navigate the hierarchical model item structure to configure each distinct referenced model. The child models appear on the configuration summary page, if configured.

A connector is a non-hierarchical relationship to an item that exists outside of the model item structure.

- A connector is defined in the Configurator Models work area, as an element of configurator model supplemental structure. The relationship to the related item is established through the item's item class.
- During a runtime configuration session, you can select the related item to be included in the configuration of the bill of materials, although the item isn't part of the model. The related items appear in the running summary pane, but not on the configuration summary page.
- References are to models, and connectors are to items, not models.

8 Test

Overview of Testing Your Model

This chapter covers testing of configurator models.

You can test configurator models using the methods listed in the following table.

Method	Description
Interactively	Whenever viewing or editing a model in the Configurator Models work area, you can launch a simulated configuration session that uses the structure, rules, and user interfaces that you have defined for the model.
Test Service	Using the Configurator Runtime Model Test service, a SOAP web service, you can run Configurator in a non-interactive mode. The service takes as input a payload that creates the configuration, performs one or more configuration operations on the components of the model and closes the configuration. All of these operations are performed without any end user interaction.

Related Topics

- Test Models Interactively
- Guidelines for Testing Your Model
- Test the Model

Test Models Interactively

You can test a configurator model interactively, at any time during its development cycle, to ensure that its structure, behavior, and appearance are as you intend.

You can test the following aspects of a configurator model, as shown on the Configurator Model details page for viewing or editing a model:

- The structure of the model, as shown on the Structure tab, with any supplemental structure that you have added.
- The configurator rules that you have defined for the model, as shown on the Rules tab.
- Any user interfaces that you have defined for the model, as shown on the User Interfaces tab.

The aspects are all active together during a test session, subject to any test parameters that you apply, or any restrictions that are part of the model, such as:

- Items that are ineffective based on the test session date

- Rules that are disabled or invalid

Test Parameters

In the Test Model dialog box, you can set the test parameters described in the following table.

Test Parameter	Meaning
Root Model	The model being viewed or edited. Read-only.
Version	Read-only.
Session Effective Date	The date and time to use in the test session for considering effectivity. The session effective date must be on or after the Effective Start Date of the workspace that the model is a participant of. Required.
User Interface	If user interfaces have been defined for the model, the UI to use for the test session, or the default UI. Optional.
Root Quantity	The quantity of the root model to use for the test session, which affects the relative quantity of child items in the configuration. Required.
Enable pricing	Enables the calculation and display of prices during a test session. If you enable pricing, then you must select a pricing strategy that includes pricing rules for this model, in the **Pricing Strategy** field. Names of pricing strategies are suggested in the field as you enter part of their name.
Generate trace file	Enables the generation of a trace file. If you enable trace file generation, then you will be prompted to save the file at the end of the test session.

Test a Model

Follow these steps to interactively test a model. The test parameters are described separately.

1. Open a configurator model for editing or viewing.

   You must open a specific model. The model can be either locked or unlocked. If the model is not locked by the user who is testing it, that user must first unlock it. The model status can be either Draft or Released. You can be using either the Structure, Rules, or User Interfaces tab; the same test session tests all those aspects of the model. If you're editing the model, you will be prompted to save any model changes.
2. On the Configurator Model details page, click **Test Model**.
3. In the Test Model dialog box, enter values for the test parameters. When you click **OK**, the configurator runtime test UI opens in a new dynamic tab of the work area window.
4. Interact with the configurator runtime test UI, as if you were an end user. The contents and appearance of the test UI reflect the test parameters that you entered for this test session.

   During the test session, you can navigate UI pages, select options and enter field values, and provide values for transactional attributes. You can configure referenced models by clicking the **Configure** control, configuring the model, then returning to the root model.

5. When you're finished with your testing, click one of the action buttons to complete the test session. The available actions are similar to, but distinct from, the actions available to the end user in the configurator runtime end user UI.
    - **Finish**: The configurator engine finishes the configuration, which means that the configurator engine uses the autocomplete process to automatically complete the remaining selections that are required for a valid configuration of the model. A valid configuration is determined by the attributes defined for the item structure in the Product Information Management work area, and by the configurator rules defined in the Configurator Models work area. After the configuration is finished, the test session ends, and you're returned to the Configurator Model details page.
    - **Finish and Review**: The configurator engine finishes the configuration, then navigates to the Review page, where your selections are displayed for review and possible further configuration by clicking **Back**. Clicking **OK** returns to the Configurator Model details page.
    - **Save for Later**: Saves the configuration in the exact state that you left it. The configurator engine doesn't finish the configuration, and the saved configuration may be invalid.
      You can assign a name to the configuration as you save it, and later restore the configuration for further testing.
    - **Cancel**: Warns you about losing any selections made, and exits the test session, returning to the Configurator Model details page.

    In all of these session actions, no configuration data is returned to any host application, since there is no host application for a test session.
6. If you enabled the generation of a trace file, then, when you're exiting the test session, a message confirms that the trace file was generated and is available for download. You can open the file or download it for later examination. When you close the confirmation dialog box, the test session ends.
7. Return to viewing or editing the tested model, on the Configurator Model details page.

Trace Files

You can select the **Generate trace file** option in the Test Model dialog box to produce a trace file that can be provided to Oracle Support.

Issues that surface while testing models that have large sets of complex rules can sometimes be difficult and time-consuming to diagnose. If you generate a trace file for a runtime test session, the file helps Oracle Support, working jointly with you and Oracle product development, to diagnose the issues and resolve the problem, using constraint technology analysis and other tools and techniques.

The trace file is named after the model being tested, and is in XML format.

Review Test Configurations

You can review the interim configuration results of a test session before leaving the session.

If you select the **Finish and Review** action during a test session, then you're navigated to the Review page after the configurator engine finishes the configuration. On that page, your selections are displayed for review.

- The model items that have been configured during the session are displayed in a hierarchical tree that can be expanded and collapsed. Values entered for transactional item attributes are displayed in the tree, but values for user-defined attributes aren't displayed.
  Supplemental structure nodes aren't included on the Review page because you can't order them, and would not be returned to a host application after a session.
- For each configured item, the page displays the following values:

- **Quantity**: The quantity of the item included in the configuration, either by your selection or because of the action of configurator rules.
- **Unit Quantity**: The quantity of the item included in the configuration for each included unit of the item's parent item.
- **UOM**: The unit of measure for the quantity of the configured item.
- **Your Price**: The price of the item as determined by the applied pricing strategy. Pricing must be enabled for the test session.
- **Amount**: The amount of Your Price multiplied by the amount of Quantity. Pricing must be enabled for the test session.

When you're finished reviewing, click one of the following action buttons:

- **Back**: You're returned to the runtime test UI for possible further configuration.
- **OK**: The test session ends, and you're returned to the Configurator Model details page.
- **Cancel**: Warns you about losing any selections made, and exits the test session, returning to the Configurator Model details page.

Restore Saved Configurations During Model Testing

You can test the latest configuration for an item against previously saved configurations, to understand what changed. This ability to review changes in configurations helps you ensure the accuracy over time of your item structure and your configurator model definition.

When testing models, you can:

- Save configurations during model testing

  When you click **Save for Later** during a configuration session, you can enter a name and optional description for the configuration. You can also record the system-generated configuration header and revision IDs, which you can later use to search for this configuration

  You can establish a naming strategy for saving configurations, to ease searching among them.
- Test model changes with previously saved configurations

  When you start a test session, you can select **Restore a configuration** in the Test Model dialog box, then select the configuration to restore from the **Configuration Name** list. You can search for a configuration by its header ID or revision ID by clicking the **Search** link in the list.

  If you modified the model after the previous test session, you can compare how the modified model behaves against one of your saved configurations.

Get Rule Explanations When You Test Your Model

Get details about how your rule affects runtime behavior when you test it in the Configurator Modeling work area.

This feature diagnoses the factors that affect your model's runtime behavior. If it finds a conflict, then Configurator displays a dialog that describes your actions and the rules or constraints that caused the conflict. The dialog also

describes why your rule selected or excluded an option in the configuration. Use these details to help you efficiently test and deploy your configurator rules.

- Help troubleshoot unexpected runtime behavior for your configurator rule.
- Quickly identify the rule that's selecting or excluding components in your model.
- Help troubleshoot a large number of statement rules.
- Analyze the runtime behavior of a complex rule in your model.
- Reduce the time it takes to debug your rule.
- Examine your model's constraints, such as the minimum and maximum value for each constraint.

