

Oracle Fusion Cloud SCM

Modeling Configurations for SCM

23D



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

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

If this is software, software documentation, data (as defined in the Federal Acquisition Regulation), or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software," "commercial computer software documentation," or "limited rights data" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed, or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

Oracle®, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Contents

Get Help	i
<hr/>	
1 Overview	1
Configurator Models	1
Create and Maintain Configurator Models	8
2 Snapshots	17
Snapshots	17
Import Items Into Configurator Models	19
Refresh Your Snapshot	21
3 Workspaces	27
Manage Your Workspace	27
Release Your Workspace	31
Manage Your Workspace Dates	38
Work on the Same Participant in Different Workspaces	40
Manage Workspace Versions	42
Remove Your Model From Production	45
4 Structures	49
Overview	49
Set Up Features and Attributes	50
Manage Structures	67
Use Spreadsheets to Manage Supplemental Structures	73
5 Rules	85
Overview of Model Rules	85
Rule Principles	85
Create Statement Rules	98
CDL Reference	106
Extension Rules	129

FAQ for Model Rules	144
6 User Interfaces	147
Overview of Model User Interfaces	147
User Interfaces for Configurator Models	148
The Default User Interface	149
Generated User Interfaces	149
How You Synchronize UIs with Structure	151
Multichannel User Interfaces	152
Templates, Pages, and Navigation Styles	153
How You Modify User Interfaces	160
How You Modify UI Elements	164
The UI Expression Language	168
How You Use Images for Selections	170
How You Visualize Configurations	171
FAQ for Model User Interfaces	174
7 Connectors	175
Overview of Connectors	175
When To Use Connectors	175
How You Create Connectors	176
How You Use Connectors	178
Connectors in Action	182
How You Filter Connectable Items	184
How You Integrate Connectors	186
What's the difference between a connector and a model reference	188
8 Test	191
Overview of Testing Your Model	191
Test Models Interactively	191
Get Rule Explanations When You Test Your Model	194
Guidelines for Testing Your Model	198
Manage Your Validations	206
Test the Model	212
9 Integrate	217
How You Integrate with Other Applications	217

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

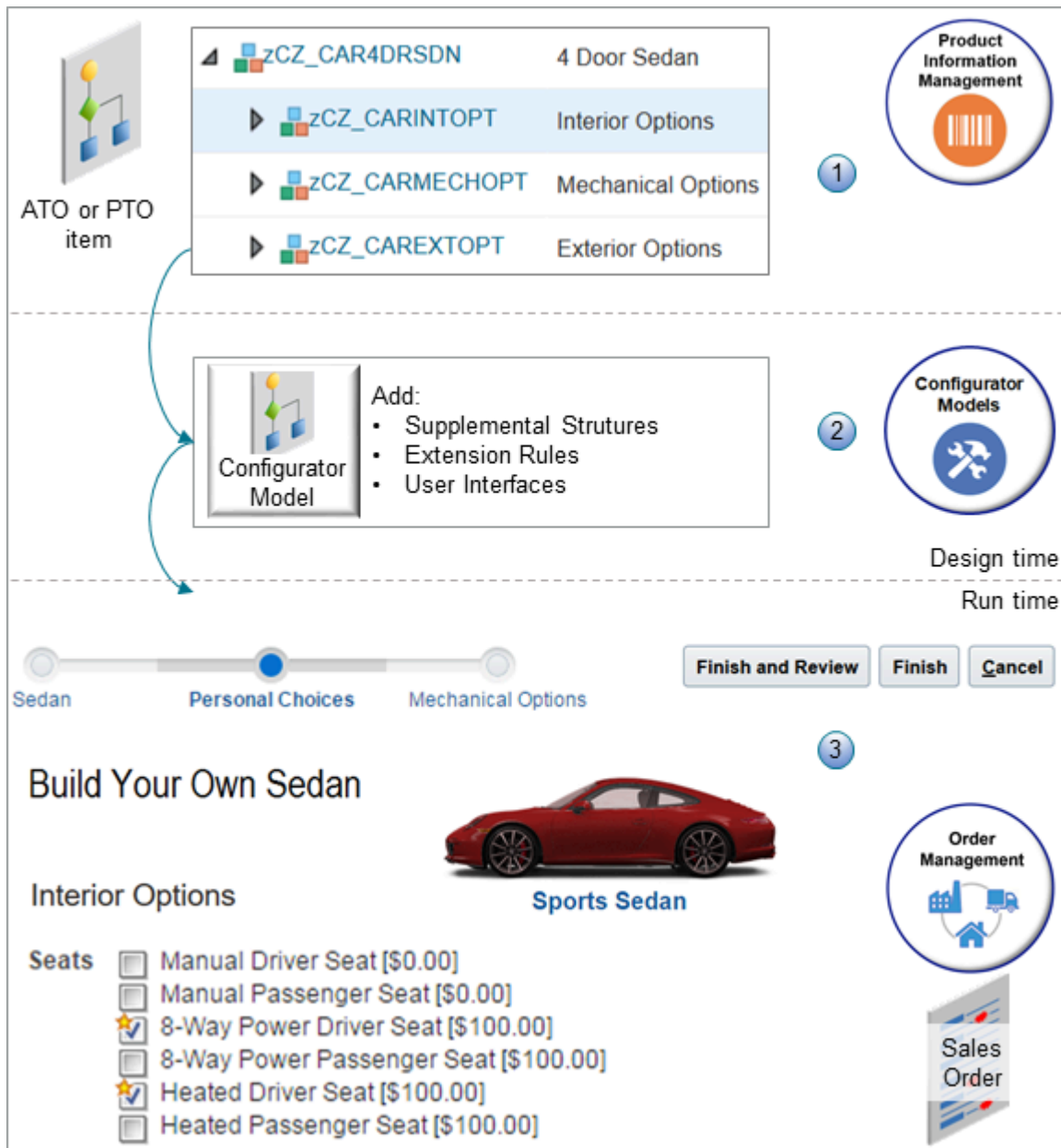
Configurator Models

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.
 - o Add a supplemental structure and rules to help simplify life for your users when they configure the item.
 - o 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.
 - o 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

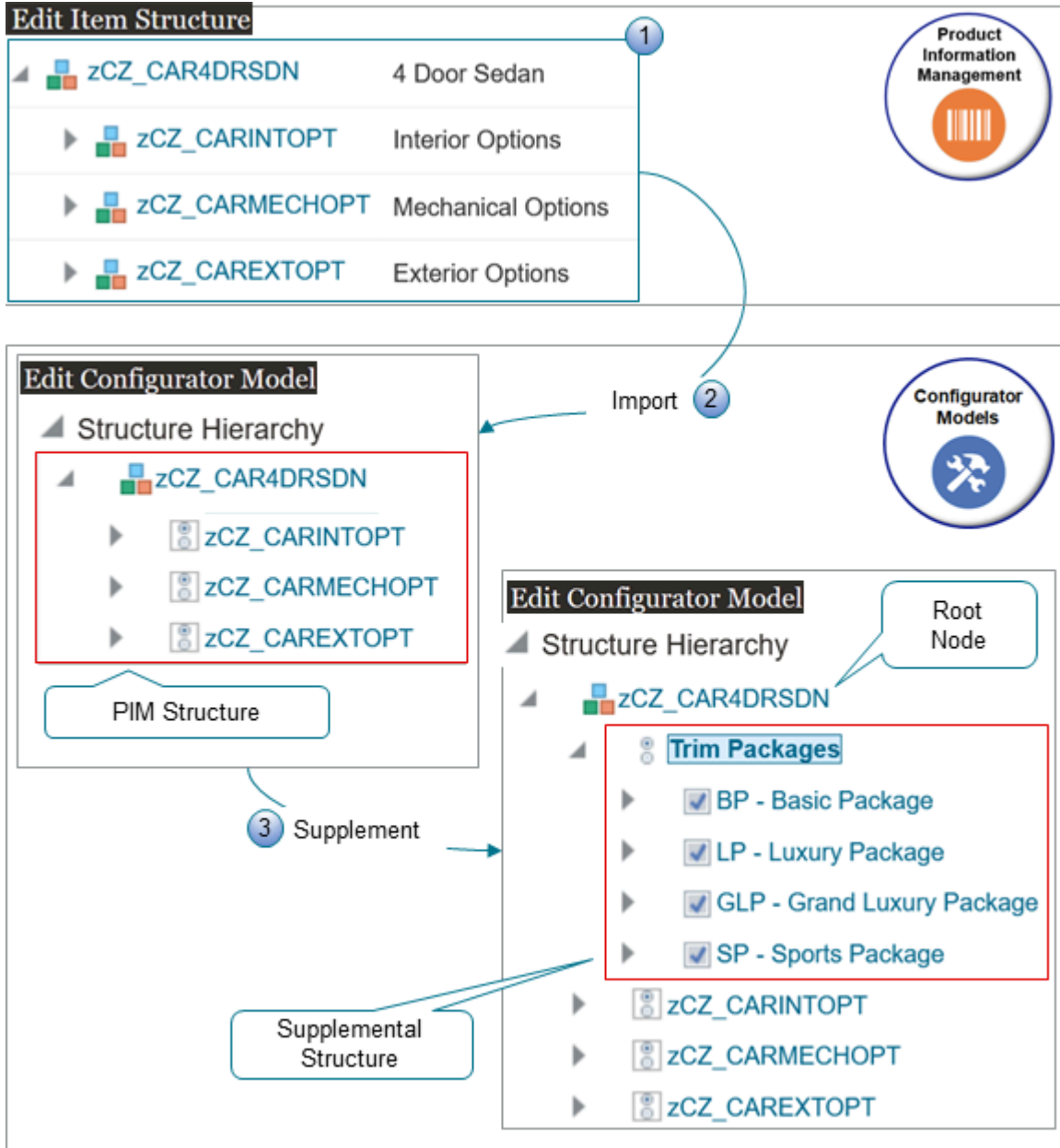
Supplemental Structure

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 Model Rules](#).

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 - Sentinel Custom Desktop, and Order Management is the host application.

The screenshot displays the Oracle Order Management interface. At the top, there is a navigation bar with the Oracle logo and various icons. Below this, the main header reads "Order Management". The central content area is titled "Configure: CN92777 – Sentinel Custom Desktop" and "Acme Bikes, Inc – 123456789 (Draft)". It includes buttons for "Finish and Review", "Finish", and "Cancel". The configuration options are grouped into sections: "Memory" (32GB Dual Channel DDR [79.99 Each]), "Casing" (790 Minitower with Standard Power Supply, 890 Desktop with Standard Power Supply, 990 Ultra Tower for ENRG Power Supply), "Processor" (Intel Core i7 2600 (3.4 Ghz, 8M), Intel Core i5 2400 (3.1 Ghz, 6M), Intel Core i3 2200 (2.8 Ghz, 4M), Dual Core Intel Xeon W3503 (2.4GHz, 4M L3), Dual Core Intel Xeon W4503 (3.4GHz, 8M L3)), and "Operating System" (Genuine Windows 8 Pro, 32-bit; Genuine Windows 8 Pro, 64-bit; Genuine Windows 8 Ultimate, 32-bit; Genuine Windows 8 Ultimate, 64-bit). A callout box labeled "Runtime user interface of a Configurator Model" points to the entire configuration area. Another callout labeled "Host application" points to the "Order Management" header. A callout labeled "Control template" points to the "Casing" section. A callout labeled "Layout template" points to the "Processor" section. A callout labeled "Shell template" points to the bottom of the configuration area.

