

Sterling Configurator



Application Guide

Release 9.1.0.52

Sterling Configurator



Application Guide

Release 9.1.0.52

Note

Before using this information and the product it supports, read the information in "Notices" on page 73.

Copyright

This edition applies to the 9.1 Version of IBM Sterling Configurator and to all subsequent releases and modifications until otherwise indicated in new editions.

© **Copyright IBM Corporation 1999, 2011.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Chapter 1. What is Sterling Configurator?	1
Chapter 2. Item Configuration: Implementation	3
Item Configuration: Solution	3
Item Configuration: Tabbed and Normal Layout	5
Item Configuration: Step-Wise Layout	7
Item Configuration: End-User Impact	9
Item Configuration: Implementation	10
Chapter 3. Deploying with WebSphere Commerce or third party applications	13
Create and Deploy the Sterling Web EAR Package	13
Calling Sterling Configurator	14
How end customers access Sterling Configurator in the production environment	16
Chapter 4. Localizing Sterling Configurator - An Overview	17
Prerequisites to Localizing the Sterling Configurator	17
Localizing Resource Bundles.	17
Localize Resource Bundles	18
Localizing Factory Setup Data	19
Chapter 5. Customizing Sterling Configurator: An Overview	21
Chapter 6. Customize Actions	23
Action Definition Customization Checklist	23
Customize Action Classes	24
Chapter 7. Customize Themes	25
Theme Customization Checklist	25
Customization examples	25
Theme Customization for a Storefront	25
Chapter 8. Customize Controls	27
Customizing an Existing Control	27
Create a new control	28
Chapter 9. Customize Control Handlers	29
Chapter 10. Customize Function Handlers	31
Function Handler Example	32
Chapter 11. Data Validation	35
Data Validation	35
Customize Data Validation	43
Chapter 12. Deploying your customizations	45
Chapter 13. Sterling Configurator Best Practices	47
Designing Models	48
Properties	51
Rules	55
Modular Development.	58
Tools	58
Performance	61
Development and Maintenance.	61
Chapter 14. Definitions for Out-of-the-box Functions	63
Chapter 15. Definitions for Out-of-the-box Configurator Properties	65
Notices	73
Index	77

Chapter 1. What is Sterling Configurator?

The IBM® Sterling Configurator is an application that is used to display configurable items to customers, thus enabling customers to select the configurations that best meet their needs. A configurable item offers various options from which a customer can select before purchasing the item. Based on how the model item has been created, Sterling Configurator is used to display the various configurations available for the item. This enables a user to select the configurations that best meet their needs.

For example, let us suppose that the product catalog contains an item, Bicycle. The item, Bicycle, represents a model created using a modeler such as the Sterling Configurator Visual Modeler. This item has four frame types, three wheel types, and so on. At the time of purchase, the customer can select one of the four frame types, one of the three wheel types, and so on. This enables the customer to select the configuration that best meets the customer's needs.

Summarize the big picture: install / build EAR & deploy to test / customize, localize / build EAR & deploy to production. Revise to summarize the 2 deployment scenarios, with Suite and with WC/third party.

Chapter 2. Item Configuration: Implementation

This section explains the configurations for this functionality:

- To enable a user to configure an item, the item must be created as a configurable item or a preconfigured item and then associated with a model using the Sterling Business Center application.
- You can configure whether to display the tabbed configuration page, normal configuration page, or step-wise configuration page by defining the UI: JSP FILENAME display property in the Sterling Configurator Visual Modeler.
- When an order is purged, the corresponding order lines and the saved configurations are also purged. For more information about order purge, refer to the *Sterling Selling and Fulfillment Foundation: Distributed Order Management Configuration Guide*.
- To configure the BOM validations during checkout, ensure that the **Validate Configurable Items When Viewing a Cart** rule is enabled in the Channel Applications Manager. For more information about configuring the **Validate Configurable Items When Viewing a Cart** rule, refer to the *Sterling Selling and Fulfillment Suite: Applications Configuration Guide*.
- To configure the BOM validations during the submission of a request to the server, ensure that the **Validate Configurable Items During Checkout** rule is configured in the Channel Applications Manager. For more information about configuring the **Validate Configurable Items During Checkout** rule, refer to the *Sterling Selling and Fulfillment Suite: Applications Configuration Guide*.
- To enable sub model validation, ensure that the CONFIG: VALIDATE SUBMODEL property is set in the Sterling Configurator Visual Modeler. For more information about setting the CONFIG: VALIDATE SUBMODEL property in Sterling Configurator Visual Modeler, refer to the *Visual Modeler: Administration Guide*.
- To disable BOM validations during the checkout process or order submission, set the SkipBOMValidations attribute to Y in either the changeOrder API or the createOrder API. For more information about the changeOrder API and the createOrder API, refer to the *Sterling Selling and Fulfillment Foundation: Javadocs*.
- To display the images of option items available under an option class, you must set the UI: SHOW ITEM IMAGES property to "true" at the option class level in the Sterling Configurator Visual Modeler, and specify the image for the option items by setting the UI: ITEM IMAGE NAME property, in the image URL field at the option item level, appropriately. For more information about working with display properties, refer to the *Visual Modeler: Administration Guide*.
- A user can set UI controls, create constraints, set model groups, models, option classes, option items, and rules. For more information, refer to the *Visual Modeler: Administration Guide*.

Item Configuration: Solution

Channel applications such as IBM Sterling Field Sales, IBM Sterling Web, and IBM Sterling Call Center and IBM Sterling Store enable a user to configure items through the Sterling Configurator before adding the items to a cart, quote, or an order. You can offer users the complete range of options available, including preconfigured items containing predefined choices. Users can purchase the preconfigured items as is or use them as the starting point for performing additional configurations.

The `sic_properties.zip` file located in the `<INSTALL_DIR>/repository/external` folder contains the `.properties` files that are utilized by the Sterling Configurator. These `.properties` files must be placed in the appropriate folder as configured in the Applications Manager.

The following files are located in the properties folder.

- `configurator.properties` — This file comprises the basic properties that can be used to modify the functionality of Sterling Configurator. For example, the number of models that can be cached by Sterling Configurator can be modified by making appropriate changes to this file. This file comprises the `pricingType` property that defines whether the price of an item is taken from the pricelist or from a model. The valid values that can be defined for the `pricingType` property are as follows:
 - `STATIC_PRICING` — To pick the price of an item from the model.
 - `DYNAMIC_PRICING` — To pick the price of an item from the price list.
 - `OVERRIDE_PRICING` — To pick the price from the price list. If the price for the item is not found in the price list, the price defined for the model is used.

Note: If the price of an item is not configured as part of the price list configuration in the IBM Sterling Business Center application, the price of the item displayed in the Sterling Configurator will be based on the price configured in the Visual Modeler. To use the same price as configured in the Visual Modeler for either a cart or an order, the `CONFIG: PRICE LOCKED` property must be set as `1` for the corresponding option item in the Visual Modeler.

- `controls.properties` — This file defines the set of controls that are available for Sterling Configurator. A user can control the functionality of the option classes and option items displayed in the UI.

Note: `DynamicInstantiationControlHandler` is a control handler class that dynamically adds child option items to a model when it is retrieved from the cache and removes the dynamic items when the model is returned to the model cache.

- `functionHandlers.properties` — The function handlers are declared in this file. Sterling Configurator provides a rule engine that is used to evaluate the rules defined for each model. The rule engine can invoke custom functions to handle scenarios where existing functions are unable to solve a configuration specification. The `lookupValues.properties` file is used by the sample `LookupFunctionHandler`. The `webServiceLookup.properties` file is used by the sample `WebServiceLookup` handler.
- `pagetypes.properties` — This file is used to determine the type of page that should be displayed when the Sterling Configurator is launched. The UI page templates that control how a model is rendered are defined in this file. Valid page templates are `NORMAL`, `TABBED`, and `STEPWISE`.

All the paths comprising the location where the models, `.properties` files, and rules are stored, are configured in the Applications Manager. The following JAR files are located in the `<INSTALL_DIR>/jar/smcfs/<current_version>` folder and must be copied to the folder specified in the path configured for the rules:

- `cmgt-rulesEngine.jar`
- `cmgt-configuredItem.jar`
- `cmgt-configurator.jar`

For more information about configuring the Sterling Configurator rules, refer to the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Based on the way the user interface display property of the model item is configured in the Visual Modeler, the configuration page is displayed in one of the following layouts:

- Tabbed
- Normal
- Step-Wise

For more information about working with display properties, refer to the *Visual Modeler: Administration Guide*.