This feature also:

- Checks to see whether your rule selects the parent node.
- Checks to see whether your rule allows you to select more than the maximum quantity for a model during a runtime configuration session.

You can use rule explanations only with the templates that you use for option classes and features. You can't use it with other objects.

Try It

1. Make sure you have the Test Configurator Model (CZ_TEST_MODEL_PRIV) privilege.
2. Go to the Configurator Models work area.
3. Create a template map with these values.

Attribute	Value
Name	Enter any text. For this example, enter `My Test Template`.
UI Template Map	Select **Single Page Navigation for Test UI with Enhanced Selection Controls**. You must use this template. Don't use any other template.

   For details, see *Generated User Interfaces*.
4. Test your model.
   - Click **Tasks > Manage Models**.
   - On the Manage Models page, search for and open your model.
   - Click **Test Model**, then set these values in the Test Model dialog.

Attribute	Value
User Interface	My Test Template
Enable Rule Explanations	Contains a check mark.

   For details about this dialog, see *Option Features*.

5. Click **OK**.
6. Configurator will test your model. It will display visual cues for your items, such as an exclusion or automatic selection. You enabled rule explanations, so you will see an `Explain` link next to each item that has a runtime change, such as a user selection, system selection or exclusion, or a quantity change.
7. Click any `Explain` link to get a rule explanation.

Example

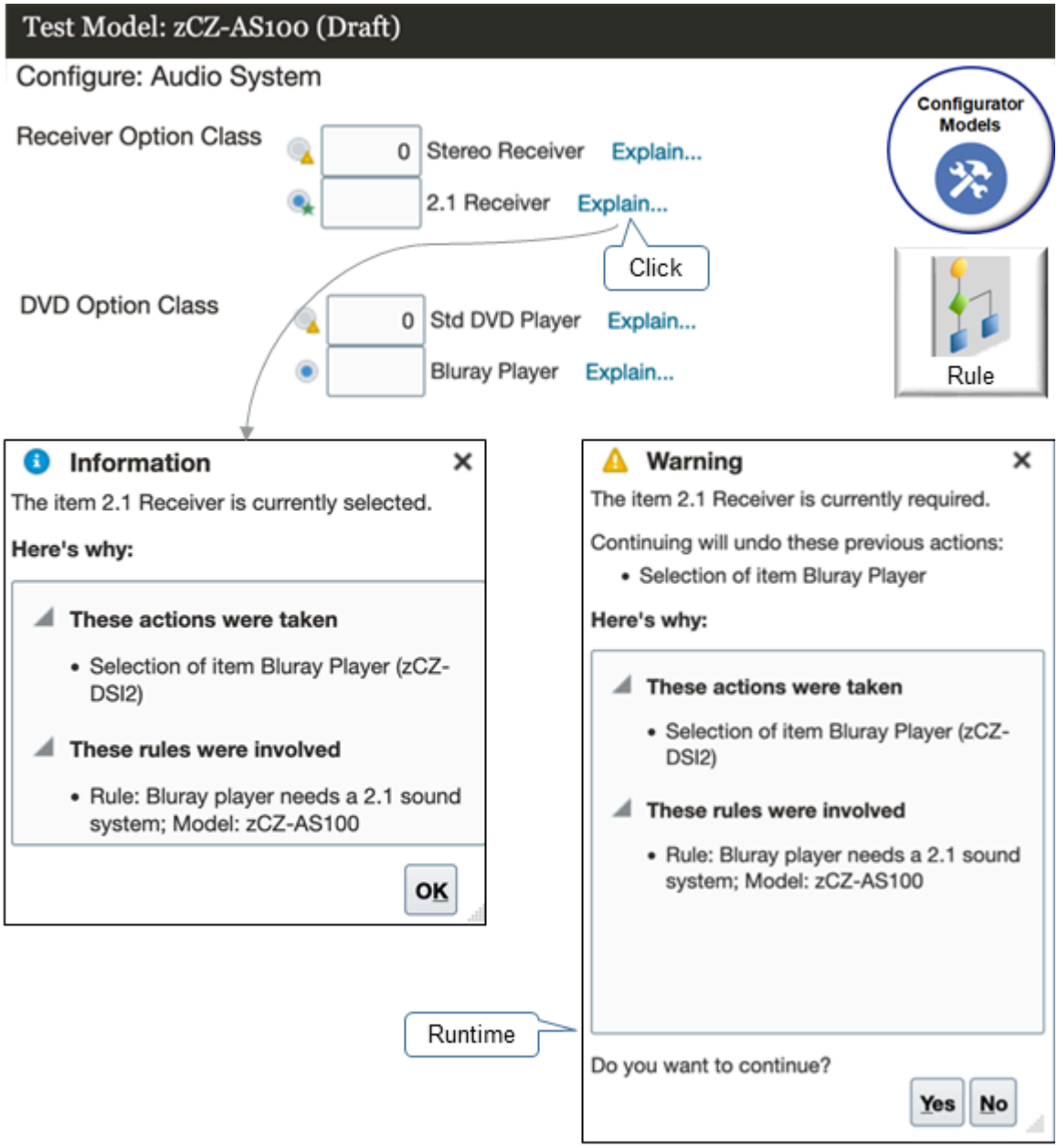
Assume you set up this model.

```
Audio System
 Receiver Option Class
 Stereo Receiver
 2.1 Receiver
 DVD Option Class
 Std DVD Player
 Bluray Player
```

You create a rule:

```
Use the Bluray Player only with a 2.1 Receiver
```

You then test your model.



During the test, select **2.1 Receiver** and select **Bluray Player**, click **Explain** next to 2.1 Receiver, then examine the details in the Information dialog.

```
Information
The item 2.1 Receiver is currently selected.
Here's why:
These actions were taken:
Selection of item Blueray Player (zCZ-DS12)
These rules were involved:
Rule: Bluray player needs a 2.1 sounds system; Model: zCZ-AS100
```

The Information dialog identifies the rule that configurator applied and why it applied that rule.

At run time, select **Stereo Receiver** and select **Bluray Player**, then examine the details in the Warning dialog that configurator displays.

```
Warning
The item 2.1 Receiver is currently required.
Continuing will undo these previous actions:
Selection of item Blueray Player
Here's why:
These rules were involved:
Rule: Bluray player needs a 2.1 sounds system; Model: zCZ-AS100
```

The Warning dialog tells you what item is required, what will happen if you continue, identifies the rule that configurator applied, and why it applied that rule.

Guidelines for Testing Your Model

To efficiently test a large number of models, you can use the Configurator Runtime Model Test service to execute model tests without requiring user input.

The Configurator Runtime Model Test service is implemented as a SOAP web service based on a request and response payload structure. In this service model, a calling application provides an input test model XML payload to the `testModel` public synchronized method of the Configurator Runtime Model Test service, then after executing the requests, the service returns a corresponding result XML payload which the calling application can introspect for the result of the requests.

Features of the test service include:

- Calling applications can launch the Configurator Runtime Model Test service using any SOAP client. The SSL port is enabled by default for this web service, because it's an external service and will be the default transport for interacting with the web service in the cloud.
- Calling applications can submit model test service requests to the synchronized method `testModel` in a serialized fashion, one request at a time.
- The Configurator Runtime Model Test service can be used in concert with other services or embedded within other applications or services.

The Configurator Runtime Model Test service's input and output payloads use a SOAP XML structure.

The Configurator Runtime Model Test service has been implemented as an autonomous service and doesn't require any other external data model entities. The service WSDL can be retrieved using the following URL format:

```
https://<hostname>:<port>/fscmRestApi/ConfiguratorRuntimeService?wsdl
```

This service supports only one synchronized method, `testModel`, which has a specific request and response payload structure.