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.	<ul style="list-style-type: none"> • Models
Product Configurator Manager	<p>Bring product data from the Product Information Management work area into the Configurator Models work area.</p> <p>Prepare the model so you can implement the configuration behavior that you need.</p>	Use a workspace so you can experiment with and test different ways to model your item.	<ul style="list-style-type: none"> • Snapshots • Workspaces • Draft Models
<ul style="list-style-type: none"> • Model Designer • Rules Programmer • UI Designer 	Add business logic to the selections that your user makes for the configuration.	<ul style="list-style-type: none"> • 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. 	<ul style="list-style-type: none"> • Supplemental structure • Configurator rules • User interfaces
<ul style="list-style-type: none"> • 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.	<ul style="list-style-type: none"> • Workspaces • Release process

Who You Are	What You Do	Why You Do This	How You Do It
<ul style="list-style-type: none"> Product Life Cycle Strategist 	Maintain the model.	Keep up with or get ahead of changes in your product line as it evolves.	<ul style="list-style-type: none"> 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](#).
- Go to [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](#)
 - [Use Transactional Attributes with Configurator Extensions](#)

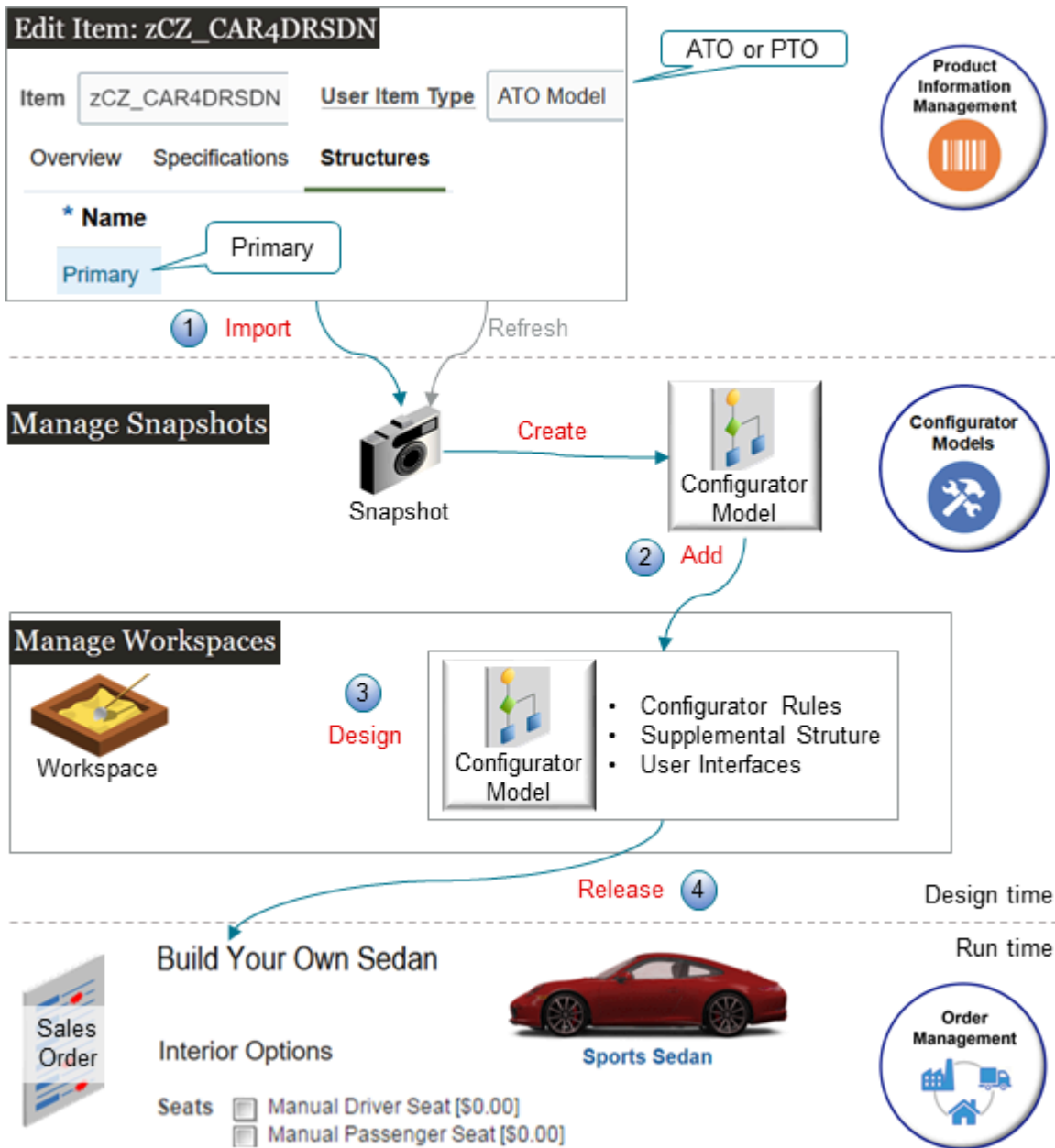
Related Topics

- [Snapshots](#)
- [Manage Your Workspace](#)
- [Overview of Supplementing Your Model](#)
- [Overview of Model Rules](#)
- [Overview of Model User Interfaces](#)
- [Overview of Oracle Pricing](#)

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.
 - o The released model includes the supplemental structure, rules, and user interfaces that you added to the model.
 - o The model goes into effect according to the start date that you specify on the workspace.
 - o Users in the host application interact with the model to configure the item. The Configurator makes sure the configuration that the user configures 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

- o 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.
- o 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

- o 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.
- o 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	<p>Set a date that happens in the future. You can't use a date that has already occurred.</p> <p>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.</p>

Note

- o Use the workspace to modify your configurator models, as necessary.
 - o You can add models to the workspace and edit them, as necessary.
 - o 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.

- 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.

- 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	<p>A set of options that model the question: "What kind of driving do you plan to do?"</p> <ul style="list-style-type: none"> Local trips Highway commuting Off-road camping <p>Note that this kind of question is an option feature, and the response that your user provides is an option.</p>
Rule	<p>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.</p>
User interface	<p>A user interface that displays the name and photo of each engine, suspension, or tire package that the rules select.</p>

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-reaching 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.	<ol style="list-style-type: none"> 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.	<ol style="list-style-type: none"> 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. <p>The Configurator Models work area will:</p> <ul style="list-style-type: none"> • 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](#)
- [How You 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.

Edit Item Structure: zCZ_CAR4DRSDN - Primary

Item	Item Description	Structure Item Type	
zCZ_CAR4DRSDN	4 Door Sedan		
zCZ_CARINTOPT	Interior Options	Option Class	
zCZ_CARMECHOPT	Mechanical Options	Option Class	
zCZ_CARENGINE	Engine	Model	

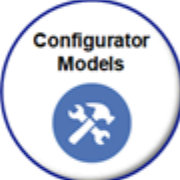
Manage Snapshots


Actions

- Refresh All other times
- Refresh Including Descendant Structure
- Refresh Including Descendant and Referenced Structure

- Import Items Associated with an Item Class
- Import Model Item First time

Import or refresh





Snapshot

Name	Description	Status	Snapshot Type	Structure Item Type	
zCZ_CAR4DRSDN	4 Door Sedan	Released	Item	Model	
			Last Refresh Date	Last Release Date	
			12/8/15 9:45 PM	12/8/15 11:07 PM	

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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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: <ul style="list-style-type: none"> • 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
<ul style="list-style-type: none"> • Description • Item Type • Primary UOM Code • Indivisible Flag • Organization • Serial Generation • Track in Installed Base 	<ul style="list-style-type: none"> • 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
	<ul style="list-style-type: none"> • 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, then 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	<p>Brings all the updates that you have made on the item in Product Information Management into the snapshot in the Configurator Models work area.</p> <p>The refresh will update:</p> <ul style="list-style-type: none"> ○ 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	<p>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.</p> <p>This type of refresh applies only for an item's snapshot. It doesn't refresh child models that the parent references.</p>
Refresh Including Descendant Structure and Referenced Structure	<p>Same as for Refresh, but the refresh will also include all structures that are children of a referenced structure, such as a referenced model.</p>

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.	<p>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.</p> <ul style="list-style-type: none">• 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. <p>If the snapshot isn't up to date, then go to the Workspace page, then click Actions > Add Updated Item Snapshots for Models.</p>

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.

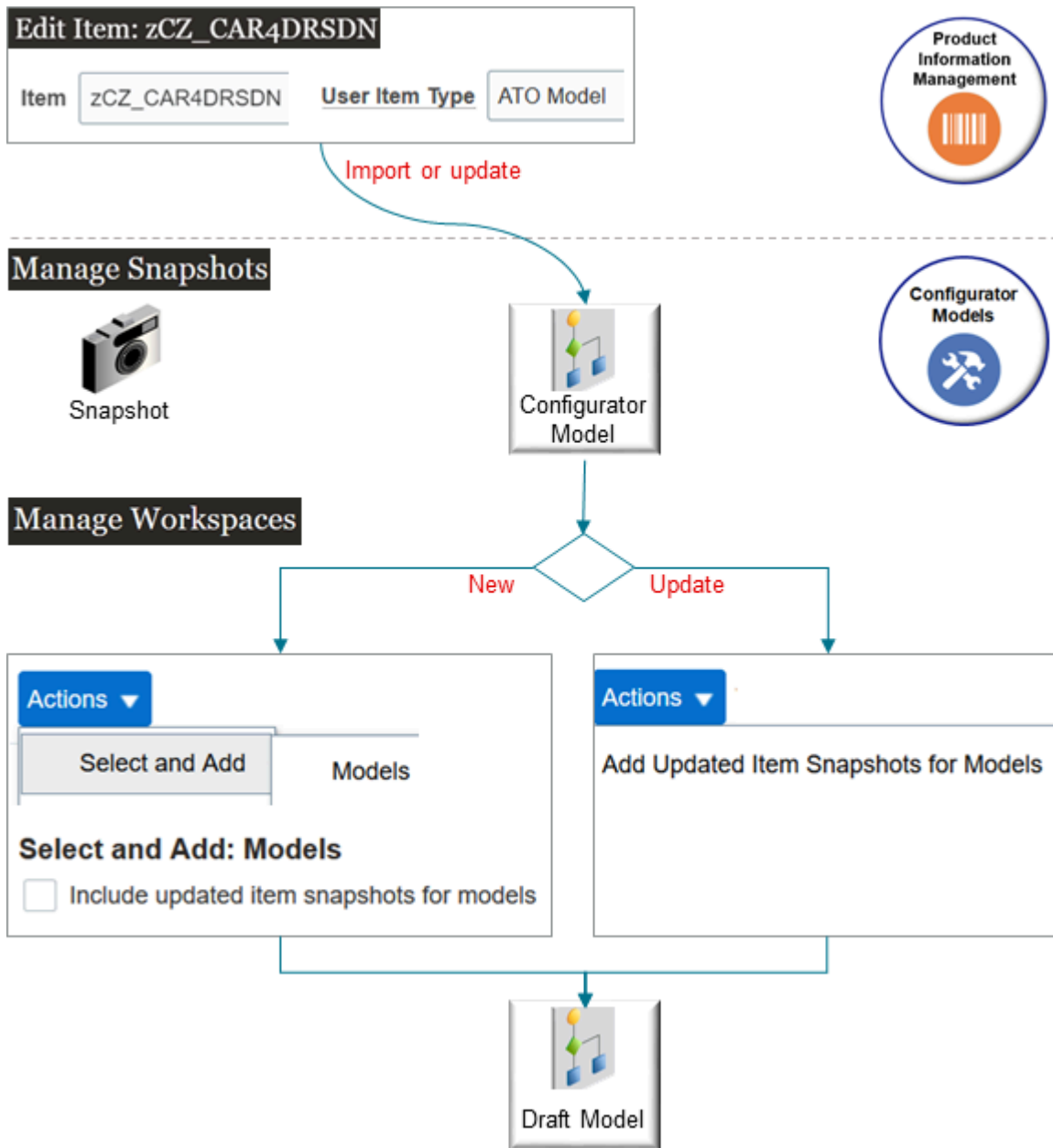
Create Your Workspace and Add Participants

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

Type of Participant	Description
Model	<p>You can add a configurator model from a list of the items that you already imported into a snapshot.</p> <ul style="list-style-type: none"> 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	<p>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.</p>
Item Class Snapshot	<p>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.</p>
Value Set Snapshot	<p>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.</p>

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.

- o Click **Tasks > Manage Workspaces**.
- o 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 <i>Release Workspace</i> 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: <ul style="list-style-type: none"> ○ 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:

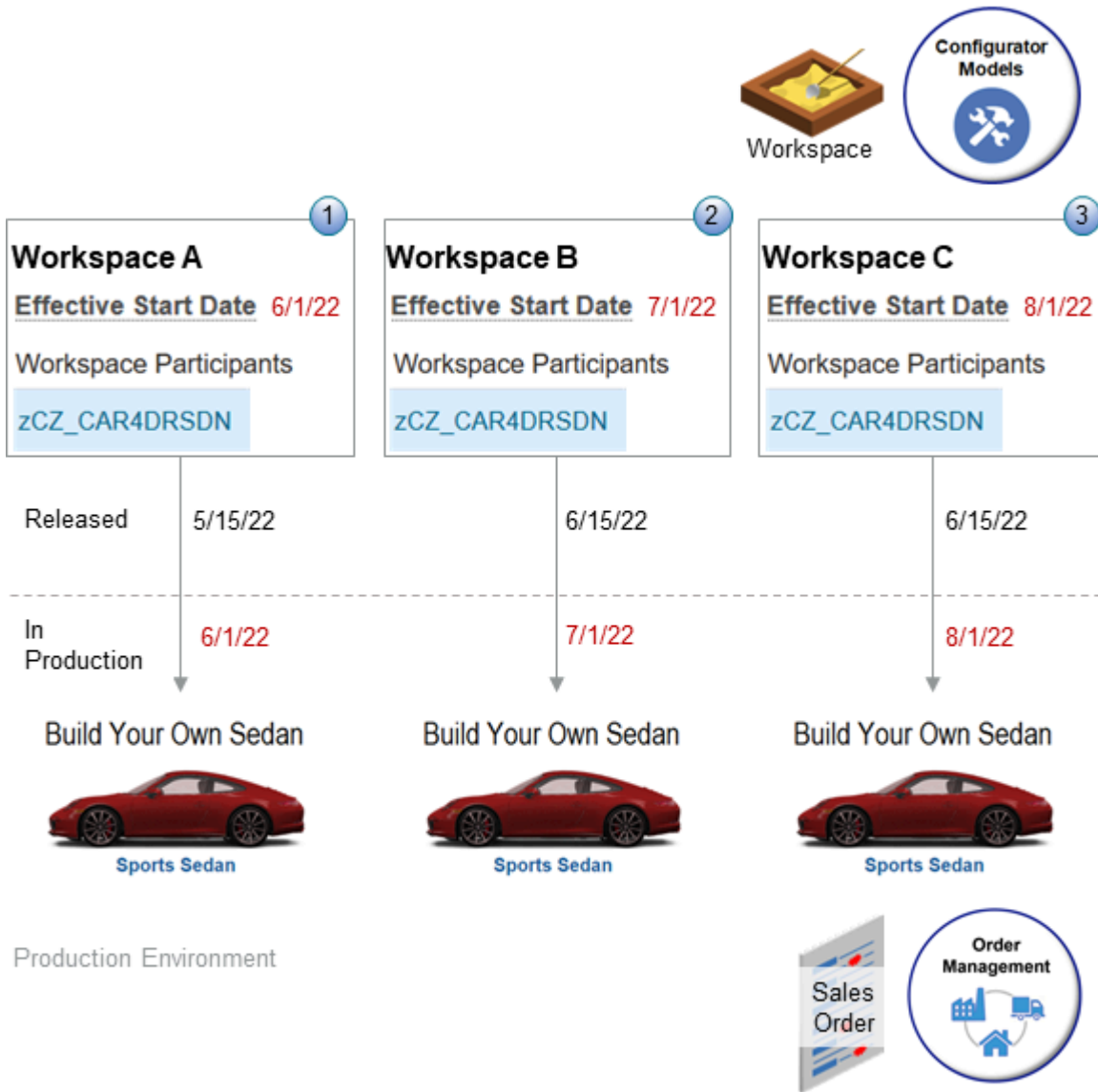
- **In production.** Configurator prevents the release. You must add the model to the workspace and fix the rules.
- **A draft.** Configurator does the release, and adds an entry in the log of the *Release Workspace* scheduled process that identifies the rule that you need to fix.