Note:

- To configure an item using the Sterling Configurator, ensure that the item's Effective Start Date and Effective End Date are configured in the Sterling Business Center application.
- The image of a selected item is displayed only when you select the radio button or the check box next to it in the step-wise configuration page.

Item Configuration: Tabbed and Normal Layout

In the Tabbed configuration layout, the option classes for a model are displayed under the respective tabs, with the corresponding option items listed under each option class. In the Normal configuration layout, the option classes are displayed on a single page along with the corresponding option items. Whenever an action is performed in the configuration page of the Sterling Configurator, a request is sent to the server from the configuration page and the ConfiguratorController is called to handle that action.

Depending on the action that is called in the configuration page, the system first performs one or many of the following operations, which are also referred to as initializeConfigurations, using the Configurator business object:

- Get and process existing XMLs
- Get prices
- Apply pricing
- Apply picks
- Fire rules
- Compute prices
- Get list of controls
- Get list of existing tabs

Following are the actions that can be called in the Tabbed configuration and the Normal configuration page:

- **configure** — This is the main action handled by the Sterling Configurator. This action is called when the Sterling Configurator is launched, a customer clicks on a tab, or a customer selects an option item in the configuration page. This action takes the results of the initializeConfiguration operation as the input, sets the appropriate attributes on request, and sets the value of the configURL parameter to display the appropriate page.
- **addToCart** — This action is called when a customer adds a configurable item to either a cart or an order. This action takes the results of the initializeConfiguration operation and builds the bill of material (BOM) that is

passed to the cart. The control is passed to the URL specified in the returnURL parameter. During the checkout process or when a user submits an order, BOM validations are disabled if the SkipBOMValidations attribute is set to Y in either the changeOrder API or the createOrder API. BOM validations are not performed if an order is in Draft status. The BOM validations can be activated or deactivated during the submission of a request or when a customer performs a checkout. To configure the BOM validations during checkout, ensure that the **Validate Configurable Items When Viewing a Cart** rule is configured in the Channel Applications Manager. To configure the BOM validations during the submission of a request to the server, ensure that the **Validate Configurable Items During Checkout** rule is configured in the Channel Applications Manager.

- To configure the BOM validations during the submission of a request to the server, ensure that the Validate Item rule is configured in the Applications Manager.
- **summary** — This action is called when a user clicks the **Summary** button. It builds the Sterling Configurator BOM based on the results of the initializeConfiguration operation and sets the request attributes required to display the Summary page.
- **subModelConfig** — This action is called when a user navigates from a parent model to a submodel. This action gets the submodel, sets the necessary input properties from the parent model and the attributes required to display the submodel, and displays the submodel for configuration.
- **subModelReturn** — This action handles the transition from a parent model to a sub model. It takes the set of output properties from the submodel, adds them to the parent model, and displays the parent model again. A parent model is configured with a sub model through the Sterling Configurator Visual Modeler. A customer can specify sub model validation by setting the CONFIG: VALIDATE SUBMODEL property.
- **showRuleTrace** — This action is called when a user clicks the **Debug** button when the Sterling Configurator is run in debug mode. It sets the property pool and trace messages based on the results of the initializeConfiguration operation, and sets the attributes necessary to display the rule trace.
- **resolveConflict** — This action is called when a user clicks the **Resolve** button to resolve a constraint violation. It collects information about the item that caused the constraint violation and builds the set of alternatives that can be used to resolve the constraint. These alternatives are displayed on the Resolver page.
- **conflictResolution** — This action is called from the Resolver page to resolve a constraint violation. The resolution selected by the user is applied to the model by removing the item that caused the constraint violation and selecting the item that the user specified in the Resolver page.
- **testCart** — This action is used to display the Test Cart page. When a model is launched from the Catalog page, the returnURL specifies the standard Add to Cart action which adds the configuration to the cart. However, when the Sterling Configurator is launched through the Sterling Configurator Visual Modeler, a different returnURL is provided. This action transfers the control to a page that simulates the task of adding the configuration to the cart, with additional functionality available for debugging.