The `testModel` method requires a request input payload which includes the necessary information to:

- Start or restore a configuration
- Perform operations on that configuration
- Query the configuration for attributes
- Save and request a configuration summary

The `testModel` method subsequently returns a response output payload which includes:

- The result of the overall test
- Each individual operation requests

- Query requests
- Configuration summary requests

Test Model Request Structure

The following code example shows a test model request in SOAP XML:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
 <soap:Body>
 <ns1:testModel xmlns:ns1="http://xmlns.oracle.com/apps/scm/configurator/runtimeService/types/">
 <ns1:request xmlns:ns2="http://xmlns.oracle.com/apps/scm/configurator/runtimeService/">
 <ns2:TestId>1</ns2:TestId>
 <ns2:TestName>test</ns2:TestName>
 <ns2:TestDescription>Test1</ns2:TestDescription>
 <ns2:InitializationParameters>
 <ns2:TestId>1</ns2:TestId>
 <ns2:ParameterId>1</ns2:ParameterId>
 <ns2:CallingApplicationCode>CZ</ns2:CallingApplicationCode>
 <ns2:Header>
 <![CDATA[
 <header>
 <HeaderId>1000</HeaderId>
 </header>
 ]] >
 </ns2:Header>
 <ns2:Line>
 <![CDATA[
 <line>
 <LineId>-22222</LineId>
 <InventoryItemNumber>zCZ-CN82441</InventoryItemNumber>
 <InventoryOrganizationCode>V1</InventoryOrganizationCode>
 <UnitQuantity>1.0</UnitQuantity>
 <UomCode>Ea</UomCode>
 </line>
 ]] >
 </ns2:Line>
 <ns2:CustomParameters>
 <![CDATA[
 <param>
 <WorkspaceName>My Workspace</WorkspaceName>
 <SnapshotBasedModel>true</SnapshotBasedModel>
 </param>
 ]] >
 </ns2:CustomParameters>
 </ns2:InitializationParameters>
 <ns2:OperationRequest>
 <ns2:OperationId>1</ns2:OperationId>
 <ns2:NodePath>zCZ-OC12100.zCZ-CM12240</ns2:NodePath>
 <ns2:Operation>Select</ns2:Operation>
 <ns2:SequenceNumber>1</ns2:SequenceNumber>
 <ns2:testId>1</ns2:testId>
 <ns2:NodeValue>1</ns2:NodeValue>
 </ns2:OperationRequest>
 <ns2:QueryRequest>
 <ns2:QueryId>1</ns2:QueryId>
 <ns2:NodePath>zCZ-OC12100.zCZ-CM12240</ns2:NodePath>
 <ns2:SequenceNumber>2</ns2:SequenceNumber>
 <ns2:testId>1</ns2:testId>
 <ns2:NodeProperty>
 <ns2:PropertyId>1</ns2:PropertyId>
 <ns2:PropertyName>Selected</ns2:PropertyName>
 <ns2:QueryId>1</ns2:QueryId>
 </ns2:NodeProperty>
 </ns2:QueryRequest>
```

```
<ns2:ConfigSummaryRequest>
<ns2:SummaryId>1</ns2:SummaryId>
<ns2:TestId>1</ns2:TestId>
<ns2:OutputMode></ns2:OutputMode>
<ns2:SequenceNumber>3</ns2:SequenceNumber>
</ns2:ConfigSummaryRequest>
</ns1:request>
</ns1:testModel>
</soap:Body>
</soap:Envelope>
```

The basic structure of a test model request includes the elements described in the following table:

Element of the testModel Request	Description
TestId	The unique identifier for the current test. The same test ID is also used in other elements of the test request to connect them with a given test.
TestName	A short name for this test
TestDescription	An optional description for this test
InitializationParameters	Includes all the information necessary to start a configuration session
OperationRequest	One or more operations to be performed during this test
QueryRequest	One or more queries to be performed after an operation request has been performed
ConfigSummaryRequest	The configuration summary which includes all the items that you can include for the configuration

InitializationParameters

InitializationParameters is an XML element that models an attribute of the test model request.

InitializationParameters is a required attribute that includes the necessary information to start or restore a configuration session. The following table lists the initialization parameters, both necessary and optional, to start a configuration.

Field name in InitializationParameters	Description
TestId	The unique identifier for the current test.
ParameterId	A unique identifier provided by the service to refer to this initialization parameters set
CallingApplicationCode	Indicates the calling application launching the Configurator session. This is used to determine the user interface to use for the session. This code is registered in the Manage Trading Community Source Systems in the case of external non-Oracle system or the FND Application short name (code) for Oracle Applications. Note: For CPQ there is already a seeded TCA data with application code as ORA_BM_CPQ

Field name in InitializationParameters	Description
Line	An XML document representing a line to be configured. See the following section for additional information
Header	(Optional) Represents a quote or order header that contains the Line. This XML document is optional and when present Configurator passes the header information to other services like Oracle Pricing for pricing integration within a configuration session
CustomParameters	(Optional) A XML document consisting of additional information to Configurator for use within Configurator Extensions rules

The XML document `Line` represents a line to be configured. It contains, at a minimum, the information in the following list.

- LineId: Unique identifier for the line being configured
- InventoryItemNumber: Item number of the product or service model item to configure
- InventoryOrganizationCode: Item validation organization code of the product or service model item. This is typically the organization from which items are imported or referenced from external applications.
- UnitQuantity: Quantity of the item being configured
- UomCode: Unit of measure for the item being configured
- ConfigHeaderId: Identifier for the configuration, passed only during reconfiguration, or a session for restoring and validating
- ConfigRevisionId: Identifier for the configuration, passed only during reconfiguration/restore and validation session
- RequestOn (Optional): Date and time used to load the model definition for the configuration session

OperationRequest

OperationRequest is an XML element that models an attribute of the test model request.

After a configuration has started, you can perform a series of operations using the test service, similar to the operations that an end user can perform when configuring the model using a runtime user interface. A test model request can have one or more operation requests. These operations can range from toggling a Boolean feature, to setting the quantity of a standard item, to adding an instance of a referenced model and configuring it.

The following table lists the attributes of an OperationRequest.

Field name in OperationRequest	Description
TestId	The unique identifier for the current test.
OperationId	A unique identifier for the operation request
NodePath	The fully qualified path to the node on which the operation will act on

Field name in OperationRequest	Description
Operation	The name of the operation, such as. Select, Toggle, SetValue, or SetQuantity. See the following table for valid operations.
SequenceNumber	The sequence number to be used during the execution to prioritize this request
NodeValue	The value to be applied to the node for this test.

The following table lists the operations that you can perform within an OperationRequest, and the node types that you can perform the operations on.

Valid Operations for OperationRequest	Node Type
Select, Toggle, SetQuantity	Option Class, Option Feature, Standard Item, or Option
Toggle	Boolean Feature
SetValue	Text Feature, Integer Feature, Decimal Feature, or Transactional Attribute (All types)
AddInstance, AddInstanceWithQuanity, AddInstanceWithQuanityAndName, RenameInstance, CopyInstance, DeleteInstance, SetQuanity, SetContextPath	Model Node Reference

The operations that can be performed at the configuration level include those in the following list.

- AutoComplete: Performs an autocomplete operation on the configuration, to include all the adjustments needed to make the configuration valid.
- UndoAutoComplete: Reverts the autocomplete operation.
- AdjustConfiguration: Puts the configuration into adjust mode after autocomplete, allowing changes to the automatically completed selections.
- Save: Saves the configuration and produces a header and revision ID.
- Finish: Finishes the configuration, by saving and closing it.

QueryRequest

QueryRequest is an XML element that models an attribute of the test model request.

In order to determine the result of an operation request, you can perform a series of query requests to introspect the node on which the operation request acted, or any other node in the model hierarchy that was changed as a result. For example, if you toggle a Boolean feature using an operation request and subsequently a rule causes a standard item to be selected, then you can add a query request to retrieve the state of that standard item.

The following table lists the fields in a QueryRequest.

Field Name in QueryRequest	Description
TestId	The unique identifier for the current test.
QueryRequestId	A unique identifier for the query request in the test
NodePath	The fully qualified path to the node on which the query operation will act
SequenceNumber	The sequence number to be used during the execution to prioritize this request
NodeProperty	The property of the node returned in the result payload. See the following table for additional information.

Each query request can include one or many node properties to be retrieved. These node properties can be of either scalar type, such as a string or number value, or a collection or list type, in which the result of the query request will return a list of values. For example, in order to determine the selection state of a standard item, the node property named `Selected` can be requested through a NodeProperty, as illustrated in the following example.

```
<ns2:NodeProperty>
<ns2:PropertyId>1</ns2:PropertyId>
<ns2:PropertyName>Selected</ns2:PropertyName>
 <ns2:QueryId>1</ns2:QueryId>
</ns2:NodeProperty>
```

The following table lists the fields in a NodeProperty request.

Field Name in NodeProperty	Description
PropertyId	A unique identifier for the property in the request
PropertyName	The name of the node property to be evaluated. Refer to the list of node properties for the properties supported for each node type.
QueryId	A unique identifier for the query request
CollectionProperty	A list of property requests if the named property being evaluated is a list or a collection type (such as AvailableChildren, SelectedChildren, or SelectableChildren). See the following description of CollectionProperty for additional information

Similar to the way that you select node properties, you request the scalar properties of a list or collection node property type, by adding the property name to the CollectionProperty element's list of properties, as illustrated in the following example.