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.

- o Open the ORA_QP_PRICING_PARAMETER_VALUE.csv file.
- o 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.
- o 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.

- o 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.
- o 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**.
 - o On the Import Setup Data from CSV File page, click **Browse**, select **Manage Pricing Parameters_20210909_093224_303.zip**, then click **Submit**.
 - o 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**.
 - o 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.
- o Open a workspace that has an effective start date that happens more than two days after today.
 - o Click **Release**.
 - o 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.

Item	Item Description	Start Date
zCZ_CAR4DRSDN	4 Door Sedan	
zCZ_CARINTOPT	Interior Options	6/15/22
zCZ_CARMECHOPT	Mechanical Options	5/15/22

Product Information Management

Dates here take precedence

If this happens...

Keep dates synchronized

... before this...

... you get

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

Workspace: 4 Door Sedan

Effective Start Date 6/1/22

Workspace Attributes

Workspace Participants

Name	Description
zCZ_CAR4DRSDN	4 Door Sedan

Workspace

Configurator Models

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](#)

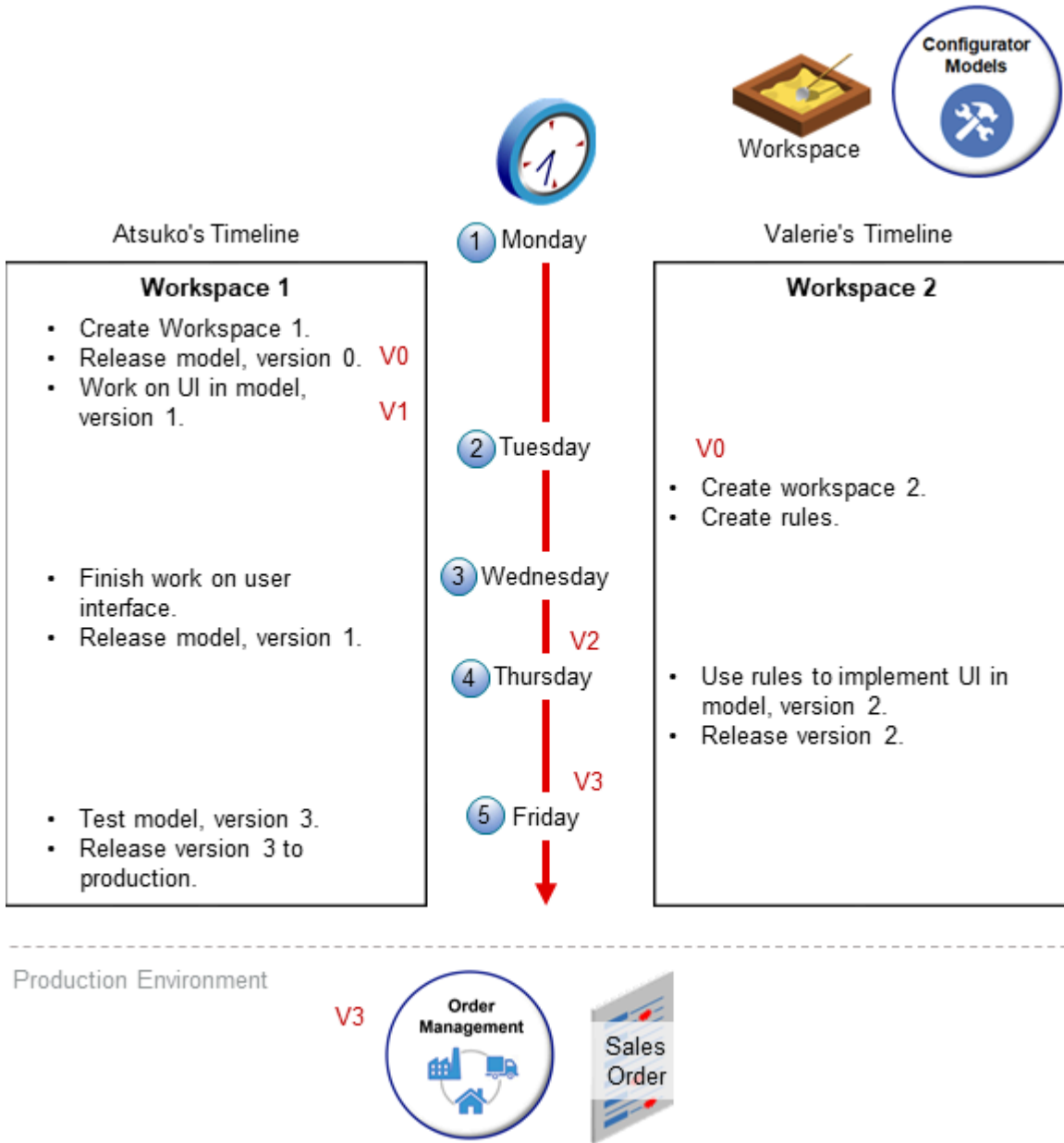
Work on the Same Participant in Different Workspaces

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.

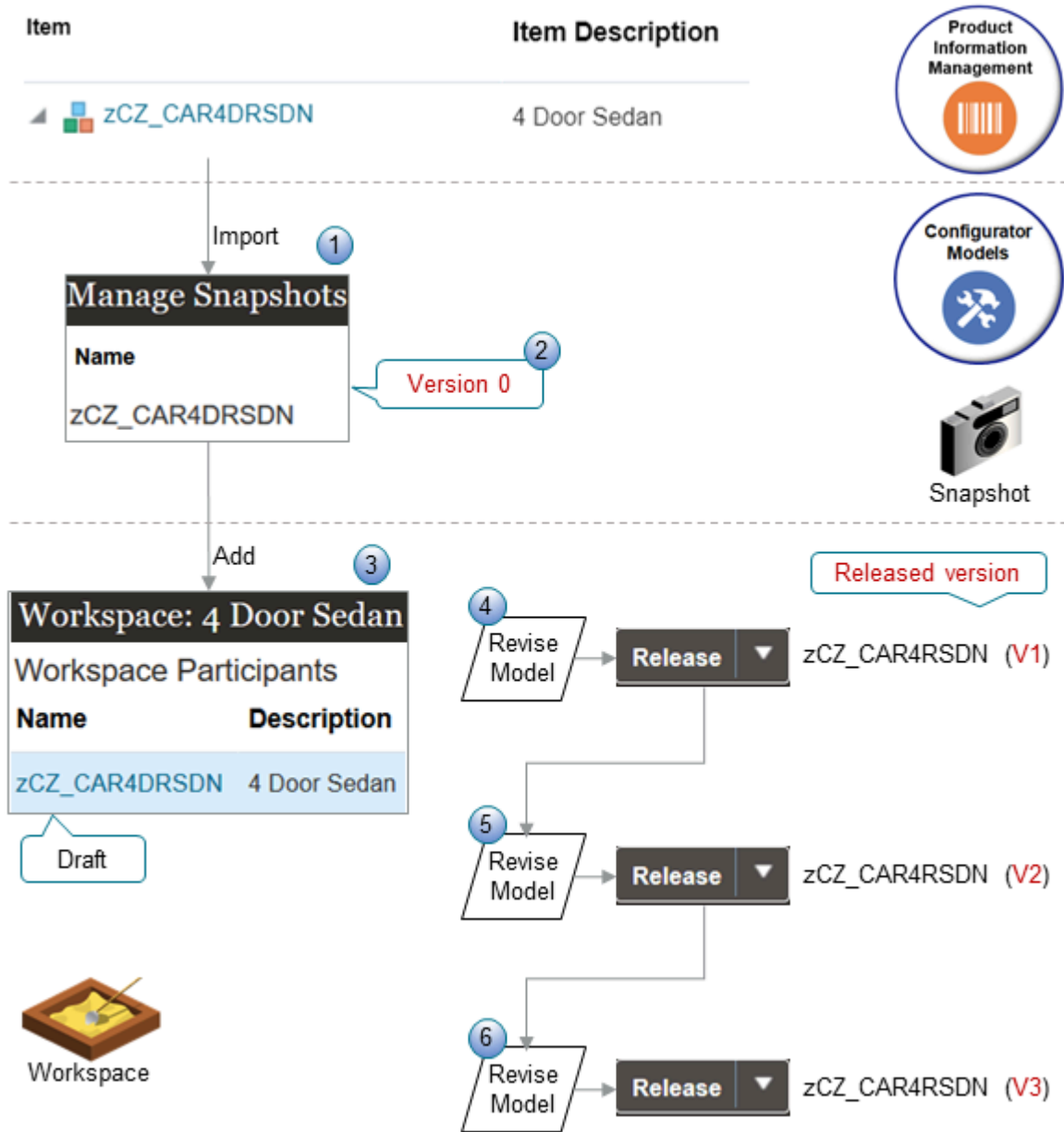
Assume you add the 4 Door Sedan model as a participant x to Atsuko's workspace. If you add participant x to Valerie's workspace, then Configurator displays a message that tells you another workspace already has participant x. If Atsuko's workspace has the same effective start date as Valerie's workspace, 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 occurs 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. 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

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

- 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.

The screenshot displays the 'Edit Configurator Model: zCZ_CAR4DRSDN (Draft)' interface. At the top, there are tabs for 'Structure', 'Rules', and 'User Interfaces'. A 'Configurator Models' icon is visible in the top right. The main area shows the 'Structure Hierarchy' with a tree view. The root node is 'zCZ_CAR4DRSDN'. Under it, the 'Trim Packages' node is highlighted with a callout box labeled 'Option Feature'. Below 'Trim Packages', there are four sub-nodes: 'BP - Basic Package', 'LP - Luxury Package', 'GLP - Grand Luxury Package', and 'SP - Sports Package'. These four sub-nodes are grouped by a bracket and labeled 'Options'. Below these are three other nodes: 'zCZ_CARINTOPT', 'zCZ_CARMECHOPT', and 'zCZ_CAREXTOPT'. The 'Actions' menu is visible at the top left of the hierarchy view.

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**.

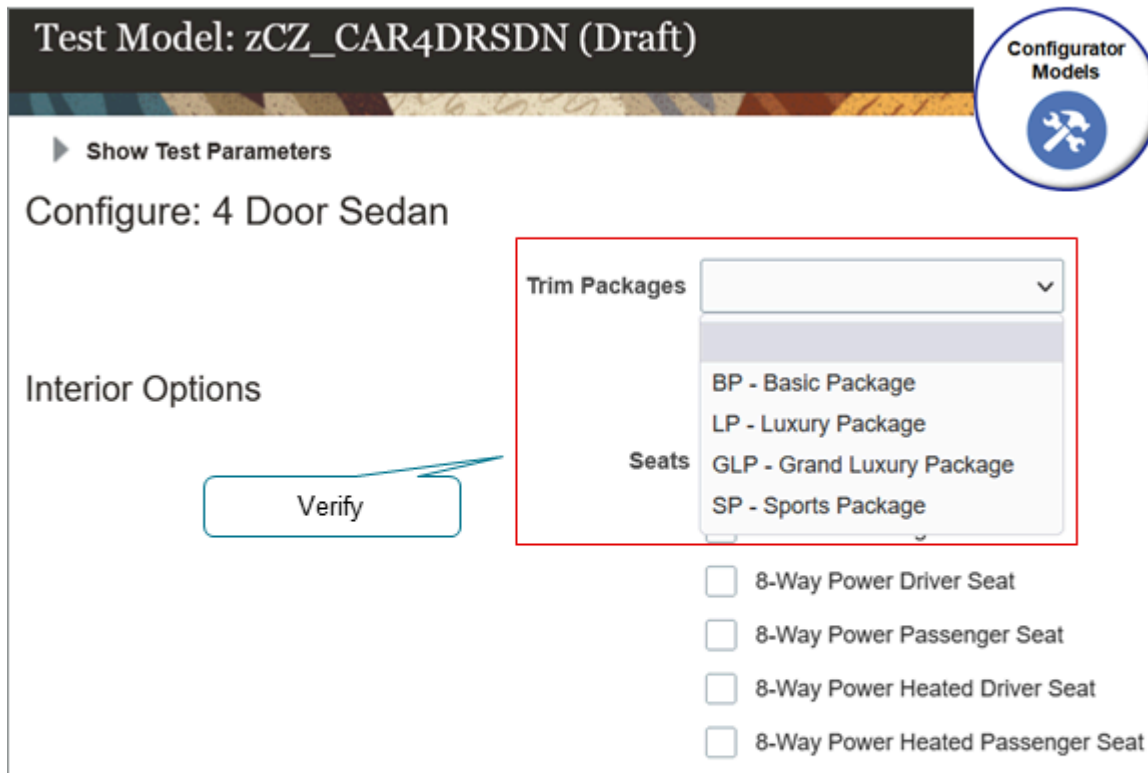
Attribute	Value
Name	BP - Basic Package

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.