The Test Cart page comprises the following tabs:

- **Test Cart** — This tab displays the BOM to be displayed in the cart.
- **Launch New Model** — This tab enables the Sterling Configurator Visual Modeler to launch a new configuration model by passing the state of the existing model.
- **BOM Details** — This tab displays the BOM comprising the hidden items.

- Editable BOM — This tab displays the actual XML representation of the BOM, which can be used by the Visual Modeler for modifying the model with the modified BOM. The generateConfigurationBOM API is called to build the XML representation of the BOM.
- configstatus — This action displays the Configuration Status page. This page is available only when the Sterling Configurator is run in the debug mode. This action allows a customer to see the state of the cached configuration models and clear the cache, if required.

Item Configuration: Step-Wise Layout

In the step-wise configuration layout, each step involved in configuring an item is displayed as a tab. Each tab comprises option classes pertaining to that step. You can associate images with models, option classes, and option items. Under each tab, the corresponding option classes are displayed as icons and have an image and description associated with them. If no image is provided, a default image is displayed. If there are multiple option classes, a user can scroll to view the additional option classes. The step-wise configuration can be launched by calling the getConfigurationModel API. The Sterling Configurator is launched by a user by invoking the /configurator/configure.action. This action passes the name of the model to be configured as the parameter and calls the ConfiguratorService.managePicks API to get the set of picks to be displayed in the Sterling Configurator.

The ConfiguratorService.managePicks API retrieves the following information:

- The tabs that are displayed for the model.

After the Sterling Configurator is loaded, an AJAX call is made to retrieve and display the tabs. The Sterling Configurator controller processes the AJAX call and generates a list of tabs and details such as tab name, guiding text, and the status of the tab. Whenever the AJAX call is processed, the controller verifies from the cache whether the tab information is available for this model. If the information is available, the controller retrieves the cached information and builds the JSON response.

The information used to build this response is based on the cached tabs and messages retrieved from the last call to the Configurator Service. After all the tabs are retrieved, the first tab on the user interface is activated. When a tab is activated, an AJAX call is made to retrieve the list of option classes associated with that tab. When the controller processes the AJAX call, a JSON object containing the list of option classes for a given tab is created. For each option class, the option class name, icon, and status are returned. When processing this call, the controller retrieves the requested information from the controller's cache. The user can view the selected tabs with the help of visual cues. Each tab comprises the guiding text at the top of the page that is displayed when the tab is selected. The guiding text for the tabs is stored as a list property defined in the model for an item. The Sterling Configurator reads this property and constructs the tab objects to set the guiding text for each tab. A tab within a tab configuration displays a collection of option classes and option items. When a new tab is selected, a request is sent to the server to render the contents of the new tab and return it back to the client. Each time a user navigates between tabs, the configuration page is reloaded to display the current tab. getTabs is the name of the action that retrieves the tab names and message indicators, if any, to be displayed. If any of the messages pertains to an item pertaining to an option class, a message indicator corresponding to the tab is set on that tab.

- The list of top-level option classes defined in the model.

To retrieve the list of option classes, an asynchronous call is made to the server. After the list of option classes is retrieved, the Sterling Configurator saves the information and renders only those option classes that are associated with the selected tab. By default, the first option class on the page is active and highlighted with a visual cue. All the option items under pertaining to this option class are displayed under a tab. The Help me decide hyperlink displays detailed information for each of the option item available within the selected option class. The user must then click the **Submit** button to submit the selection. Whenever a user selects an option or enters a value in a text box, the selection is added to the current pick. The managePicks action manages the selection and removal of an option item by invoking the processConfigurationPicks API.

- The property pool that determines how certain elements of the model are displayed.

The property pool is used while rendering the option items within an option class. The properties in the property pool decide whether an item is visible or not, whether an image is displayed or not, and so on.

- The list of messages displayed in the messages panel.

The getMessage action retrieves the list of messages displayed in the messages panel. Each message comprises the type of message, the text of the message, and the information necessary to allow the Sterling Configurator to navigate to the source of the message

- The summary displayed in the summary panel.