```
<ns2:NodeProperty>
 <ns2:PropertyId>1</ns2:PropertyId>
 <ns2:PropertyName>SelectedChildren</ns2:PropertyName>
 <ns2:QueryId>1</ns2:QueryId>
 <ns2:CollectionProperty>
 <ns2:PropertyId>1</ns2:propertyId>
 <ns2:CollectionPropertyId>1</ns2:CollectionPropertyId>
```

```
<ns2:PropertyName>Name</ns2:PropertyName>
</ns2:CollectionProperty>
<ns2:CollectionProperty>
<ns2:PropertyId>1</ns2:propertyId>
<ns2:CollectionPropertyId>2</ns2:CollectionPropertyId>
<ns2:PropertyName>Description</ns2:PropertyName>
</ns2:CollectionProperty>
</ns2:NodeProperty>
```

The following table lists the elements that identify a CollectionProperty.

Field Name in CollectionProperty	Description
PropertyId	The unique identifier for the property name of the collection node property to be evaluated.
PropertyName	A property name from the collection node property to be evaluated
CollectionPropertyId	A unique identifier for the collection node property

ConfigSummaryRequest

ConfigSummaryRequest is an XML element that models an attribute of the test model request.

A test model request can include an optional request to retrieve the configuration summary. The configuration summary can be either brief or full, as controlled by the output mode.

The following table lists the fields of the ConfigSummaryRequest element.

Field Name in ConfigSummaryRequest	Description
TestId	The unique identifier for the current test.
SummaryId	A unique identifier for the summary request
OutputMode	The configuration summary output mode. The allowed values are: • Brief: Include only configuration-level information such as the validity of the configuration, and the header ID and revision ID of the configuration object. • Full (the default): Include the brief output and also all of the items that you can include in the configuration.
SequenceNumber	The sequence number to be used during the execution to prioritize this request

Test Model Response Structure

The model test response payload structure directly corresponds to the test model request payload. For every operation or query request there is a corresponding operation or query result that can be correlated to its request counterpart using the unique operation or query ID.

The test model response includes test level information such as the overall test status and any failure messages that prevented the configuration from starting. This level of information is also available at each operation or query result, which allows for introspection of the results at a much more granular level.

The following code example shows a test model response in SOAP XML:

```
<env:Envelope xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
 xmlns:wsa="http://www.w3.org/2005/08/addressing">
 <env:Header>
...
</env:Header>
<env:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
 wsu:Id="Body-WCk1o7Bv0d7m0fziDdW7lQ22">
 <ns0:testModelResponse xmlns:ns0="http://xmlns.oracle.com/apps/scm/configurator/runtimeService/types/">
 <ns1:result xmlns:ns0="http://xmlns.oracle.com/apps/scm/configurator/runtimeService/"
 xmlns:ns1="http://xmlns.oracle.com/apps/scm/configurator/runtimeService/types/"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="ns0:TestModelResponse">
 <ns0:TestId>1</ns0:TestId>
 <ns0:TestName>test</ns0:TestName>
 <ns0:TestDescription>Test1</ns0:TestDescription>
 <ns0:TestStatus>SUCCESS</ns0:TestStatus>
 <ns0:TestFailureMessage xsi:nil="true"/>
 <ns0:OperationResult>
 <ns0:OperationId>1</ns0:OperationId>
 <ns0:NodePath>zCZ-OC12100.zCZ-CM12240</ns0:NodePath>
 <ns0:Operation>Select</ns0:Operation>
 <ns0:SequenceNumber>1</ns0:SequenceNumber>
 <ns0:NodeValue>1</ns0:NodeValue>
 <ns0:OperationStatus>SUCCESS</ns0:OperationStatus>
 <ns0:OperationFailureMessage xsi:nil="true"/>
 <ns0:TestId>1</ns0:TestId>
 <ns0:ContextPath xsi:nil="true"/>
 </ns0:OperationResult>
 <ns0:QueryResult>
 <ns0:QueryId>1</ns0:QueryId>
 <ns0:NodePath>zCZ-OC12100.zCZ-CM12240</ns0:NodePath>
 <ns0:QueryResultStatus>SUCCESS</ns0:QueryResultStatus>
 <ns0:QueryResultFailureMessage xsi:nil="true"/>
 <ns0:TestId>1</ns0:TestId>
 <ns0:SequenceNumber>2</ns0:SequenceNumber>
 <ns0:ContextPath xsi:nil="true"/>
 <ns0:NodeProperty>
 <ns0:PropertyId>1</ns0:PropertyId>
 <ns0:PropertyName>Selected</ns0:PropertyName>
 <ns0:PropertyValue>true</ns0:PropertyValue>
 <ns0:QueryId>1</ns0:QueryId>
 <ns0:PropertyResultStatus>SUCCESS</ns0:PropertyResultStatus>
 <ns0:PropertyResultFailureMessage xsi:nil="true"/>
 </ns0:NodeProperty>
 </ns0:QueryResult>
 <ns0:ConfigSummaryResult xmlns:ns1="http://xmlns.oracle.com/adf/svc/types/">
 <ns0:SummaryId>1</ns0:SummaryId>
 <ns0:TestId>1</ns0:TestId>
 <ns0:ConfigHeaderId xsi:nil="true"/>
 <ns0:ConfigRevisionId xsi:nil="true"/>
 <ns0:ValidConfigurationFlag>false</ns0:ValidConfigurationFlag>
 <ns0:SequenceNumber>3</ns0:SequenceNumber>
 <ns0:RequestOn>2016-12-30</ns0:RequestOn>
 <ns0:ConfigLine>
 <ns0:LineId>1</ns0:LineId>
 <ns0:ParentLineId xsi:nil="true"/>
 <ns0:InventoryItemId>300100017155319</ns0:InventoryItemId>
 <ns0:InventoryOrganizationId>204</ns0:InventoryOrganizationId>
 <ns0:InventoryItemType>1</ns0:InventoryItemType>
```