- 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
	<p>Set Option Quantities to No.</p> <p>If you set Minimum Selections to 1, then the user must select at least one option.</p>
<p>You have an option feature named Car Keys, they come in red, green, or blue, but all keys must be red, green, or blue.</p> <p>You sell up to 10 copies of the key for each car.</p> <p>The user must select at least one key.</p>	<p>Create an option feature named Car Keys.</p> <p>Set Minimum Selections to 1.</p> <p>Set Maximum Selections to 1.</p> <p>Set Option Quantities to Yes.</p> <p>Set Maximum Quantity per Option to 10.</p> <p>Add these options to Car Keys: Red, Green, Blue.</p>
<p>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:</p> <ul style="list-style-type: none"> • 3 red keys and 3 blue keys • 2 red keys, 2 green keys, and 2 blue keys <p>You sell up to 10 copies of the key for each car.</p>	<p>Create an option feature named Car Keys.</p> <p>Set Minimum Selections to 1.</p> <p>Set Maximum Selections to 3.</p> <p>Set Option Quantities to Yes.</p> <p>Set Maximum Quantity per Option to 10.</p> <p>Add these options to Car Keys: Red, Green, Blue.</p>
<p>You sell a laptop computer. As an incentive, you allow the customer to choose from 0 to 3 free applications.</p>	<p>Create an option feature named Free Applications.</p> <p>Set Minimum Selections to 0.</p> <p>Set Maximum Selections to 3.</p> <p>Set Option Quantities to No.</p> <p>Add these options to Free Applications:</p> <ul style="list-style-type: none"> • 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.

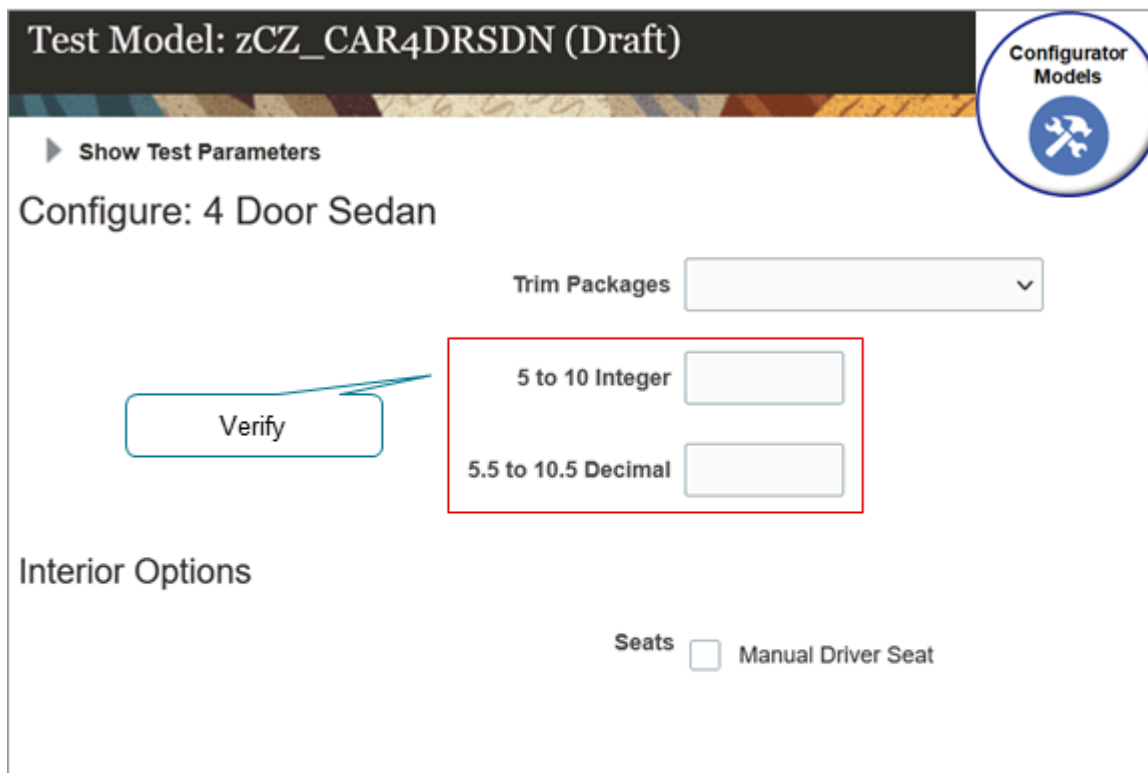
1. Open the workspace that has your model.
2. In the Structure Hierarchy area, click the zCZ_CAR4DRSDN **root node**.
3. Click **Actions > Create > Integer Feature**.
4. In the dialog that displays, set the values, then click **OK**.

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

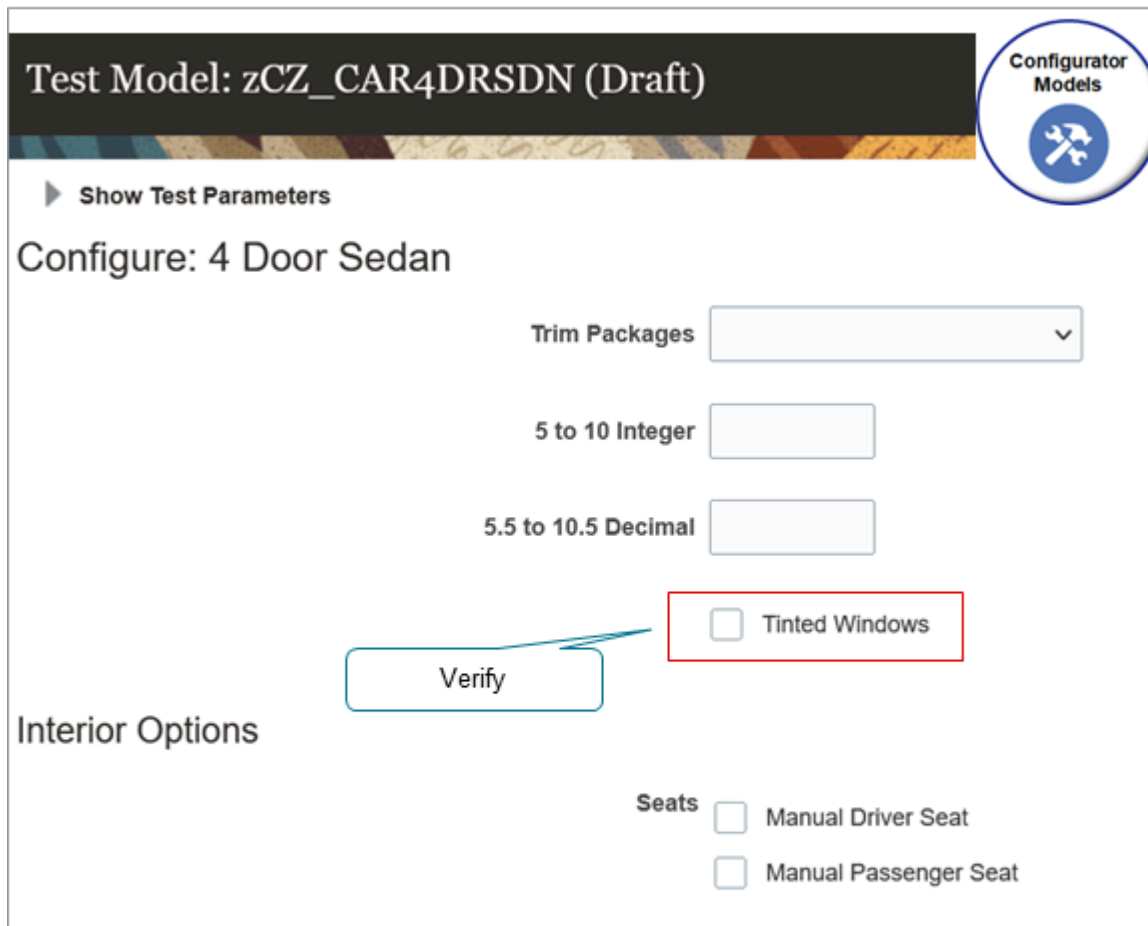
Assume you import a PIM model named zCZ_CAR4DRSDN into Configurator. You need to add a Boolean feature that allows your user to say whether they want tinted windows.

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	<p>Configurator sets the Boolean feature to False, by default.</p> <p>If another rule prevents Configurator from setting it to False, then Configurator will set it to True.</p> <p>For example, a <i>negates</i> rule for the Boolean feature might prevent Configurator from setting it to True or to False.</p>
Prefer True	<p>Configurator sets the Boolean feature to True, by default.</p> <p>If another rule prevents Configurator from setting it to True, then Configurator will set it to False.</p>

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	<p>6</p> <p>The maximum length is 32,000 characters.</p> <p>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.</p>

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	<p>Use a default rule to set the default value that Configurator displays for a text element.</p> <p>Configurator applies the default rule at runtime when the user manually selects an option and enters a value.</p> <p>Configurator can also apply a default rule when it starts the configuration at runtime.</p> <p>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.</p>
Constraint Rule	<p>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.</p> <p>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.</p>
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	<p>Select the attribute that you need to add.</p> <p>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.</p>
Value	<p>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.</p> <p>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.</p>
Description	<p>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.</p>
Value Set	
Data Type	

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	<p>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.</p> <p>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.</p>

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'. 'zCZCARMCHOPT'. '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.
- `'zCZCARMCHOPT'. '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:
 - o On the Edit Configurator Model page, in the Structure Hierarchy area, click the **root**.
 - o You can add a supplemental structure only on the root of an item structure.
 - o Click **Actions > Create**.
 - o 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

- o Notice that the Configurator adds a branch for your new feature to the tree.
 - o In the dialog that displays, enter your details.
 - o 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 occur 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 the user selects 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)
ChangedByAl	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
DetailedSele	Yes	Yes	Yes	Yes	Yes	No	No	Yes	Yes
DisplayName	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
DisplayName	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
InputRequired	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
InputRequired	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IsBound	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IsBoundQuantity	Yes	No	Yes	Yes	No	No	No	Yes	Yes
IsMinSelection	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
SelectedCount	Yes	Yes	No	No	No	No	No	No	No
SelectionState	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:

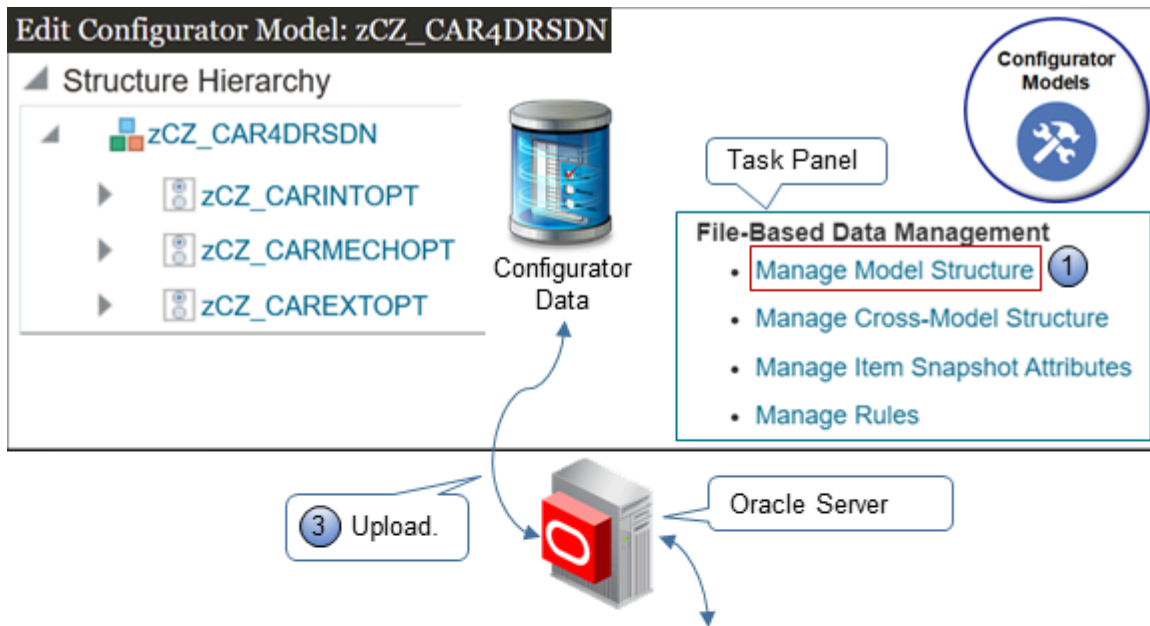
- **Enable the placeholder.** Configurator doesn't create a new page for the option class at runtime.
- **Don't enable the placeholder.** Configurator creates a new page for the option class at runtime.

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.