Whenever a user selects an option item, the page makes an AJAX call to the server to retrieve the summary information. If this call was made in the past, the controller retrieves the summary information cached from the previous call. The controller processes this request and retrieves the summary information specified for the selected model. The getPricingSummary action returns pricing summary details such as the base price, order-level pricing discounts, and so on. The summary panel displays the price of the selected items after applying the pricing rules, if any. If any item-level pricing rule has been applied to the selected item, the adjustments resulting from the pricing rule are displayed. However, these adjustments are displayed only if the pricingType property is set to DYNAMIC_PRICING or OVERRIDE_PRICING. The pricing information is calculated when a pick is added to a configuration and is cached on the server. The getPricingSummary action retrieves the cached information and sends it to the JSON object to be displayed on the user interface.

For each option class that has been selected by the user, the corresponding option items are displayed with their price. An image of the option item that is currently selected is also displayed. If the user moves the pointer over an option item, the image of this option item is displayed. However, the image will not be displayed if you have configured not to display the image in the Visual Modeler. The displayChildren action invokes the getConfigurationModel API to get the option items of an option class.

On the configuration page, if a user selects a single selection control, the addPick action is invoked. This action removes the previous pick and adds the new pick. If the user selects a check box or a multiselect control, the previous selection is not removed and the addPick action is invoked to pick a new selection. When a selection is made in an option class, the Sterling Configurator calls the processConfigurationPicks API with the new pick information. This API comprises a new set of picks cached along with messages resulting from the call. This cached information is available for responding to any asynchronous calls made to the server.

The following information can be cached:

- Tabs
By default, the first option class for a model is active, highlighted with a visual cue, and is displayed under a tab. Whenever a tab is selected, it displays the corresponding option items.
- Picks
The selections made by a user on the option class are defined as picks.
- Messages
The messages are displayed in the message panel, in the user interface, based on the rules that are fired on the model.
- Summary Rail Information
The Sterling Configurator displays the summary of selections made by a user in the Summary Rail area. The summary is updated every time the user selects a new option item. Users can view their selections. Additionally, buyer users can save the selections and purchase the configured item. You can specify the duration of time for which saved configurations will be available to the users using a cron job. This enables you to clean up old and outdated configurations. The Summary Rail displays the summary of option item selections under different headers. The summary headers are displayed based on the way they are configured in the Visual Modeler. You can configure the summary headers by defining a list property and attaching it to the root node of the model. If no property is set on the root node of the model, the model's tabs will be displayed as summary headers.

A pick made by the user in the Sterling Configurator may result in a constraint. The constraint table is used in a model to specify the valid configurations. All the information required to resolve the constraint is stored as properties for an item. For example, a model can be designed to support a 150 GB hard disk with 512 MB RAM. If a user selects a 256 MB RAM and a 150 GB hard disk, a constraint will occur. When an error occurs because of a constraint violation, the user can resolve the constraint by clicking on the error message. A Constraint Resolution page is displayed which enables the user to select a different combination of items and resolve the constraint. The resolveConstraint action sends the control to the Constraint Resolution page. If the user does not want to resolve any constraint, the user can click the **Cancel** button.

A user can navigate between pages by clicking the **Next** and **Previous** buttons. The getNextPreviousButtons action retrieves the text that should be displayed on the **Next** and **Previous** buttons. The text to be displayed on the **Next** and **Previous** button is determined by traversing the list of option classes retrieved by the processPicks action. The option class names in the list can be searched and then the next and previous names in the list are returned.

Note: Dynamic instantiation is not supported in the step-wise configuration layout.

Item Configuration: End-User Impact

This section explains the end-user impact of this functionality:

- Based on the configuration, the Sterling Configurator displays the tabbed, normal, or the step-wise configuration page.
- When an administrator launches the Sterling Configurator with the DEBUG parameter set to TRUE, the following buttons are visible:

- Show Rule Trace
- Test Cart
- Debug mode
- Clear cache
- A customer cannot launch the Sterling Configurator from the step-wise configuration page.
- BOM validations are not performed on an order when the order is in the **Draft** status.
- Whenever a customer performs a configuration in the step-wise configuration page, the Sterling Configurator updates the screen automatically. The submit to server functionality is not supported in the step-wise layout configuration.
- A customer can view the consolidated list of items selected during configuration for tabbed and normal configurations by clicking the **Summary** button in the configuration page. A customer can view the consolidated list of items selected during configuration for step-wise configuration by clicking the **Review And Buy** button in the configuration page.
- The **Instant Savings** value is displayed in the step-wise configuration page when an item-level pricing rule is applied to the price of an item only if the pricingType property is set to DYNAMIC_PRICING or OVERRIDE_PRICING.