```
<ns0:LineQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:LineQuantity>
<ns0:UOMCode>Ea</ns0:UOMCode>
<ns0:ConfiguratorPath>300100017155319</ns0:ConfiguratorPath>
<ns0:SummaryId>1</ns0:SummaryId>
<ns0:InventoryItemNumber>zCZ-CN82441</ns0:InventoryItemNumber>
<ns0:InventoryOrganizationCode>V1</ns0:InventoryOrganizationCode>
<ns0:UnitQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:UnitQuantity>
</ns0:ConfigLine>
<ns0:ConfigLine>
<ns0:LineId>2</ns0:LineId>
<ns0:ParentLineId>1</ns0:ParentLineId>
<ns0:InventoryItemId>300100017233297</ns0:InventoryItemId>
<ns0:InventoryOrganizationId>204</ns0:InventoryOrganizationId>
<ns0:InventoryItemType>2</ns0:InventoryItemType>
<ns0:LineQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:LineQuantity>
<ns0:UOMCode>Ea</ns0:UOMCode>
<ns0:ConfiguratorPath>300100017155319.300100017233297</ns0:ConfiguratorPath>
<ns0:SummaryId>1</ns0:SummaryId>
<ns0:InventoryItemNumber>zCZ-OC12100</ns0:InventoryItemNumber>
<ns0:InventoryOrganizationCode>V1</ns0:InventoryOrganizationCode>
<ns0:UnitQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:UnitQuantity>
</ns0:ConfigLine>
<ns0:ConfigLine>
<ns0:LineId>3</ns0:LineId>
<ns0:ParentLineId>2</ns0:ParentLineId>
<ns0:InventoryItemId>300100017232813</ns0:InventoryItemId>
<ns0:InventoryOrganizationId>204</ns0:InventoryOrganizationId>
<ns0:InventoryItemType>4</ns0:InventoryItemType>
<ns0:LineQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:LineQuantity>
<ns0:UOMCode>Ea</ns0:UOMCode>
<ns0:ConfiguratorPath>300100017155319.300100017233297.300100017232813</ns0:ConfiguratorPath>
<ns0:SummaryId>1</ns0:SummaryId>
<ns0:InventoryItemNumber>zCZ-CM12240</ns0:InventoryItemNumber>
<ns0:InventoryOrganizationCode>V1</ns0:InventoryOrganizationCode>
<ns0:UnitQuantity xmlns:tns="http://xmlns.oracle.com/adf/svc/errors/"
unitCode="Ea">1</ns0:UnitQuantity>
</ns0:ConfigLine>
</ns0:ConfigSummaryResult>
</ns1:result>
</ns0:testModelResponse>
</env:Body>
</env:Envelope>
```

Related Topics

- Test the Model
- Use Node Properties to Affect Runtime Behavior and Results
- Add a Supplemental Structure
- Chunk Your Large Option Classes

Manage Your Validations

Manage how and when Configurator validates your configurator model.

Configurator might need to validate your model when you import, create, revise, or submit a sales order that contains a configured item in Oracle Order Management, or when Order Management explicitly requests a validation.

Sometimes you might not need to validate the model. Assume you create a draft sales order that has a configured item, Configurator validates the configuration, but you don't submit the order because your customer isn't ready to commit to the purchase, isn't sure what configuration they need, needs more time to mull over the purchase, and so on. Several weeks later you revise the order with the customer's latest changes, and then submit the order. Configurator already validated the configuration on the draft order, so it can quickly examine what's different in the configuration on the submitted order to determine whether it can skip validation.

If you sell lots of configured items, or large complex configured items, then you can skip validation under specific conditions to realize performance improvements and make your deployment more efficient.

Skip Validation

Specify whether to skip validation for a model.

1. Open your model for editing in a workspace that you haven't released.
2. On the Edit Configurator Model page, click **Structure**, select the model's root node, then click **Preferences**.
3. Make sure the Skip Validation When It's Not Required option contains a check mark.
4. Save the model and release the workspace.

If you don't enable Skip Validation When It's Not Required, then configurator will validate your runtime configuration regardless of whether or not you have made any changes to your model that affect the runtime configuration.

Regardless of whether you enable the option, you might need to specify the Configuration Effective Date.

Specify the Configuration Effective Date

You can use the Configuration Effective Date parameter to identify the version that you want to validate.

You might have different versions of your model, and you might set each version's start and end dates differently. Assume you have:

- Version 1, in effect June 1 to June 10.
- Version 2, in effect June 11 to June 15.
- Version 3, in effect June 16 to June 30.

Configurator compares the version that you submit to the version that exists according to the Configuration Effective Date parameter.

Try It

1. Go to the Setup and Maintenance work area, then go to the Manage Order Management Parameters task:
    - Offering: Order Management
    - Functional Area: Orders
    - Task: Manage Order Management Parameters

2. Set the value for the Configuration Effective Date parameter.

Value	Description
Configuration Date	Use the version that's in effect on the date that you add the order line and configure the item on the line. Assume you create an order line, add the Stove to the line and configure the stove on June 1. The configuration date is June 1. In our example, Configurator will use Version 1 because Version 1 is in effect on June 1. If you configure the stove on June 11, then Configurator will use Version 2.
Ordered Date	Use the version that's in effect on the ordered date. Order Management sets the Ordered Date to the date when you create the order, by default, but you can manually change it before you submit the order. For example, if the Ordered Date attribute on the order header contains June 16, then use the version that's in effect on June 16. In our example, that's Version 3.
Current Date	Use the version that's in effect on the current date. For example, if the current date is June 1, 2023, then use the version that's in effect on June 1, 2023. The current date is the date that you submit the order to fulfillment. Assume you create an order on June 1, save it as a draft but don't submit it until June 10. The ordered date is June 1 and the current date is June 10. In our example, if you set the parameter to Current Date, then Configurator won't validate the model because it already validated Version 1 on June 1 and Version 1 is still in effect. If you don't set the parameter to any value, then Order Management uses the current date, by default.
Requested Date	Use the version that's in effect on the requested date. For example, if the Requested Date attribute on the order line contains July 14, then use the snapshot that's in effect on July 14. In our example, that's Version 2.

For most deployments, we recommend that you set the Configuration Effective Date parameter to Current Date because that's the date that you submit the order. The configuration that's on the order line when you submit usually includes the configuration you need because it typically includes any changes that you might have made to the configuration to reflect your customer's current requirements.

You might have some specific requirements to use another value. For example, Requested Date is the date the customer wants to receive item. If you plan to modify your model between the time you configure it and the time you actually deliver it, and you want to apply that modification before you deliver it, then you could use Requested Date.

For details, see *Manage Order Management Parameters*.

Changes That Affect Validation

Configurator looks at the changes that you make on each version, then compares them to the version it already validated. It uses the Configuration Effective Date parameter to determine which version to look at.

Assume:

- You set Configuration Effective Date to Current Date.
- You update the Maximum Quantity attribute on a component that's part of your model, and you specify when the change goes into effect on the model.
- You revise the sales order in Order Management.

If:

Change on Attribute Value Goes Into Effect	Will Configurator Validate?
On or before the current date.	Yes
After the current date.	No

Example

Assume your model has this hierarchy.