ManageModelNodes.xlsx - Excel

Layout Formulas Data Review View Manage Supplemental Structure Acrobat

Login Sign Out Clear All Data Options Edit About Search Create or Update Delete Status Viewer

ADFDI Command Ribbon (2)

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	<ul style="list-style-type: none"> • 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	<ul style="list-style-type: none"> • Manage the dates that determine when the attribute values are in effect for each snapshot.
Supplemental attribute	<ul style="list-style-type: none"> • 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.

Name	Description	Worksp	Status	Created By	Creation Date
zCZ_CAR4DRSDN	4 Door Sedan	Car Model	DEVELOPMENT	PROD_CONFIGURATOR	7/3/2018 7:29

Changed	Flagged	Status	*Name	Description	*Node Type
			zCZ_CAR4DRSDN	4 Door Sedan	Model
			zCZ_CARINTOPT	Interior Options	Option Class
			zCZ_CARMECHOPT	Mechanical Options	Option Class
			zCZ_CAREXTOPT	Exterior Options	Option Class
			Trim Packages	I added a description.	Option Feature
			BP - Basic Package		Option
			LP - Luxury Package		Option
			GLP - Grand Luxury Package		Option
			SP - Sports Package		Option

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 Application Development Framework Desktop Integration to Manage 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:
 - o Is in Draft status.
 - o 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.
 - o 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 Application Development Framework Desktop Integration to Manage 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 ADFDI command 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 (ADF DI) 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	<p>Displays the status of your action, such as Row Inserted Successfully, Update Failed, or Insert Failed.</p> <p>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.</p> <p>If an error happens, then the Status column displays a summary of the error. Use the Status Viewer to view error details.</p>
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
<p>You receive a message.</p> <p>Unable to execute the command Create or Update while a cell is in edit mode.</p>	<p>Step out of the cell you're editing, then retry the update.</p>
<p>You receive a message.</p> <p>Maximum: cannot convert the input value to the expected data type (BigDecimal).</p> <p>Some messages in the Status Viewer includes details.</p> <ul style="list-style-type: none"> • 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. 	<p>Make sure the value in the Maximum attribute contains a BigDecimal data type.</p>
<p>You receive a message.</p> <p>Required property UniqueAttribute missing.</p>	<p>Make sure each required attribute contains a value.</p>

Trouble	Shoot
<p>You receive a message.</p> <p>View row with key Oracle x is not found.</p>	<p>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.</p> <ol style="list-style-type: none">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. <p>To fix this problem, refresh your data after you navigate back to the Manage Model Structure worksheet, then make and upload your changes.</p>

5 Rules

Overview of Model Rules

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

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

You access model rules in the following ways.

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

Related Topics

- [Configurator Rules](#)
- [How You Create Statement Rules](#)
- [The Constraint Definition Language](#)

Rule Principles

Configurator Rules

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

Typical uses for configurator rules are to:

- Set default choices or values for model nodes
- Automatically select options based on another choice by end user
- Prevent users from selecting invalid combinations

- Control how many instances of a model can be created at run time
- Calculate and set the values for numeric options

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

- Variables have a defined domain and a run time range of values.
- Variables must be bound to a value.
- Variables can require input.
- Configurator can automatically try to solve the problem, using a feature called autocompletion.

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

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

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

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

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

- Constraint
- Default
- Search Decision

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

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

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

Related Topics

- [Rule Classes](#)
- [Logic Rules](#)
- [Accumulator Rules](#)
- [Compatibility Rules](#)

Rule Classes

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

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

Rule Class	Description
Constraint	<p>A rule with a class of Constraint must always be true.</p> <p>See the fuller description in the information about Constraints.</p>
Default	<p>A rule with a class of Default is like a less strict constraint. The constraint engine will try to make the rule true but if that's not possible the rule will be ignored.</p> <p>A Default rule gives you some control over the sequence of rule evaluation</p> <p>Some operators can't be used in Default rules.</p> <p>See the fuller description in the information about Defaults.</p>
Search Decision	<p>Like Default, but but applied during the process of finishing a configuration (the autocompletion process).</p> <p>See the fuller description in the information about Search Decisions.</p>

Choosing Rule Classes

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

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

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

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

Constraints

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

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

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

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

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

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

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

Defaults

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

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

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

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

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

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

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

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

```
Weight <= 3000
```


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

Range: 1000 to 3000

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

Rules classified as Defaults may be expressed as assignment functions.

Search Decisions

Rules that you classify as Search Decisions are applied:

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

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

Specifying a Sequence for Defaults and Search Decisions

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

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

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

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

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

Related Topics

- [Configurator Rules](#)
- [Logic Rules](#)
- [Accumulator Rules](#)
- [Compatibility Rules](#)

Logic Rules

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

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

- Requires
- Negates
- Implies
- Excludes





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

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

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

Requires





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

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

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

Negates

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

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

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

Implies

The following figure shows the effect of the Implies relation.

	A	IMPLIES	B	
	True	→	True	
	False		Unknown	
	Unknown		True	
	False	←	False	

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

Excludes

The following figure shows the effect of the Excludes relation.



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

Related Topics

- [Configurator Rules](#)
- [Rule Classes](#)
- [Accumulator Rules](#)
- [Compatibility Rules](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)

Logic States and Rule Variables

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

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

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

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

Related Topics

- [Configurator Rules](#)
- [Rule Classes](#)
- [Logic Rules](#)
- [Accumulator Rules](#)
- [Compatibility Rules](#)

How You Use Attributes in Model Rules

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

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

Attribute Type	Syntax	Examples
User-defined attributes	<p><nodePath>.userAttrs["<attrGroupName>"]</p> <p>If a model node has user-defined attributes, you can insert them in rule definition text from the Item tab of the Attributes pane of the rule editor. Select an attribute and click the Insert into Rule Text button.</p> <p>The values of user-defined attributes are static at run time, since the values are part of the model definition.</p>	<pre>'Home Theater System'. 'Speaker System'.userAttrs["PhysicalAttributes.Color"]</pre>
Supplemental attributes	<p><nodePath>.suppAttrs["<attrName>"]</p> <p>If a model node has supplemental attributes, you can insert them in rule definition text from the Supplemental tab of the Attributes pane of the rule editor. Select an attribute and click the Insert into Rule Text button.</p> <p>The values of supplemental attributes are static at run time, since the values are part of the model definition.</p>	<pre>'Home Theater System'. 'Speaker System'.suppAttrs["color"]</pre>
Transactional attributes	<p><nodePath>.transAttrs["<attrName>"]</p> <p>If a model node has a transactional attribute, you can insert them in rule definition text from the Structure pane of the rule editor. Select a transactional attribute from the structure tree and click the Insert into Rule Text button.</p> <p>The values of transactional attributes are dynamic at run time, since the values are</p>	<pre>ADD 'Custom Window'. 'Frame'.transAttrs["Linear Length"]/5 TO 'Custom Window'. 'Frame'. 'Track'.Quantity()</pre>

Attribute Type	Syntax	Examples
	<p>determined during a configuration session by user action or model rules.</p> <p>The example divides the linear length of a window by 5 (5 feet in length for material) and adds it to the number of standard tracks needed.</p>	
<p>Configurator system attributes, such as:</p> <ul style="list-style-type: none"> • Name • Value • Quantity • State • Options • SelectedCount 	<p><code><nodePath>.<attrName>()</code></p> <p>You can insert system attributes in rule definition text from the System tab of the Structure pane of the rule editor. Select an attribute and click the Insert into Rule Text button.</p>	<p><code>'Home Theater System'. 'Speaker System'. '5.1'. Quantity()</code></p>

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

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

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

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

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

Related Topics

- [Configurator Rules](#)
- [Rule Classes](#)
- [Logic Rules](#)
- [Accumulator Rules](#)
- [Compatibility Rules](#)

Accumulator Rules

Accumulator Rules are a rule type that enables end-user selections to add to or subtract from a value from a variable at run time.

For example, when the end user selects the 512 MB RAM option you want to add 512 to a Total called Total RAM Selected.

Background

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

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

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

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

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

Keep the following in mind when using Accumulator Rules:

- If the Model contains more than one accumulator rule that adds to or subtract from the same target node, and that node exists in a referenced Model, generating logic creates a single constraint that equates the target to a sum of all the terms expressed in the individual rules in that model.

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

- When a numeric rule compares two decimal values, they may be interpreted as not being equal even when they're, due to their representation by the Java double data type. The decimal tolerance value (also referred to as epsilon value) is set to 0.000000001 (scale of 9), which will then be used by the configurator engine to convert the decimal comparison to a mathematical equation for evaluation. For example, the expression `a=b` could be converted to `Abs(a-b) <= epsilon`. This would give an accurate result as long as the scale is less than or equal to the number of decimal digits specified in the epsilon (the decimal tolerance value, which is 0.000000001).

Procedure

To create an Accumulator Rule::

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

`ADD n TO x`

or

`SUBTRACT n FROM x`

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

Related Topics

- [Configurator Rules](#)
- [Rule Classes](#)
- [Logic Rules](#)
- [Compatibility Rules](#)
- [CDL Accumulator Statements and the ADD or SUBTRACT Keywords](#)

Compatibility Rules

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

Background

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

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

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

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

Example

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

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

Related Topics

- [Configurator Rules](#)
- [Rule Classes](#)
- [Logic Rules](#)
- [Accumulator Rules](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)