Item Configuration: Implementation

This section explains the configurations for this functionality:

- To enable a user to configure an item, the item must be created as a configurable item or a preconfigured item and then associated with a model using the Sterling Business Center application.
- You can configure whether to display the tabbed configuration page, normal configuration page, or step-wise configuration page by defining the UI: JSP FILENAME display property in the Visual Modeler.
- When an order is purged, the corresponding order lines and the saved configurations are also purged. For more information about order purge, refer to the *Sterling Selling and Fulfillment Foundation: Distributed Order Management Configuration Guide*.
- To configure the BOM validations during checkout, ensure that the **Validate Configurable Items When Viewing a Cart** rule is enabled in the Channel Applications Manager. For more information about configuring the **Validate Configurable Items When Viewing a Cart** rule, refer to the *Sterling Selling and Fulfillment Suite: Applications Configuration Guide*.
- To configure the BOM validations during the submission of a request to the server, ensure that the **Validate Configurable Items During Checkout** rule is configured in the Channel Applications Manager. For more information about configuring the **Validate Configurable Items During Checkout** rule, refer to the *Sterling Selling and Fulfillment Suite: Applications Configuration Guide*.
- To enable sub model validation, ensure that the CONFIG: VALIDATE SUBMODEL property is set in the Visual Modeler. For more information about setting the CONFIG: VALIDATE SUBMODEL property in Visual Modeler, refer to the *Visual Modeler: Administration Guide*.
- To disable BOM validations during the checkout process or order submission, set the SkipBOMValidations attribute to Y in either the changeOrder API or the createOrder API. For more information about the changeOrder API and the createOrder API, refer to the *Sterling Selling and Fulfillment Foundation: Javadocs*.

- To display the images of option items available under an option class, you must set the UI: SHOW ITEM IMAGES property to "true" at the option class level in the Visual Modeler, and specify the image for the option items by setting the UI: ITEM IMAGE NAME property, in the image URL field at the option item level, appropriately. For more information about working with display properties, refer to the *Visual Modeler: Administration Guide*.
- A user can set UI controls, create constraints, set model groups, models, option classes, option items, and rules. For more information, refer to the *Visual Modeler: Administration Guide*.

Chapter 3. Deploying with WebSphere Commerce or third party applications

Create and Deploy the Sterling Web EAR Package

About this task

To deploy the Sterling Web application, you must create the IBM Sterling Selling and Fulfillment Foundation EAR and the Sterling Web EAR. After configuring the appropriate property files pertaining to Sterling Web, the Enterprise ARchive (EAR) package must be deployed on the application servers. The Sterling Selling and Fulfillment Foundation EAR server and the Sterling Web EAR packages are deployed on two different servers. Sterling Web supports deployment on the Oracle® WebLogic, IBM® WebSphere®, and JBoss application servers. For information about creating and deploying EAR on Oracle WebLogic, IBM WebSphere, and JBoss, refer to the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

To make the Sterling Web application available for use, you must perform the following tasks:

Procedure

1. Set up the application server appropriately for deploying the application. For more information about setting up the application server, refer to the *Sterling Selling and Fulfillment Foundation: Installation Guide*.
2. Create the Sterling Selling and Fulfillment Foundation EAR. To create the Sterling Selling and Fulfillment Foundation EAR, run the following command:

```
.\buildear.sh (.cmd for Windows) -Dappserver=<application server>  
-Dwarfiles=smcfs,sbc -Dearfile=smcfs.ear
```
3. When Sterling Selling and Fulfillment Foundation EAR is created, the EARs located in <INSTALL_DIR>/external_deployments are deleted. To avoid the deletion of EARs, ensure that you create a backup of the external_deployments folder.
4. Create the Sterling Web EAR. To create the Sterling Web EAR, run the following command:

```
.\buildear.sh (.cmd for Windows) -Dappserver=<application server>  
-Dwarfiles=swc -Dearfile=swc.ear
```

Results

After creating the EAR files, you must deploy the EAR files on the application servers so that the Sterling Web application is ready for use. For more information about deploying the EAR file, refer to the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

After deploying the Sterling Web application, a user must create an organization (also referred as storefront in Sterling Web application), create guest user, and generate catalog search index through Applications Manager to bring up the Sterling Web application. For more information about these tasks, refer to the *Sterling Selling and Fulfillment Foundation: Application Platform Configuration Guide*.