```
Stove
 Oven
 10K BTU Gas Burner
 20K BTU Gas Burner
 Stovetop
 10K BTU Gas Burner
 20K BTU Gas Burner
```

Assume:

- You find through market research that customers want to have up to four 10K BTU Gas Burners on the stove top instead of only three, so you modify the Maximum Quantity attribute on the 10K BTU Gas Burner from 3 to 4, and you set the Maximum Quantity's Start Date so your new quantity goes into effect on June 15. The new start date gives you enough time to increase supply for the 10K burners and provides time for your factory to adjust to the new workflow they'll use to add another burner.
- You create sales order 56497 on June 1, so the Ordered Date is June 1. You add an order line for a refrigerator and save the order as a draft because the customer says they might want to add the stove in a day or two.
- You create a new order line on June 2, and the Stove on the line, configure the stove, and save the order as a draft. So, the Configuration Date is June 2.
- The Requested Date on the order line is June 20.
- You don't submit the order until June 10, so the Current Date is June 10.

Configurator might or might not validate when you submit the order depending on how you set the Configuration Effective Date parameter.

Set Configuration Effective Date Parameter To	Validate?
Configuration Date	The Configuration Date is June 2 but the Maximum Quantity's start date doesn't happen until June 15, so there's no need to validate your change. Configurator can use the current version, and it already validated that version when you created the draft order. Customers can add no more than three 10K burners on the stove top.
Current Date	The Current Date is June 10 but the Maximum Quantity's start date doesn't happen until June 15, so there's no need to validate your change. Configurator can use the current version, and it already validated that version when you created the draft order. Customers can add no more than three 10K burners on the stove top.
Ordered Date	The Ordered Date is June 1 but the Maximum Quantity's start date doesn't happen until June 15, so there's no need to validate your change. Customers can add no more than three 10K burners on the stove top.
Requested Date	The Requested Date is June 20, the Maximum Quantity's start date happens on June 15 which is before the requested date, so the configuration needs to consider your modified quantity. Configurator will validate your change. Customers can add up to four 10K burners on the stove top.

The Maximum Quantity attribute	Configurator Will
After the Requested Date on the sales order.	Skip the validation.
Before the Requested Date on the sales order.	Validate the configuration.

Skip Validation for Models That You Don't Import from Product Information Management

Configurator examines different attributes to determine if it can skip validation depending on whether you import your model from the Product Information Management work area into the Configurator Models work area or create it directly in the Configurator Models work area.

If you don't import your model from Product Information Management, but instead create it in the Configurator Models work area, then Configurator can skip validation when:

- You haven't changed the model.
- You haven't changed any of the attributes on the model's child components.

Configurator examines these attributes on the model to see if you modified them:

- Order Management Indivisible

- Structure Item Type

Configurator also examines these attributes on the model's child components to see if you modified them:

- Quantity
- Minimum Quantity
- Maximum Quantity
- Optional
- Mutually Exclusive Options
- Instantiability. The value can be Multiple Instances, Optional Single Instance, or Requires Single Instance.

For details about some of these attributes, see *Valid Component Attributes and Structure Types*.

Configurator also determines whether you end dated any of the model's child components or removed any component from the model.

Skip Validation for Models That You Import from Product Information Management

If you import a model into the Configurator Models work area as a snapshot from the Product Information Management work area, and if you modify and release the snapshot, then Configurator:

- Examines the changes that you released to determine whether your updates affect the model's item structure, rules, or user interface.
- Determines whether you updated and released any snapshots for the model's child items.

If Configurator finds that you made changes that affect the snapshot, then it won't skip validation. Instead, it will validate your changes.

Guidelines for Not Skipping Validation

Make sure you don't skip validation when:

- The effective date on the configuration's run time instance happens on or after the effective date of the configuration's most recent revision, and before you create, import, or submit the sales order, or when Order Management explicitly requests a validation. If you change the ordered date or the requested date on the sales order after Configurator initially created or validated the configuration, then you must validate the run time configuration.
- You release an item class or a value set in you snapshot after you initially create or validate the configuration and before you create, import, revise, or submit the sales order.
- Your model contains an extension rule that calls an external application and that call modifies the configuration.
- The configuration involves a transactional attribute. See *Transactional Attributes*.

If any of these situations apply to your model, then make sure the Skip Validation When It's Not Required option doesn't contain a check mark.

Related Topics

- Item Order Management Specifications

Test the Model

Calling applications can launch the Configurator Runtime Model Test service using any SOAP client. The instructions here describe the process for creating a web service proxy and data control to run the service within the context of a jUnit test case for Oracle JDeveloper.

The required process is:

1. Create a web service proxy.
2. Create a web service data control.
3. Create a jUnit.

The details for these steps follow.

Creating a Web Service Proxy

The first step is to create a web service proxy using the Oracle JDeveloper Web Service Proxy wizard. Using a web service's WSDL, this wizard creates all the necessary Java objects for the request and response payloads and calling the web service.

To create a web service proxy:

1. Create a generic application and project in Oracle JDeveloper.
2. Select the project in your application and click **File** > **New**.
3. Search for the Web Service Proxy project technology, select it and click **OK**.
4. Click **Next** on the wizard's overview page.
5. Leave the Client Style as JAX-WS Style and click **Next**.
6. Enter the URL to the Web Services Description Language (WSDL) endpoint on the host where the Configurator Runtime Test Model service is deployed (in a cloud deployment this would be on the HTTPS port) then click **Next**.
7. On the Specify Default Mapping Options step, deselect the **Generate As Async** option (because asynchronous invocation isn't supported) and click **Next** twice, to navigate through the steps related to port endpoints without changing the options
8. On the Asynchronous Methods step, be sure to select the **Don't generate any asynchronous methods** option, then click **Next**.
9. On the Policy step, wait for the list of policies to be populated. Select **oracle/wss11_username_token_with_message_protection_client_policy** from the list. Click **Next**.
10. On the Defined Handlers step, click **Next**. Finally, click **Finish**.
11. The wizard creates all the necessary Java objects needed for constructing the request and response payloads needed to interact with the web service. See the description of creating a jUnit for an example of how these objects can be used.

Creating a Web Service Data Control

A web service data control is an effective way to create a connection using the connection factory object in order to interact with a web service. A data control encapsulates the security policies, credentials and the data model needed when calling the service. With a properly prepared data control, all that's required for a client to call the web service is to retrieve a service connection using its Java Naming and Directory Interface (JNDI) name. See the description of creating a jUnit for an example of how to call a web service using the service connection factory.

To create a web service data control:

1. Create and select a project where the web service data controls will be created, then click **File New**.
2. Search for the Web Service Data Control project technology, select it, then click **OK**.
3. Enter a name for the data source, which will later be used as the JNDI name for looking up this web service connection. Provide the URL to the WSDL endpoint on the host where the Configurator Runtime Test Model service is deployed (in a cloud deployment this would be on the HTTPS port) then click **Next**.
4. On the Data Control Operations step, select and move the **testModel** operation from the Available operations list to the Selected operations list. Click **Next**.
5. On the Response Format step, click **Next**.
6. On the Endpoint Authentication step, select **ConfiguratorRuntimeServiceSoapHttpPort**, then enter the user name and password for the web service. Click **Next**.
7. Click **Finish**. The web service data control wizard creates all the type XML files and adds a connection reference in the `connections.xml` file of the calling application.

Creating a jUnit

There are many ways to call a web service. Here, the vehicle described for calling the Configurator Runtime Model Test service is a jUnit. jUnits are an effective way to perform repeatable tests for this service. Also, Oracle JDeveloper has full support for the Apache jUnit 4 framework.

To build a simple jUnit in Oracle JDeveloper,

1. Select an existing project, or create a new project, in your application where you want to create the jUnit.

   > **Note:** Ensure that the web service proxy and web service data control projects that you previously created are in this project's library path. It's essential that your jUnit have access to the object in those projects.

2. Search for and select the Test Case project technology. Click **OK**.
3. In the **Class Under Test** list, select **<None>**. Click **Next**.
4. Enter a name and package for the jUnit. Select the standard jUnit setup and tear-down methods to be generated. Click **Next** to continue.
5. Click **Finish**. The test case wizard creates the jUnit Java class in your project.
6. The content of the jUnit should be similar to the following code example.

```
package oracle.apps.scm.cz;
import org.junit.After;
import org.junit.AfterClass;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.BeforeClass;

public class MyServiceTest {
 public MyServiceTest() {
 }

 @Before
 public void setUp() throws Exception {
 }

 @After
 public void tearDown() throws Exception {
 }

 @BeforeClass
 public static void setUpBeforeClass() throws Exception {
```

```
 }

 @AfterClass
 public static void tearDownAfterClass() throws Exception {
 }
}
```

7. Add a test method to the jUnit that's similar to the following code example.

```
@Test
public void testMyTest() {
 ConfiguratorRuntimeService_Service configuratorRuntimeService_Service;
 // Insert the name of the file that includes the model test request XML
 File testModelFile = new File("filename.xml");

 TestModelResponse response = null;
 try {
 // 1. Lookup the web service connection using its JNDI name
 Context ctxnew = ADFContext.getCurrent().getConnectionsContext();
 WebServiceConnection wsc =
 (WebServiceConnection)ctxnew.lookup("WebServiceJNDIName");

 // 2. Get a reference to the web service port
 configuratorRuntimeService_Service =
 new ConfiguratorRuntimeService_Service();
 ConfiguratorRuntimeService configuratorRuntimeService =
 wsc.getJaxWSPort(ConfiguratorRuntimeService.class);

 // 3. Create a test model request using JAXB
 JAXBContext jaxbContext = JAXBContext.newInstance(TestModel.class);
 Unmarshaller jaxbUnmarshaller = jaxbContext.createUnmarshaller();
 TestModel testModel =
 (TestModel)jaxbUnmarshaller.unmarshal(testModelFile);

 // 4. Call the web service method
 TestModelResponse testModelResponse = new TestModelResponse();
 testModelResponse =
 configuratorRuntimeService.testModel(testModel);

 } catch (Exception e) {
 Assert.fail(e.getMessage());
 }

 // 5. Introspect the response object as needed
 JAXBElement<String> status = response.getResult().getTestStatus();
 if (!"SUCCESS".equalsIgnoreCase(status.getValue()))
 Assert.fail(response.getResult().getTestFailureMessage().getValue());