Consider 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 the user sets 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
```

Here are some other alternatives:

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

Change the Sequence of Your Operands

If the solutions that we describe above don't work for you, then you can change 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.

Create Statement Rules

How You Create Statement Rules

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

Creating a Statement Rule

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

The following table presents the actions required to create statement rules, and the reason for each action.

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

Testing a Configurator Rule

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

To test a configurator rule:

1. Click the **Test Model** button. You don't have to be on the Rules tab of the model to test rules.

2. In the Test Model dialog box, ensure that you select a user interface that includes nodes of the model that are affected by the rule behavior that you intend to test, in the **User Interface** field. By default, the previously tested UI is selected.
3. Make selections among the configuration options, emulating both the likely and also the possible choices made by an end user. Navigate through the pages of the UI, and observe how your own selections affect other selections that are made by the rule you're defining.

Related Topics

- [The Statement Rule Editor](#)
- [Configurator Rules](#)
- [The Constraint Definition Language](#)

The Statement Rule Editor

You use the statement rule editor to create statement rules.

Features of the Statement Rule Editor

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

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

The Rules tab also provides

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

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

The Actions menu contains actions that let you reorder Default and Search Decision rules.

- The **Structure** pane, which enables you to insert syntactically correct references to a node directly into the rule text, by selecting the node and clicking the **Insert into Rule Text** button.

The following table summarizes the features and benefits of the rule editor.

Feature	Benefit
Enabled check box	You can disable a rule while you work on its definition, or if you don't want its behavior affecting the model.
Status	The status values are: <ul style="list-style-type: none"> • Modified: The rule text was edited, or some attribute of the rule was changed since the last time it was compiled. • Error: An error prevented the rule from being compiled and executed, or the rule has a syntax error. • Valid: The rule is valid, and can be compiled and tested.

Feature	Benefit
End Date	The read-only End Date field displays the date when the rule becomes invalid because one of the participants in the rule was end-dated.
Definition region	This region of the editor page groups the controls for modifying the rule.
Rule Class selection	Enables you to change the rule class that you selected when creating the rule. If the class is a Default or Search Decision, a Reorder button is enabled so that you can change the sequence in which your defaults or search decisions are executed.
Validate button	Checks the validity of the syntax of your rule text.
Text pane	You enter the CDL rule text of your statement rule here. Basic text editing is provided. The toolbar above the text pane provides insertion menus and text buttons that enable you to enter valid CDL code without using the keyboard.
Insertion menus	A set of menus enables you to insert valid CDL syntactical elements directly into the rule text. The menus are: <ul style="list-style-type: none"> • Keywords: Major CDL keywords around which statements are built. • Logic Operators: The operators for defining logic rules. • Functions: a set of cascading menus for entering functions that are grouped by categories: <ul style="list-style-type: none"> ○ Logic ○ Arithmetic ○ Trigonometric ○ Set ○ String <p>The menus are tear-off style, so you can tear an often-used menu off the toolbar and leave it positioned over the editor page for convenient access while you work.</p>
Text buttons	A set of buttons enables you to insert valid CDL syntactical elements directly into the rule text by clicking. The groups of buttons provide: <ul style="list-style-type: none"> • Characters for delimiting statements • Comparison operators • Boolean operators • Mathematical operators

Related Topics

- [How You Create Statement Rules](#)
- [Configurator Rules](#)
- [The Constraint Definition Language](#)

Use Application Development Framework Desktop Integration to Manage Rules

You can use ADF Desktop Integration with Microsoft Excel (ADFdi) to maintain a common set of rules across multiple models and environments.

Download rule definition data to a spreadsheet connected to your development environment, optionally modifying the definitions, and uploading the definitions to the same or another model or environment.

Rule Management Summary

You can perform the following operations when managing configurator rules:

Search	To 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	To create a rule for the model you searched for, download its rule data to an using an Application Development Framework Desktop Integration (ADFDI) enabled spreadsheet, add details for the rule, then upload the rule data.
Update	To update an existing rule for the model you searched for, download its rule data to an ADFDI-enabled spreadsheet, make changes to certain details for the rule, then upload the rule data.

Management Procedure

The following procedure applies to both updating or supplementing existing rules, and to replicating rules in another environment.

Before using this feature, you must have downloaded and installed the latest version of ADF Desktop Integration with Microsoft Excel, by selecting **Download Desktop Integration** from your Home page.

The models containing the rules to be managed must be in Draft status and must be participants in an unreleased workspace (a workspace with the status of **In Development**).

To create or modify rules using ADFDI:

1. From the File-Based Data Management section of the task drawer panel, click the Manage Rules task.
2. When prompted, open the Excel spreadsheet.
3. When prompted, sign in with the credentials that you use for the Configurator Models work area. You may need to sign in each time you make the ADFDI connection.

The spreadsheet for managing rules opens automatically in Microsoft Excel. The spreadsheet includes controls for working with ADFDI functions, on a worksheet ribbon tab named Manage Rules.

4. Click the **Search** button on the ADFDI tab. Enter selection criteria for the models whose rules you want to manage. You can restrict the search by model or workspace name, or both.

When your search completes, select the model whose rules you want to manage and click **OK**. The spreadsheet becomes populated with a row for each rule currently existing in the model.

Optionally, you can log out of your ADFDI connection, to work offline, by clicking a button on the ADFDI tab.

5. If you want to modify the downloaded rules, you can edit existing values for the rules, or insert new rows to create new rules or cell values that accomplish your update, using the columns of the spreadsheet as described in the following table. The columns whose values can be updated correspond to fields on the Rules tab of the Edit Configurator Model page.

Don't replicate data in an existing row by copying and pasting the entire row, because the generated read-only value in the Key column must be unique. Only replicate cells that can be updated, which are displayed without shading.

Column	Can Be Updated?	Description
Changed	No	Indicates that a row was changed or entered since the last download or upload.
Flagged	No	Reserved for future use.
Status	No	Displays the results of the validation of the row data that's performed on each upload. Values include: <ul style="list-style-type: none"> ○ Update failed ○ Insert failed ○ Row inserted successfully
Model	Yes	Name of the existing model containing the rule. You can't create a new model this way.
Workspace	Yes	Name of the existing workspace containing the model draft that contains the rule. You can't create a new workspace this way.
Name	Yes	Name of the rule. Entering a new name creates a new rule in the specified model, workspace, and folder. You can't change the name of an existing rule this way. You can't use a rule name that already exists in another draft of the model.

Column	Can Be Updated?	Description
Description	Yes	Description of the rule. You can change the description as desired.
Folder Path	Yes	Name and path of the folder containing the rule. Entering a new folder name creates a new folder under the top-level Rules folder for the model. To create or use subfolders, use the following notation: subfolder/ sub-subfolder/ . . .
Type	Yes	Type of the rule. Only Statement Rules are supported.
Rule Class	Yes	The current rule class of the rule. To change, the class, select one of: <ul style="list-style-type: none"> ○ Constraint ○ Default ○ Search Decision
Rule Text	Yes	The CDL text that defines the rule. The rule text is validated when you upload it, following all the rules that apply to rules in the Configurator Models work area. It's possible to upload invalid rule text, but it will cause the status of the uploaded rule to be Error . Ensure that node names are enclosed with single quotation marks, so that nodes can be located correctly when the rule text is parsed. If a rule begins with a node name, add an additional single quotation mark before it, to accommodate the behavior of Microsoft Excel.
Enabled	Yes	Whether the rule is enabled or not. Select Yes or No.
Rule Status	No	Displays the results of the validation of the rule text that's performed on each upload. Values are:

Column	Can Be Updated?	Description
		<ul style="list-style-type: none"> ○ Modified ○ Valid ○ Error
End Date	No	The date this rule becomes invalid due to end dating of participants.
Error Message	No	Error message generated when validating the rule during upload. The messages are the same as those generated by validation in the Configurator Models work area.
Key	No	Unique key value used by ADFDI to associate spreadsheet rows with rules.

A marker appears in the **Changed** column of the spreadsheet in each row that you modify. This marker indicates which rows will be uploaded to the model.

Tip: You can maintain a separate offline spreadsheet containing the data to be used for updates, then copy and paste that data into the ADFDI connected spreadsheets that you download.

6. Click the ADFDI **Create or Update** button, which submits the update using your current ADFDI connection.

The upload process updates the rule definitions based on the changes you made in the downloaded ADFDI spreadsheet. After every upload, each row in the spreadsheet is subjected to two-step validation:

- a. The data in the spreadsheet row is validated by ADFDI. The results are displayed in the **Status** column.
- b. The rule text is validated by Configurator. The results are displayed in the **Rule Status** column.

7. Examine the messages in the **Status** and **Error Message** columns of the spreadsheet for information on the update, or possible errors.

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, and examine click the **Status Viewer** button to examine details about the failure.

To see the effect of your rule updates, you might have to close the affected Rules page, if it's open, and reopen it to refresh the display.

Migrating Rules

You can migrate rules from one instance of the Configurator Models work area to another instance, in a way similar to creating or updating rules in a single instance.

For example, assume that you want to migrate rules from a Test instance to a Production instance. To migrate rules:

1. Sign into Test, search for the model containing the rules, and download the rules from that model.

2. Copy the downloaded rule data from the spreadsheet for Test. Only copy the cells that allow updating. You can modify the rules, as long as you follow the restrictions described here about which data can be changed.
Tip: You can set aside a backup of the copied rule data by pasting it in a separate spreadsheet that's not connected by ADFDI.
3. Sign into Production, search for the model that will receive the rules, and download the rules from that model. The workspaces and models referenced in your copied rule definitions must exist in Production.
4. Copy the rule data that came from Test, and paste it into the spreadsheet connected to Production, in the columns that are eligible for updating.
5. Click the ADFDI **Create or Update** button, to upload the rule data to the model in Production.

Related Topics

- [Configurator Rules](#)
- [How You Create Statement Rules](#)
- [How You Write Extension Rule Text](#)
- [Overview of Using Spreadsheets to Manage Supplemental Structures](#)

CDL Reference

The Constraint Definition Language

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

Overview of the Constraint Definition Language (CDL)

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

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

Relationships Expressed in CDL

Using CDL, you can define the following relationships:

- Logical
- Numeric
- Compatibility
- Comparison

Related Topics

- [Anatomy of a Configurator Rule Written in CDL](#)
- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)
- [CDL Functions](#)

Anatomy of a Configurator Rule Written in CDL

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

This section contains the following:

- Rule Definition
- Rule Statements
- Data Types

Rule Definition

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

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

Rule Statements

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

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

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

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

An expression is the part of a statement that contains an operator and the operands involved in a rule operation. An operator is a predefined keyword, function, or character that involves the operands in logical, functional, or mathematical operations. REQUIRES and the plus sign (+) are examples of operators. An operand can be an expression, a literal, or an identifier. The literal or identifier operand can be present in the rule as a **singleton** or as a **collection**. Model nodes that are referred to in a rule are called rule participants.

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

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

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

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

Data Types

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

- INTEGER
- DECIMAL
- BOOLEAN
- TEXT
- Node types

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

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

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

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

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

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

Related Topics

- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)
- [CDL Functions](#)
- [CDL Operators](#)

CDL Statements

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

The two kinds of statements are:

- Explicit statements
- Iterator statements

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

Explicit Statements

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

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

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

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

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

Iterator Statements

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

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

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

- FOR ALL...IN
- WHERE

Multiple Iterators in One Statement

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

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

Example: Multiple Iterators in One CONSTRAIN Statement

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

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

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

Multiple Iterators in One ADD...TO Statement

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

Related Topics

- [The Constraint Definition Language](#)
- [Anatomy of a Configurator Rule Written in CDL](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)

CDL Syntax Details

Miscellaneous syntactical elements of CDL are described here.

CDL Syntactical Elements

The following table lists miscellaneous syntactical elements of CDL.

Element	Description
Comments	Comments are included in rule definitions at your discretion to explain the rule. Comments aren't tokens and therefore ignored by the Configurator parser.
White space	White space, which includes spaces, line feeds, and carriage returns, format the input for better readability. The white space category of elements aren't tokens and therefore ignored by the Configurator parser.
Case Sensitivity	Keywords aren't case sensitive. Keyword operators aren't case sensitive. Model object identifiers are case sensitive. Formal parameters are case sensitive and can't be in quotes. The constants E and PI as well as the scientific E aren't case sensitive. The keywords TRUE and FALSE aren't case sensitive. Text literals are case sensitive. All keywords, constant literals, and so on aren't case sensitive.
Quotation Marks	Model structure node names that contain white space, or text that would be interpreted by the parser as keywords or operators, must be enclosed in single quotation marks.

Syntax Notation

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

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

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

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
...	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example or relevant to the discussion have been omitted.
boldface text	Boldface type in text indicates a new term, a term defined in the glossary, specific keys, and labels of user interface objects. Boldface type also indicates a menu, command, or option, especially within procedures
[]	Brackets enclose optional clauses from which you can select one or none.
>	The left bracket alone represents the MS DOS prompt.
\$	The dollar sign represents the DIGITAL Command Language prompt in Windows and the Bourne shell prompt in Digital UNIX.
%	The per cent sign alone represents the UNIX prompt.
name()	In text other than code examples, the names of programming language methods and functions are shown with trailing parentheses. The parentheses are always shown as empty. For the actual argument or parameter list, see the reference documentation. This convention isn't used in code examples.

Terminology

The following table defines the terms used here.

Term	Description
Cartesian product	A set of tuples that's constructed from two or more given sets and comprises all permutations of single elements from each set such that the first element of the tuple is from the first set and the second is from the second set, and so on.
clause	A segment of a rule statement consisting of a keyword and expression.
collection	A set of multiple operands within parentheses and separated by commas.
compiler	The part of Configurator that first parses rule definitions and then generates code that's executable at run time.
explicit statement	Explicit statements express relations among explicitly identified participants and restrict execution of the rule to those participants and the Model containing those participants.
expression	A subset of the statement that contains operators and operands

Term	Description
formal identifier	A variable that's defined in the scope of an iterator statement to represent an iterating identifier.
iterator statement	Iterators are query-like statements that iterate, or repeat, over one or multiple relations or constraints.
non-terminal	The kind of symbols used in the notation for presenting CDL grammar that represent the names of grammar rules.
parser	A component of the Configurator compiler that analyzes the syntactic and semantic correctness of statements used in rule definitions.
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 definition can consist of multiple statements, each consisting of clauses containing expressions, and separated by semi-colons.
terminal	The kind of symbols used in the notation for presenting CDL grammar that represent the names, characters, or literal strings of tokens.
token	The result of translating characters into recognizable lexical meaning. All text strings in the input stream to the parser, except white space characters and comments, are tokens.
unicode	A 16-bit character encoding scheme allowing characters from Western European, Eastern European, Cyrillic, Greek, Arabic, Hebrew, Chinese, Japanese, Korean, Thai, Urdu, Hindi and all other major world languages, to be encoded in a single character set.

Separators

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

The following table lists the separators that are valid in CDL.

Separator	Description
(The open parenthesis indicates the beginning of function arguments or the beginning of an expression.
)	The close parenthesis indicates the end of function arguments or the end of an expression.

Separator	Description
,	The comma separates arguments or collection elements.
;	The semi-colon separates statements.
.	The dot character separates identifiers in compound references.

Related Topics

- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)
- [CDL Functions](#)
- [CDL Operators](#)

CDL Expressions

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

Examples

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