Calling Sterling Configurator

Sterling Configurator can be integrated into an existing application so that customers can select products, configure them, and add the configured products to their cart as a seamless experience.

To enable customers to call into the Sterling Configurator, the configure struts action must be invoked with a URL of this form:

`http://<machine_name:port>/<context>/configurator/configure.action`

The following information must be part of the inbound post into the Sterling Configurator:

Input Type	Description	Comments
Model	The name of the model that Sterling Configurator must invoke.	Either ConfigurationKey or Model is mandatory
ReturnURL	URL used to return the configured product to the calling application.	Mandatory
OrganizationCode	The organization code or storefront used to fetch models, prices and entitlements.	Mandatory
ThemeId	The CSS theme used for Sterling Configurator. By default, the application uses the theme associated with the organization.	Optional
CancelURL	URL used to return to the calling application on canceling the configuration.	Optional
ConfigXML	Information about a current configuration in XML. This must conform to the DTD described in the input XML of manageConfiguration API for ConfiguratorBOM element.	Optional
Currency	Currency in which the prices are fetched by the application, based on the logged in customer profile.	
EnterpriseCode	The enterprise used for pricing.	Optional
ConfigurationDate	The date and time, passed in the format same as that of XML APIs. Example: "2014-10-29T00:00:00-04:00". This date is used to fetch prices, entitlements, rules and constraints of model. The date defaults to current date if the ConfigurationDate is not passed or if it does not exist in the session. To remove configurationDate from the session, pass ConfigurationDate as "".	
LocaleCode	Locale in which the configuration should be launched, based on the locale from the logged in customer profile	Optional

Input Type	Description	Comments
CustomerId	Customer identifier for pricing and entitlement queries. CustomerID is derived based on the logged in customer id (corresponds to the customer contact id).	Optional
DoneButton	Text for the button to exit after successful configuration. Example: AddToCart.	Optional
DEBUG	Launches the configurator in debug mode that displays the values for the properties assigned and also the rule and constraint execution status.	Optional
ConfigurationKey	The identifier of the configuration.	Either ConfigurationKey or Model is mandatory
DisplayFooter	The struts action that displays the footer.	Optional
DisplayHeader	The struts action that displays the header.	Optional
IgnoreCustomerEntitlements	If passed as "Y" or "true" customer entitlement checks are disabled. ("IsForOrdering" flag is passed as "N" to getItemListForOrdering API internally). Default value is "N".	Optional
SaveConfiguration	If passed as "true", the configuration is saved when Done/AddToCart button is clicked.	Optional
BuyerUserID	Additional call in parameter is exposed so applications can pass. The attribute exposed across API's to identify UserID in WebSphere Commerce	

A sample html submit form that calls into Sterling Configurator, hosted on localhost at port 9020 is provided below:

```
<form id="home" name="home" action="http://localhost:9020/sicapp/
configurator/configure.action" method="POST">

<input type='hidden' name="organizationCode" value="allnet" />

<input type='hidden' name="currency" value="USD" />

<input type='hidden' name="DEBUG" value="true" />

<input type='hidden' name="Model" value="Matrix/PCs/Desktops/MXDS_002D7500"
/>

<input type='hidden' name="ReturnURL" value="/sicapp/test/testCart.action"
/>

<input type='hidden' name="themeId" value="green" />

<input type="submit" value="Submit" />

</form>
```

How end customers access Sterling Configurator in the production environment

A user is a person who can perform certain functions depending on the role the user plays in the corresponding organization. Sterling Configurator provides Guest user or anonymous user support to enable users from external systems such as WebSphere Commerce to call into the application. A Guest user is provided as part of the Sterling Configurator factory setup.

A guest user is a customer who can access the Sterling Configurator without logging into the application. To enable guest user access, a guest user is created in the profile manager, with Login ID and Password as **guest**. The item prices and entitlements fetched by the application are for the anonymous/guest customer. For more information , see API Javadocs.