}
```

8. Be sure to provide the relative path to the model test request XML file and the correct JNDI name to look up the web service connection. The JNDI name is the name you provided when you created the web service data control.
9. Run the Java class as a jUnit within Oracle JDeveloper.

Developer Testing Recommendations

The Configurator Runtime Model Test service is an autonomous SOAP-based web service; it can be called from any SOAP client.

Because the Configurator Runtime Model Test service is deployed as an external service in the cloud environment, the exposed port will be enabled for SSL (Secure Sockets Layer) by default. We strongly recommend that you build and test client applications using the SSL port.

To ensure that the initial SSL handshake between the Java client application (such as jUnit) and the web service is established, you must import the SSL certificate where the Configurator Runtime Model Test service is hosted into the local Java key store where the Java client application is running.

To ensure that the proper SSL certificate for the Configurator Runtime Model Test service is available during the client-server SSL handshake, perform the following steps:

1. Access the Configurator Runtime Model Test service on the SSL port from a standard browser. For an example using Firefox:
    a. Select **Tools** > **Options** > **Security**
    b. Click **View Certificate**.
2. Export the server SSL certificate as a certificate in the PEM (Privacy-Enhanced Mail) security format.
    a. On the Details tab of the Certificate Viewer dialog box, click **Export**.
    b. Save the PEM file to a desired location.

    ```
    my_server_com.PEM
    ```
3. Using the Java `keytool` utility, import the certificate file to the local Java keystore where the Java client application is running.

    ```
    keytool -import -trustcacerts
     -file my_server_com.PEM
     -keystore $JAVA_HOME/lib/DemoTrust.jks
    ```

Related Topics

- Guidelines for Testing Your Model
- Use Node Properties to Affect Runtime Behavior and Results

9 Integrate

How You Integrate with Other Applications

You can integrate Configurator with an application, such as quoting or order capture, to provide an interactive configuration experience for configuring products and services.

Configurator is agnostic about the technology stack of an external application and implements a standards-based approach for a seamless UI integration experience. The integration approach leverages Representational State Transfer (REST) APIs for creating the intent to configure a complex item and for retrieving the final configuration, and an entry point URL for launching the configurator runtime UI, either within an IFrame or within a browser window. REST API isn't a technology, but rather a client-server architectural style in which participants perform actions on data-oriented resources.

Integration

Integration to launch Configurator from an external application and return after a configuration session consists of the elements described in the following table.

Integration Element	Purpose
REST resources	Provide data-oriented services for: • Creating an intent to start a configuration session • Retrieving configuration details, such as lines, line attributes and messages • Performing custom configuration actions.
External application data model entities	Provide data from the host application that's needed by Configurator to establish integration, such as a unique identifier for a configuration header.
Public entry point URL	Provides the entry point for launching a configurator user interface session.
REST input and output payloads	Carry the specific inputs to and outputs from a configuration session.
Integration tasks	Actions required to complete an integration.

REST Resources

External applications can leverage these Configurator REST resources to interact with configurations

- **Initialization Parameters**: External applications must use this resource to specify the intent to configure a product or service interactively by passing an input payload which provides details that the configurator uses at run time to start a configuration session.

  At a minimum the following information must be posted to this REST resource:

  - CallingApplicationCode: The application code registered in the Manage Trading Community Source Systems task in the case of external non Oracle applications or the FND Application short name (code) for Oracle applications.
  - Line: A JavaScript Object Notation (`JSON`) document serialized in string format. For each field, a name-value pair corresponds to the sales context line entity definition.

  The Initialization Parameters resource is used to load the appropriate model and user interface to launch an interactive configuration session.

  The Initialization Parameters resource can be used only once to launch Configurator, then the resource is marked as expired and can't be reused. The purpose of this restriction is to prevent attacks such as distributed denial of service.
- **Configurations**: External applications must use this resource to retrieve the details of a configuration, using the configuration identifiers that are returned by Configurator after finishing a configuration session.

  The Configurations resource consists of one root entity and encompasses three child entities:

  - ConfigurationLines: a collection resource that describes all the lines in a configuration.
  - ConfigurationLineAttributes: a collection resource that describes all the transactional attribute lines in a configuration.
  - ConfigurationMessages: a collection resource that describes one or more messages added in a configuration.

  The Configurations resource by default filters out option class lines when returning configuration results. An external application can request the full result by specifying the OuptutMode query parameter value as `full`.
- The Configurations resource supports the following custom actions that can be performed on a configuration:

  - **copy**: Copy a configuration mirroring the copy action on a quote/order line.
  - **validate**: Validate a configuration before submitting a quote or order to Oracle Order Management for fulfillment to eliminate quote or order errors before submission.

See the REST API for Oracle Supply Chain Management Cloud guide for full details on:

- Descriptions of parameters and query examples for the Initialization Parameters and Configurations REST endpoints
- Examples of use cases for the custom actions

Note: The REST API for Oracle Supply Chain Management Cloud guide is the comprehensive reference for working with SCM REST resources, and includes information on getting started with REST APIs.

External Application Data Model Entities

To persist configuration-related information, all external applications integrating with Configurator must add the following attributes in their entity that represents a configuration line.

Attribute	Purpose
ConfigHeaderId	Configurator-generated unique identifier for a configuration header
ConfigRevisionId	Configurator-generated unique identifier for a revision of a configuration
ConfiguratorPath	Configurator-generated runtime inventory ID path to an item for a configuration line

The combination of ConfigHeaderId and ConfigRevisionId uniquely identifies a configuration. The identification attributes must be passed to Configurator during reconfiguration flows, and to Oracle Order Management when submitting a quote or order for fulfillment.

In addition to the attributes that identify a configuration, external applications can add the following attributes in their configuration line entity, to synchronize with existing quote or order lines during a reconfiguration session, instead of deleting and recreating the quote or order lines. This information is retrieved using the Configurations resource when you expand the child resources.

Attribute	Purpose
ConfigLineId	Configurator-generated unique identifier for a configuration line
ParentConfigLineId	Configurator-generated unique identifier to indicate the parent configuration line from a child line
ConfigLineAttributeId	Configurator-generated unique identifier for a configuration line attribute representing a transactional attribute associated to an Item.

Public UI Entry Point URL

Configurator provides a public entry point URL to launch the configurator runtime UI, either within an IFrame or within a browser window.

The URL entry point to launch the configurator runtime UI is publicly available, and secured to provide access to an authenticated user. The entry point expects the following URL parameters:

- **ParameterId**: An initialization parameter resource identifier generated by the Initialization Parameters REST API while creating the intent to configure, which identifies a set of initialization parameter attribute-value pairs. If this parameter value isn't specified or is invalid then an error dialog will be displayed to the user and configurator will redirect to the ReturnUrl.
- **ReturnUrl**: A fully qualified URL of the host application to return to after finishing a configuration session. An HTTP 400 error code will be returned if this parameter value isn't passed.

The entry point URL syntax is:

```
<protocol>://<host>:<port>/fscmUI/faces/ConfigLaunch.jspx
```

Example:

```
https://acme.cloud.enterprise.com/fscmUI/faces/ConfigLaunch.jspx?ParameterId=123456&ReturnUrl=https://
acme.cloud.quote.com/context/processResults.jsp
```

In the case of an error either during or while starting a configuration session, Configurator navigates to the `ReturnUrl` with an `ExitStatus` of `error`. The host application can perform a REST GET operation on the Initialization Parameters resource, with the `ParameterId` passed as part of the `ReturnUrl`, to obtain the full error message details.

Integration Tasks

To prepare to call the runtime configurator from a host application, and to prepare and process input and output payloads, follow these high-level steps.

1. Register the external application:

   Go to the Setup and Maintenance work area, then go to the task:

   - Offering: Product Management
   - Functional Area: Product Spoke Systems
   - Task: Manage Trading Community Source Systems

   For details, see *Product Spoke Systems*.

   Then create or register your external application as a source system.

   When an application is registered, you can define applicability parameters for modified user interfaces, in the Configurator Models work area, so that those user interfaces are displayed when the application calls Configurator.
2. Enable the allowed origins white list:

   This step is required if launching the runtime configurator in an IFrame. By default, Oracle Applications will not render contents in an IFrame except when called from the same origin. You can use a configurable white list mechanism to specify the list of allowed external origins that can render contents in an IFrame. You control the allowed origins by setting the profile option `ORACLE.ADF.VIEW.ALLOWED_ORIGINS`, using the Manage Administrator Profile Values task in the Setup and Maintenance work area.

   This step isn't required for launching the runtime configurator in a browser window.

3. Call the runtime configurator UI, in an IFrame or browser window, using the public entry point URL:

   The public entry point URI is `/fscmUI/faces/ConfigLaunch.jspx`. The URL parameters `ParameterId` and `ReturnUrl` are required.

   The following steps are required for calling the runtime configurator.

   a. Prepare an initialization payload.

      Prepare a REST payload using the JSON format.

      Example, showing the minimum required payload:

```
{ "CallingApplicationCode":"CZ",
 "Line":"{\"InventoryItemNumber\":\"AS100\",\"RequestOn\":\"2017-01-22T09:09:28-0700\",
\"InventoryOrganizationCode\":\"V1\"}"
}
```

   b. Post the payload to the Initialization Parameters resource.

      Send the payload to the Initialization Parameters REST resource using the POST operation, using the following URI

```
/fscmRestApi/resources/latest/initializationParameters
```

      You can replace `latest` with the string that specifies the desired installed release of Oracle Supply Chain Management, such as `11.13.18.02`.The string `latest` is a synonym for the latest installed release.

      Example, using the cURL command-line tool:

```
curl ' https://acme.cloud.enterprise.com:7000/fscmRestApi/resources/latest/
initializationParameters' \
-i \
-X POST \
-H "Content-Type:application/json" \
-u myuserid:mypass \
-d '{ "CallingApplicationCode":"CZ",
 "Line":"{\"InventoryItemNumber\":\"AS100\",\"RequestOn\":\"2017-01-22T09:09:28-0700\",
\"InventoryOrganizationCode\":\"V1\"}"}'
```

      For the payload request shown in the example, the response from the Initialization Parameter resource is like the following:

```
{
 "ParameterId": 300100116074154,
 "CallingApplicationCode": "CZ",
 "Header": null,
 "Line": "{\"UnitQuantity\":1,\"InventoryItemNumber\":\"AS100\",\"RequestOn\":
\"2018-01-11T09:09:28-0700\",\"InventoryOrganizationCode\":\"V1\",\"InventoryOrganizationId\":204,
\"InventoryItemId\":300100016171847}",
 "CustomParameters": null,
 "ManualPriceAdjustments": null,
 "PageTitlePrefix": null,
 "ExpiredFlag": false,
 "ErrorMessageText": null,
```

```
 "TerminalActionCaption": null,
 "ValidationCannotAlterFlag": true,
 "ValidationFailFastFlag": true,
 "links": [
 {
 "rel": "self",
 "href": "<a target="_blank" href="https://node001.my.srvr.com:7021/fscmRestApi/resources/
latest/initializationParameters/300100116074154",">https://node001.my.srvr.com:7021/fscmRestApi/
resources/latest/initializationParameters/300100116074154",</a>
 "name": "initializationParameters",
 "kind": "item"
 },
 {
 "rel": "canonical",
 "href": "<a target="_blank" href="https://node001.my.srvr.com:7021/fscmRestApi/resources/
latest/initializationParameters/300100116074154",">https://node001.my.srvr.com:7021/fscmRestApi/
resources/latest/initializationParameters/300100116074154",</a>
 "name": "initializationParameters",
 "kind": "item"
 }
 ]
}
```

c. Extract the ParameterId.

The external application must process the response payload from the POST operation to obtain the parameter identifier `ParameterId` generated by the Initialization Parameters REST resource when initializing the configuration. The value from the example is:

```
"ParameterId": 300100116074154
```

d. Call the public entry point URL.

The value obtained for ParameterId is passed as a URL parameter value when calling the public entry point `ConfigLaunch.jspx`. The resulting example URL, which launches the runtime Configurator user interface in a session that configures the products specified in the initialization payload, is like:

```
https://acme.cloud.enterprise.com/fscmUI/faces/ConfigLaunch.jspx?
ParameterId=300100089663945&ReturnUrl=https://acme.cloud.quote.com/context/processResults.jsp
```

4. The end user interactive configuration session happens at this point.

The end user can complete an interactive configuration session and exit the configurator by using one of the following actions, by clicking the corresponding button in the runtime configurator user interface:

- **Cancel**: The end user can simply cancel the configuration session. A warning dialog is displayed to the end user about losing the changes made to configuration. An exit status of `cancel` is specified in the `ReturnUrl` parameter to indicate to the host application that there no changes to process upon return.
- **Save for Later**: The end user decides to save the configuration to finish it at a later point in time. An exit status of `save` is specified in the `ReturnUrl` parameter to indicate to the host application that there are changes to the configuration. The host application can use the configuration identifiers (`ConfigHeaderId` with `ConfigRevisionId`) to retrieve the configuration details.
- **Finish** or **Finish and Review**: The end user completes the configuration, by implicitly running the autocomplete process, and returns to the host application. An exit status of `save` is specified in the `ReturnUrl` parameter to indicate to the host application that there are changes to the configuration. The host application can use the configuration identifiers (`ConfigHeaderId` with `ConfigRevisionId`) to retrieve the configuration details.

5. Process the configuration results:

   When the end user completes an interactive configuration session and exits, Configurator navigates to the `ReturnUrl` that was provided when launching the configuration session, and appends the configuration identifiers and exit status as URL parameter values. The external application must use the URL parameter values to determine the exit status and take the necessary action to navigate to its appropriate page.

   - The `ExitStatus` parameter indicates the exit method. Possible values are `save`, `cancel` and `error`.
   - When the exit status is `save`, the following configuration identifiers are included:
     - `ConfigHeaderId`: Identifier for the configuration.
     - `ConfigRevisionId`: Identifier for the configuration revision.

     When used together, the combination of `ConfigHeaderId` and `ConfigRevisionId` uniquely identify a configuration. Example:

     ```
     q=ConfigHeaderId=300100112392208;ConfigRevisionId=300100112392209
     ```

   - If the `ExitStatus` is `save`, then the external application must obtain the configuration results by passing URL parameter values to the Configurations REST resource. The URI for the Configurations REST resource is:

     ```
     /fscmRestApi/resources/latest/configurations
     ```

     The required URL parameters are:

     ```
     q=ConfigHeaderId=000;ConfigRevisionId=000
     ```

     Example URL that queries the Configurations REST resource for a specified configuration and returns all its configuration lines, including option class lines (`OutputMode=full`), and transactional attribute lines (`expand=ConfigurationLines,ConfigurationLineAttributes`):

     ```
     https://acme.cloud.enterprise.com:7000/fscmRestApi/resources/latest/configurations?
     q=ConfigHeaderId=300100112392208;ConfigRevisionId=300100112392209;OutputMode=full&expand=ConfigurationLines
     ```

     This REST resource produces a JSON document in the following structure for one configuration:

     ```
     Configuration
      ConfigurationLines (collection)
      ConfigurationLine [0...n]
      ConfigurationLineAttributes (collection)
      ConfigurationLineAttribute [0...n]
      ConfigurationMessages (collection)
      ConfigurationMessage [0...n]
     ```

     The external application must process the JSON document to create quote or order lines from it.

6. Reconfiguration flow.

   An external application can perform a reconfiguration of an existing configuration by following the same integration steps as for a new configuration, but must include the configuration identifiers and configurator path information for the root line in the initialization parameters payload. Example, showing the minimum required payload:

   ```
   {
    "CallingApplicationCode":"ABC_CPQ",
   ```

```
 "Line": "{\"InventoryItemNumber\":\"AS100\", \"InventoryOrganizationCode\":\"V1\",\"RequestOn\":
\"2016-04-07 13:00:00\"}",
\"ConfigHeaderId\":300100112392208, \"ConfigRevisionId\":300100112392209, \"ConfiguratorPath\":
\"987654321\"}
}
```

Based on the exit status, an external application can obtain the configuration results using the configuration identifiers. Instead of deleting and recreating quote or order lines, an external application can use the configuration line identifiers to synchronize the existing lines. The important point to note is that Configurator returns the same line identifiers for reconfiguration flows as for new configuration flows. However, the lines in the configuration aren't ordered, and you can't provide a sorting order.

For details and examples, go to *REST API for Oracle Supply Chain Management Cloud*, then expand:

- Order Management > Configuration Initialization Parameters
- Order Management > Configurations
- Use Cases > Configurator

Pricing with Configurator

You can display the prices for standard items and connected items in your user interface. Pricing is performed within Configurator using Oracle Pricing.

When item prices are displayed during a configuration session, the prices for items or services appear in the running summary pane on the configuration summary page, and with the items themselves. Connected items and services are displayed as root-level lines in the configuration summary. The price for the connected items and services is included in the total net price for the configuration.

The requirements for using pricing with Configurator are:

- You are running Configurator with a host application that passes customer information to Configurator, such as Oracle Order Management or Oracle Configure, Price, and Quote.
- Your items are on a price list, managed in the Pricing Administration work area.

  Price lists define the amount for each item. Pricing strategies assemble price lists to be used when pricing is applied at runtime. A pricing strategy ensures that the correct prices are calculated for a given customer (for instance, for discounts). Pricing strategies are dynamic and differ between customers. When you test a configurator model, the customer information isn't available, so in the Test Model dialog box you can explicitly specify which pricing strategy to use for the test session, which determines which price lists are in effect.
- The pricing of price-listed items in Configurator is on by default. You can set the **Pricing in Configurator** setup parameter to turn pricing off (and back on) in Configurator.

To control whether the prices for items are displayed in Configurator:

1. In the Setup and Maintenance work area, use the Manage Order Management Parameters task:
   - Offering: Order Management
   - Functional Area: Orders
   - Task: Manage Order Management Parameters
2. Select the parameter **Pricing in Configurator**.

3. Select a value for the parameter:
    - **Always:** (Default). Configurator always calls pricing. Prices appear for all configuration nodes that can be priced.
    - **Never:** Configurator never calls pricing. No prices appear while configuring a product.

Related Topics

- Test Models Interactively