```
2 * frame.border
```

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

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

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

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

Following are some additional examples of expressions.

The following expressions result in a BOOLEAN value:

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

The following expressions result in a INTEGER or DECIMAL value:

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

Related Topics

- [Anatomy of a Configurator Rule Written in CDL](#)
- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Functions](#)
- [CDL Operators](#)

CDL Functions

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

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

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

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

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

CDL supports the following functions:

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

Arithmetic Functions

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

Function	Description
Abs(x)	Takes a single number as an argument and returns the positive value (0 to +infinity). The domain range is -infinity to +infinity. Returns the positive value of x. Abs(-12345.6) results in 12345.6

Function	Description
AggregateSum(x)	Can be used in a Constraint, Default, or Search Decision, but only as a sub-expression.
Round(x)	Takes a single decimal number as an argument and returns the nearest integer. If the A side of a numeric rule is a decimal number, contributing to an imported bill of materials that accepts decimal quantities, then the Round(x) function is unavailable. The reason that the Round(x) function is unavailable is that the contributed value doesn't need to be rounded as the B side accepts decimal quantities. This function is available when the bill of materials item accepts only integer values.
RoundDownToNearest(x,y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded to the nearest smaller multiple of the second argument. For example, RoundDownToNearest(433,75) returns 375.
RoundToNearest(x,y)	This is a binary function. x is a number between -infinity and +infinity, y is a number greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. RoundToNearest(433,10) returns 430.
RoundUpToNearest(x,y)	This is a binary function. The number x is between -infinity and +infinity, and the number y is greater than 0 and less than +infinity. A number is returned between -infinity and +infinity. The first argument is rounded up to the nearest multiple of the second argument. For example, RoundUpToNearest(34.1,0.125) returns 34.125.
Ceiling(x)	Takes a single decimal number as an argument and returns the next higher integer. For example, ceiling(4.3) returns 5, and ceiling(-4.3) returns -4.
Floor(x)	Takes a single decimal number as an argument and returns the next lower integer. For example, floor(4.3) returns 4, and floor(-4.3) returns -5.
Log(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the logarithmic value of x. An error happens if x=0.
Log10(x)	Takes a single number greater than 0 and less than +infinity and returns a number between -infinity and +infinity. Returns the base 10 logarithm of x. An error happens if x=0.
Min(x,y,z,...)	Returns the smallest of its numeric arguments.
Max(x,y,z,...)	Returns the largest of its numeric arguments.
Mod(x,y)	This is a binary function. Returns the remainder of x/y where x and y are numbers between -infinity and +infinity. If y is 0, then division by 0 is treated as an error. If x=y, then the result is 0. For example, Mod(7,5) returns 2.
Exp(x)	Returns e raised to the x power. Takes a single number between -infinity and +infinity and returns a value between 0 and +infinity.
Pow(x,y)	This is a binary function. Returns the result of x raised to the power of y. The number x is between -infinity and +infinity. The integer y is between -infinity and +infinity and the returned result is between -infinity and +infinity. If y=0, then the result is 1. For example, Pow(6,2) returns 36.

Function	Description
Sqrt(x)	Sqrt(x) returns the square root of x. Takes a single number between 0 and +infinity and returns a value between 0 and +infinity. An input of -x results in an error.
Truncate(x)	Truncate(x) takes a single decimal number x and truncates it, removing all digits after the decimal point. For example, truncate(4.15678) returns 4.

Trigonometric Functions

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

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

Logical Functions

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

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

Text Functions

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

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

Text functions can only be used in static context; for example the WHERE clause of iterators.

Function	Description
Contains	Compares two operands of text literals and returns true if the first contains the second.
Matches	Compares two operands of text literals and returns true if they match.
BeginsWith	Compares two operands of text literals and returns true if the first begins with the characters of the second.
EndsWith	Compares two operands of text literals and returns true if the first ends with the character(s) of the second.
Equals	Compares two operands of text literals and returns true if the first equals the second.
NotEquals	Compares two operands of text literals and returns true if the first doesn't equal the second

Hierarchy or Compound Functions

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

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

Related Topics

- [Anatomy of a Configurator Rule Written in CDL](#)
- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)
- [CDL Operators](#)

CDL Operators

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

This section includes the following:

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

Operators Supported By CDL

The following table lists the predefined operators supported by CDL.

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

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

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

Operator Results

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

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

Mapping of Operators and Data Types

Operators	Data Type
Arithmetic	INTEGER DECIMAL
Logical	BOOLEAN

Operators	Data Type
Comparison	BOOLEAN

Operator Precedence

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

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

Precedence of Operators

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

LIKE and NOT LIKE Operators

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

LIKE Expression Resulting in a BOOLEAN Value

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

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

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

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

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

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

For a list of comparison operators, see [Operators Supported By CDL](#).

Text Concatenation Operator

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

COLLECT Operator

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

Related Topics

- [CDL Statements](#)
- [CDL Syntax Details](#)
- [CDL Expressions](#)
- [CDL Functions](#)
- [CDL Iterator Statements and the COLLECT Operator](#)

CDL Attribute Compatibility Rules

The COMPATIBLE...OF keyword supports compatibility statements.

COMPATIBLE...OF Keyword

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

Example

In the following example, the rule iterates over all the items of the Tint option class in the Glass child model and over all the items of the Color option class in the Frame child model of a Window model. A color and tint are compatible

whenever the Stain user-defined attribute in an item of the Color option class equals the Stain user-defined attribute in an item of the Tint option class.

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

Related Topics

- [Compatibility Rules](#)
- [CDL Expressions](#)
- [CDL Functions](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)

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

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

FOR ALL and IN Keywords

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

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

WHERE Keyword

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

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

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

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

Examples

In the following example, the result is 3 contributions to option a.

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

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

Example: FOR ALL IN and WHERE Clause using Node Attributes

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

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

Related Topics

- [CDL Statements](#)
- [CDL Iterator Statements and the COLLECT Operator](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)
- [CDL Accumulator Statements and the ADD or SUBTRACT Keywords](#)

CDL Iterator Statements and the COLLECT Operator

The COLLECT operator supports iterator statements.

This section includes the following:

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

COLLECT Operator

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

COLLECT Operator, Single Contribution

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

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

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

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

```
CONSTRAIN &varA IMPLIES model.optionClass.item
FOR ALL &varA IN {Option11, Option32, OptionsOf(optionFeature1)}
```

```
WHERE &varA.userAttrs["UDA1"] = 5;
```

The following example uses COLLECT, which prevents the error.

COLLECT Operator Contributions

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

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

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

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

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

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

COLLECT Operator with DISTINCT

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

Related Topics

- [CDL Statements](#)
- [CDL Iterator Statements and the COLLECT Operator](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)
- [CDL Accumulator Statements and the ADD or SUBTRACT Keywords](#)

CDL Constraint Statements and the CONSTRAIN Keyword

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

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

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

- REQUIRES
- NEGATES
- IMPLIES
- EXCLUDES

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

Examples

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

Constraint statements with the `CONSTRAIN` keyword

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

Constraint statements without the `CONSTRAIN` keyword

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

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

Constraint Statement with the `FOR ALL...IN` Iterator

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

Related Topics

- [Logic Rules](#)
- [CDL Statements](#)
- [CDL Iterator Statements and the COLLECT Operator](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)

CDL Compatibility Statements and the `COMPATIBLE ...OF` Keyword

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

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

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

Example

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

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

Related Topics

- [Compatibility Rules](#)
- [CDL Statements](#)
- [CDL Iterator Statements and the COLLECT Operator](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)
- [CDL Accumulator Statements and the ADD or SUBTRACT Keywords](#)

CDL Accumulator Statements and the ADD or SUBTRACT Keywords

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

Example

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

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

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

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

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

If	AND	Then
A is an integer	Option 1 and Option 2 are both integers	No further action is needed on A

Related Topics

- [CDL Statements](#)
- [CDL Iterator Statements and the FOR ALL...IN and WHERE Keywords](#)
- [CDL Iterator Statements and the COLLECT Operator](#)
- [CDL Constraint Statements and the CONSTRAIN Keyword](#)
- [CDL Compatibility Statements and the COMPATIBLE ...OF Keyword](#)

Extension Rules

Overview of Extension Rules

Extension rules extend the functionality of configurator rules, through code that you write yourself.

This section provides information on the following:

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

Related Topics

- [Extension Rules](#)
- [Configurator Events](#)
- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [Create an Extension Rule](#)

Extension Rules

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

- An extension rule is bound to one or more predefined events that can happen during a configuration session. Example: A change in the value of a decimal feature node.

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

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

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

Related Topics

- [Configurator Events](#)
- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [Create an Extension Rule](#)

Configurator Events

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

Event Binding

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

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

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

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

Event Binding Scope

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

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

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

Event Descriptions

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

Event	Description
postConfigInit	Event dispatched immediately after initializing a new or restored configuration session. You can use this event to set values in your model as soon as an end user starts configuring the model.
postConfigSave	Event dispatched immediately after a configuration is saved. You can use this event to set the values in your model whenever an end user saves changes to the configuration of the model.
postValueChange	Event dispatched immediately after the value of a node is changed. You can use this event to set the values in your model whenever an end user changes the value or selection of the node associated to the event explicitly or by rules.
preAutoComplete	Event dispatched immediately before the autocompletion process runs. The autocompletion process is triggered by clicking Finish or Finish and Review in the configurator UI, at run time. You can use this event to set the values in your model before an end user reviews the configuration, or saves the configuration and finishes the configuration session.

Related Topics

- [Extension Rules](#)
- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [Create an Extension Rule](#)

How You Create Extension Rules

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

The essential tasks in creating an extension rule are:

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

Prerequisites

The following things are required when defining an extension rule.

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

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

Creating Extension Rules and Assigning the Base Node

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

You can associate an extension rule with any type of node, within the model containing the rule.

4. Right-click the model node and select **Set as Base Node** from the context menu. The fully-qualified node name path of the model node is inserted in the **Base Node** field.

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

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

Entering the Rule Text

The rule text defines the behavior of the extension rule.

1. Enter the text of a valid Groovy script into the **Rule Text** field. Click **Save** to save the rule text and the rest of the rule definition.
Tip: You can save the rule as you work on the rule text, even if it's not complete or correct.
2. To ensure that the extension rule, and the rule text, are valid, click **Validate**, at any point during the definition.

Adding Event Bindings

Add at least one event binding to the rule.

1. In the Event Bindings table, click **Create**
2. In the new row for the event binding, from the **Event** list, select the event to bind.
3. From the **Event Scope** list, select the event scope in which the event is to be evaluated.
4. From the **Class** list, select the class that contains the method that you're binding to the event.
If no classes are defined in the script, select **ScriptClass**, which contains globally-defined methods.

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

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

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

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

Adding Argument Bindings

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

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

Validating Extension Rules

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

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

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

Related Topics

- [Extension Rules](#)
- [Configurator Events](#)
- [How You Write Extension Rule Text](#)
- [The Extension Rule Text Editor](#)
- [Create an Extension Rule](#)

The Extension Rule Text Editor

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

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

Related Topics

- [Extension Rules](#)
- [Configurator Events](#)
- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [Create an Extension Rule](#)

How You Write Extension Rule Text

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

The Groovy Scripting Language

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

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

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

- Many base Java packages are automatically imported, so that you don't have to import them in your scripts.
- You can declare variables with the `def` keyword, without having to declare their type.
- You can define methods and variables outside a class. They're considered as global within the script. Global definitions are executed within the built-in class `scriptClass`, and executed under the built-in method `run()`.
- Some advantages to placing methods and variables in classes are:
 - You can use encapsulation and inheritance.
 - You can build complex logic, as in Java.
 - Organizing methods in classes makes it easier to select them from the **Class** and **Method** lists when adding event bindings.

Advantages to not placing methods and variables in classes include:

- If you don't use classes, you can write code and test your code more quickly, simply adding and running code bound to the global class `scriptClass` and the global method `run()`.

Accessing Important Configurator Objects

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

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

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

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

Object	Entity Accessed	Interface Used	Example of Creating the Object	Example of Using the Object
Configurator event	The event specified by an event binding in the rule.	ICXEvent	No code needed. The object cxEvent is created automatically when the rule runs.	Get the current configuration associated with the event that triggered the rule. <pre>ICConfiguration config = cxEvent.getConfiguration()</pre>
Base node of rule	The base node associated with the rule.	ICXEvent	Get the base node of the rule triggered by a bound event. <pre>def baseNode = cxEvent.getBaseNode()</pre>	Test whether the base node of the rule is selected by the end user. <pre>if (baseNode.isSelected()) ...</pre>
Configuration	The active configuration during a session.	IConfiguration	Get the configuration affected by the rule triggered by a bound event. <pre>IConfiguration config = cxEvent.getConfiguration()</pre>	Get the root node of the item-based model currently being configured. <pre>def root = config.getRootBomModel()</pre>
Root node of model	The root node of the model being configured.	IBomModelInstance	Get the root node of the item-based model currently being configured. <pre>IBomModelInstance root = config.getRootBomModel()</pre>	Get a child node of the current model, using its item name to search the model tree, starting from its root node. <pre>def childItem = root.getChildByName("CM85010")</pre>

Interacting with Model Node Values

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

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

- The Java interface containing methods to access that object
- The prototype of a method for getting the current value or state of the object.

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

The interfaces of the Configurator API also provide many other interfaces and methods for other types of interactions with node objects. This table provides an introduction to methods that are useful for common operations. For details, see the [Java API Reference for Oracle Fusion Configurator](#).

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

Other Model Interactions

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

- Getting and Setting Logic States
- Accessing properties
- Access to options
- Overriding contradictions
- Handling logical contradictions
- Handling exceptions

Reference Documentation for Available Classes

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

The reference documentation for the interfaces in that package that you use in writing extension rule scripts is the [Java API Reference for Oracle Fusion Configurator](#).

Some members of supported classes and interfaces aren't supported for use in extension rules, and their use will cause an error upon compilation. Unsupported members are omitted from the API reference documentation. If your script refers to a class or member that's not available for use in extension rules, a validation message identifies the invalid reference.

Related Topics

- [Extension Rules](#)
- [Configurator Events](#)
- [How You Create Extension Rules](#)
- [The Extension Rule Text Editor](#)
- [Create an Extension Rule](#)

Create an Extension Rule

In this example, you define an extension rule that calculates the effect of applying different discounts to the list price of an item, in order to arrive at the sale price.

The formula used in this example to calculate the discount is simple, but you can use this technique to construct more complex extension rule functionality, through the script in your rule text.

The tasks in this example are:

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

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

Defining Supplemental Features

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

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

Field	Value
Name	List Price
Minimum	20,000

Field	Value
Maximum	60.000

Field	Value
Name	Discount
Minimum	0
Maximum	10

Field	Value
Name	Sale Price
Minimum	0
Maximum	100.000

7. Click **Save**.

Creating the Extension Rule

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

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

Writing the Rule Text

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

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

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

```
import oracle.apps.scm.configurator.runtime.core.IDecimalFeature

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

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

// Calculate the price.
salePrice = listPrice - ( listPrice * ( discount / 100 ) )

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

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

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

Defining the Event Binding

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

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

Defining the Argument Bindings

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

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

Testing the Model

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

1. On the top of the Edit Configurator Model page, select **Test Model**.
2. In the Test Model dialog box, ensure that **User Interface** is set to `Default`, then click **OK**.
3. On the Test Model page, fields for the three decimal features that you added should appear, in the sequence that you created them:
 - o List Price
 - o Discount
 - o Sale Price
4. In the **List Price** field, enter the value 30,000. Notice that you can only enter values between the minimum and maximum that you defined when you created the decimal feature.
5. In the **Discount** field, enter the value 5. Notice again that the allowable values are between the minimum and maximum that you defined.
6. When you press Enter in the Discount field, or tab out of it, notice that the value of the **Sale Price** field changes, from empty to 28500.
7. Enter a different value in the Discount field, and notice the changes in the Sale price field. This change happens because you bound the execution of the extension rule to changes in the value of Discount.
8. Enter a different value in the List Price field, and notice that there is no change in the Sale price, because there is no binding in the rule to changes in the value of List Price.
9. Click **Finish** to end the test session.

Related Topics

- [Extension Rules](#)
- [Configurator Events](#)
- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [The Extension Rule Text Editor](#)

How You Obtain External Information Using Extension Rules

You can call an external service from within an extension rule.

This is useful when you need to consult an external source to assist in the proper configuration of a model. It is often more efficient to obtain data (such as product specifications or government regulations) from an external source than to attempt to maintain it in your model.

In order to call an external service from an extension rule, you must:

- Register a web service connection to be used by a service call
- Define an extension rule in which the script calls the service using that connection

Registering the Web Service Connection

In order for Configurator to access an external web service, you must register a connection for it.

To define a web service connection:

1. In the Setup and Maintenance work area, select **Search** from the task panel. Search for and select the task **Manage External Service Details for Extensions**.
2. In the Web Service Details table, select **Add Row** from the Actions menu.
3. From the **Target System** list, select a registered trading community partner application or Oracle Application.
4. In the **Connector Name** field, enter the name that you will use to refer to the service, in your extension rule. such as `Axe1WS`.
5. In the **Connector URL** field, enter the URL to the external web service, such as `http://axe104.com:7011/services/Axe1ServicesPort`.
6. From the **Invocation Mode** list, select **Synchronous service**, which is the mode supported.
7. Optionally enter other service information that may be required for calling the web service, such as User Name and Password.
8. Select **Save and Close**.

Defining the Extension Rule

In order for Configurator to call an external web service, you must define an extension rule that uses a particular API to call the service and specify the desired operation.

1. Navigate to the Configurator Models work area.
2. In a workspace, add your model and open it for editing.
3. On the Rules tab of the Edit Configurator Model page, select **Create Extension Rule** from the Actions menu.
4. In the Structure pane, select a base node for the rule, such as the root node of the model.
5. In the Rule Text pane, enter a valid Groovy script that calls the external service. A sample script is provided in this topic.
6. When defining the event binding for the rule, select when you want the service to be called:
 - o To call the service when the configuration session begins, select `postConfigInit`.
 - o To call the service when the associated model node value is changed, select `postValueChange`.
7. For the example script provided, the event binding must have a **Class** of `ScriptClass` and a **Method** of `getCpuHw(String region)`.
8. The argument binding for the argument `region` must be a value that can be interpreted by the web service.

Calling an External Service

The example script provided here demonstrates the essential points of calling an external web service in an extension rule script.

Important points to remember:

- It is essential to import the classes `SoapServiceResponse` and `SoapServiceRequest`. The need for other imports is determined by the desired behavior of your script.
- When constructing the payload string, this example uses the namespace `http://services.axe1` and the service method `cpuHw`. Assume that these are defined in the web service that you registered.
- You must use the method `invokeSoapService()` from the configuration object to call the service, and pass it the connector name that you registered and the payload string that you constructed. In this example, the connector name is `Axe1WS`.
- Some external SOAP services can only be invoked using SOAP 1.1 specs. They don't work via SOAP 1.2, possibly because WSDL is only SOAP 1.1 compatible. In order to get around this issue, add the following HTTP headers when invoking such services from an extension rule:

- Content-Type: `text/xml`
- SOAPAction: `executeWebCX`

Here is an example of how these HTTP headers can be set in an extension rule:

```
SOAPMessage message = request.getSoapMessage();
message.getMimeHeaders().addHeader("Content-Type", "text/xml; charset=utf-8");
message.getMimeHeaders().addHeader("SOAPAction", "executeWebCX");
```

If these HTTP headers aren't set then a SOAP 1.1 call fails.

This example script constructs a SOAP message for a web service that tries to select some items:

```
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 XML payload here
        QName bodyName = new QName("ws", "executeWebCX");
        SOAPBodyElement bodyElement = request.getSoapBody().addBodyElement(bodyName);
        bodyElement.addNamespaceDeclaration("ws", "http://services.axel/");
        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 {
// Invoke the external web service that was registered in the Setup and Maintenance work area
response = node.getConfiguration().invokeSoapService("AxelWS", 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();

}
}
```

Related Topics

- [How You Create Extension Rules](#)
- [How You Write Extension Rule Text](#)
- [Create an Extension Rule](#)
- [Configurator Events](#)

FAQ for Model Rules

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

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

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

Related Topics

- [The Constraint Definition Language](#)

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

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

If you don't make an invalid rule valid, you can still release the model if you disable that rule.

Related Topics

- [The 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 the configuration behavior for a node changes, then you might have to change the type of control template that renders the node in the UI. For example, if an option class that originally had mutually exclusive options now allows multiple selections, then you might want to change the control template for the node's page item from a radio button group to a check box group.
- If supplemental structure nodes have been reordered in the model structure, then you must reorder them in the existing UI, if they're explicitly displayed in the UI. The option nodes of an option feature aren't affected.

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:
 - o **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.
 - o **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 instantiable 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, an instantiable 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.
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 instantiable 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 instantiable 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.
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.

Template Name	Description
Enhanced Item Selection Table, Single-Select	Table for selecting or configuring a single item, showing quantity, description, pricing, and detailed selection state.
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 an instantiable 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 orderable 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 an instantiable child model.
Item Instance Management Table with Facets	Table for adding, deleting, and configuring instances, and entering quantities for an instantiable 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

Template Name	Description
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.
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 an instantiable 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 an instantiable 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 orderable 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 orderable 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 orderable 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 orderable 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.

Template Name	Description
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.
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.
 - o **Add as Page Item** adds the node as a page item, which enables you to select a different control template and display condition.
 - o **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.
 - o **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 autocompletion process can run (which is triggered by clicking **Finish** or **Finish and Review** at run time). If you select the **Prerequisite for autocomplete** check box on the UI Presentation tab of the node details, in the Structure view of the Edit Configurator Model page, then an asterisk appears next to the node in the UI when the required value is missing. This indicator appears in any UI, and 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.

- o 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
```

- o 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
```

- o 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
```

- o 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:

- o 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}
```

- o 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 **SelectionMode**.
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	<code>Item Name: #{amn.name} - Item Description: #{amn.description}</code>
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](#)
- [The UI Expression Language](#)
- [Use Node Properties to Affect Runtime Behavior and Results](#)

The UI Expression Language

You can use the UI expression language to dynamically create values for text parameters of basic UI elements based on the associated model node attributes of any basic element UI item.

UI Expression Language Syntax

Expressions in the UI expression language consist of a reserved keyword, which refers to an associated model node, qualified with a name that refers to an attribute of the model node.

An example of using the expression language is:

The node named `{amn.name}` has a maximum quantity of `{amn.maxQuantity}`.

- All expressions start with `{` and end with `}`.
- The keyword `amn` establishes a reference to the associated model node for a UI basic element.
- Expressions are case-sensitive, including the keyword `amn`.
- The keyword `amn` is always qualified by a reference to an attribute of the associated model node, following this form:
`{amn.referenceName}`
- References must be to a valid attribute of the associated model node. Valid attributes are enumerated in the user assistance for configurator model node properties. References to attributes that don't apply to the associated model node generate errors when attempt to use them in expressions.
- 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
<code>name</code>	The name of the associated model node.	<code>{amn.name}</code>
<code>description</code>	The description of the associated model node.	<code>{amn.description}</code>
<code>quantity</code>	The quantity of the associated model node.	<code>{amn.quantity}</code>
<code>boundQuantity</code>	Attributes named like <code>IsBoundQuantity</code>	<code>{amn.boundQuantity}</code>
<code>hasChildren</code>	Attributes named like <code>HasChildren</code>	<code>{amn.hasChildren}</code>
<code>value</code>	The value of a transactional attribute, if the associated model node is an attribute under the root model node.	<code>{amn.value}</code>

UI Expression Language Reference	Refers To	Example Expression
<code>suppAttrs['<suppAttrName>']</code>	The value of the named supplemental attribute.	<code>#{amn.suppAttrs['color']}</code>
<code>userAttrs['<UDAttrGroup>.<UDAttrName>']</code>	The value of the named user-defined attribute.	<code>#{amn.userAttrs['PhysicalAttributes.Color']}</code>

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**.

Uses for the UI Expression Language

Uses for the expression language include the following:

- Creating 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}
```

- Creating 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}
```

- Storing a central set of Cascading Style Sheets (CSS) in a model node's supplemental attribute and using 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 item images displayed at run time in a horizontal row on the UI page.
 - **Inline Style** is an optional CSS style expression applied at run time to the entire Selectable Image Group template.
 - **Option Inline Style** is an optional CSS style expression applied at run time to each selectable image.
6. Save your model changes and click **Test Model**.
7. In the test UI, the images displayed for the options of the option class or option feature depend on the selection state of the options:
 - The Primary image is displayed if the option is available, and not selected or excluded.
 - The Selected image is displayed if the option is selected by a rule or user selection.
 - The Excluded image is displayed if the option is excluded by a rule.
8. If any images are missing from the **Item Selection Images** of the Structure view, then the following substitutions are made at run time:
 - If the Primary image for an item is missing, then a graphic placeholder icon is displayed.
 - If the Selected image is missing, and the item is selected, then the Primary image is displayed, enclosed by a rectangle.
 - If the Excluded image is missing, and the item is excluded, then the Primary image is displayed, in dimmed shading.
9. If your UI uses the Selectable Image Group with Header template, then the Description of the option class or option feature and the Description of the selected option are displayed together as a header for the group of option images.

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	SelectionMode	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:
 - o The specified item class itself
 - o All standard items that are associated with the specified item class
 - o 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 orderable trailers, created in the Product Information Management work area. The example item class is: Trailers - Trailers
Standard items	The set of orderable 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()));
```

Related Topics

- [Create an Extension Rule](#)

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 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.
 - o 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.
 - o 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:
 - o The Name is MyConnector.
 - o The target item class of your choice.
 - o The Minimum Connections and Minimum Quantities are 1 or more.
 - o 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:
 - o The Base Node is set to the root node of the model.
 - o This event binding:

Field	Value
Event	postConfigInIt
Event Scope	Global
Class	ScriptClass
Method	run()

- o 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:
 - o The running summary, if you included that template in a UI that you created.
 - o 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.

Copy `https://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.
 - o **Finish:** The configurator engine finishes the configuration, which means that the configurator engine uses the autocompletion 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.
 - o **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.
 - o **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.
 - o **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 orderable 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, since they aren't orderable, 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 the user 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-DSI2)
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 Bluera y 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: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:NodeProperty>
          </ns2:QueryRequest>
        </ns1:request>
      </ns1:testModel>
    </soap:Body>
  </soap:Envelope>
```

```

<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 orderable items 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
<code>TestId</code>	The unique identifier for the current test.
<code>OperationId</code>	A unique identifier for the operation request
<code>NodePath</code>	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, AddInstanceWithQuantity, AddInstanceWithQuantityAndName, RenameInstance, CopyInstance, DeleteInstance, SetQuantity, SetContextPath	Model Node Reference

The operations that can be performed at the configuration level include those in the following list.

- **AutoComplete:** Performs an autocompletion operation on the configuration, to include all the adjustments needed to make the configuration valid.
- **UndoAutoComplete:** Reverts the autocompletion operation.
- **AdjustConfiguration:** Puts the configuration into adjust mode after autocompletion, 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 <code>AvailableChildren</code> , <code>SelectedChildren</code> , or <code>SelectableChildren</code>). See the following description of <code>CollectionProperty</code> 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: <ul style="list-style-type: none"> 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 orderable items 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, which leads us to. . .

Specify the Configuration Effective Date

You can use the Configuration Effective Date parameter to tell Configurator when your configuration goes into effect.

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.

You can use the Configuration Effective Date order management parameter to specify when your configuration goes into effect, and that date affects the model's version that Configurator uses. Configurator compares the version that you submit to the version that exists according to when the configuration goes into effect.

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	Use the Snapshot Version That's in Effect On
Configuration Date	<p>Use the version that's in effect on the date that you add the order line and configure the item on the line.</p> <p>Assume you create an order line, add the Viking Stove to the line and configure the stove on June 1. The configuration date is June 1.</p> <p>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.</p>
Ordered Date	<p>Use the version that's in effect on the ordered date.</p> <p>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.</p> <p>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.</p>
Current Date	<p>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.</p> <p>The current date is the date that you submit the order to fulfillment.</p> <p>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.</p> <p>If you don't set the parameter to any value, then Order Management uses the current date, by default.</p>
Requested Date	<p>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.</p>

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.

```
Viking 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 Viking 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. See [Extension Rules](#).
- 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 serialization and deserialization of 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 isn't a technology, but rather a client-server architectural style in which participants perform actions on data-oriented resources.

Integration Architecture

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: <ul style="list-style-type: none"> • 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 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 `OupputMode` 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:

- o Offering: Product Management
- o Functional Area: Product Spoke Systems
- o 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:

- o **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.
- o **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.
- o **Finish or Finish and Review:** The end user completes the configuration, by implicitly running the autocompletion 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.

- o The `ExitStatus` parameter indicates the exit method. Possible values are `save`, `cancel` and `error`.
- o 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
```

- o 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=ConfigurationLine
```

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](#)