Chapter 4. Localizing Sterling Configurator - An Overview

The Sterling Configurator is an internationalized application, which can be used by customers from different locales to configure an item. In such a scenario, it is important to localize the user interface components of the application in the language and format that is specific to a customer's locale. It is also important to ensure that the fixed descriptive text displayed in the user interface (UI) fits the Web page area assigned to it. Sterling Configurator supports only left to right and top to bottom text layouts. Therefore, only the languages with this orientation are supported for localization.

The default locale for the Sterling Configurator application is en_US, which implies that the default language is English and the default country or region is the United States. If a UI component cannot be translated into the language of a given locale, the symbol will be displayed in the US-English format.

The Sterling Configurator application is dependent on IBM Sterling Selling and Fulfillment Foundation for localizing the formatting of currency values, date values, and numbers. For more information about understanding the concept of localization and localizing components, refer to the *Sterling Selling and Fulfillment Foundation: Localization Guide*.

The Sterling Configurator application is dependent on Sterling Business Center for localizing the following:

- Description and value of an item in the user interface.
- Information pertaining to catalogs.

A customer can modify the `ExtnCatalogSearchConfigProperties.xml` file to localize the catalog search. For more information about extending the catalog search, refer to the *Sterling Selling and Fulfillment Foundation: Extending the Database*.

For more information about localizing item descriptions and values, refer to the *Sterling Business Center: Localization Guide*.

Prerequisites to Localizing the Sterling Configurator

Before you start localizing the Sterling Configurator, ensure that you have read all the prerequisites. For more information about the prerequisites to localizing the application, refer to the *Sterling Selling and Fulfillment Foundation: Localization Guide*.

Localizing Resource Bundles

A resource bundle is a file that comprises resource bundle keys and their corresponding values. The values pertaining to these resource bundle keys are translated as part of localization. Each field in the user interface (UI) has a key associated with it. To display the translated literal for a field in the UI, the localized value of the key associated with the field is fetched from the bundle file.

A property resource bundle is a collection of property files that are accessed as Java™ resources. In Sterling Configurator, the resource bundle is named as `package.properties`. The `package.properties` file is located under the `com/comergent/reference/apps/configurator` directory.

You can use the `package.properties` file to localize the following:

- Error and warning messages displayed on the user interface.
- Descriptions pertaining to labels, panels and headings displayed in the user interface.
- Sentences displayed in the user interface.
- Dynamic data within a literal. For example, the user interface may have to display a literal that informs a user that an input value cannot exceed a certain number of characters, and that the number of characters is dynamic. In this case, if the maximum character length for a description is set at 428 characters, 428 is the value of the parameter. The corresponding bundle entry is defined as `"b_MaxCharLengthExceeded":"The value cannot exceed {0} characters"`.

Note: As per localization specifications, translators can increase the width of values in the `package.properties` file by an additional 25% and expect the page to look reasonable. If the field length is increased beyond 25%, other JSP level customizations may be necessary.

The `package.properties` file is provided to a third party for translation. The `package.properties` file contains the keys required to translate the literals in the configuration page, and their language-specific translation. Duplicate keys can exist between the files, but you must ensure that there are no duplicate literals within a file.

The `package.properties` file contained in the `sic.jar` file located in the `<INSTALL_DIR>/repository/eardata/sic/war/WEB-INF/lib` folder is the base property file for the resource bundle. If the localized resource bundle file cannot be located, the system will use the base property file contained in the `sic.jar` file to display the literal on the user interface. The base property file contains the `en_US` (English-United States) mapping for each key. This file is named according to the resource bundle name with the `.properties` extension. For example, `com/comergent/reference/apps/configurator/package.properties` is the name of the base property file for the Sterling Configurator resource bundle. The translated property file has the same name, but the locale name is interposed between the resource bundle name and the `.properties` extension. Thus, the `package_fr.properties` file located under the `com/comergent/reference/apps/configurator` directory is the translated property file for the `fr_FR` (French_France) locale.

Sterling Configurator supports all the Java resource bundle localization techniques. Java resource bundle localization defines a locale by language, country or region. However, it is recommended that you localize a resource bundle file to the broadest category. For example, instead of creating a translation for French-France (`fr_FR`), create a translation for French (`fr`) only. This translation is equally applicable in France, Canada, Switzerland, Niger, Algeria, Mali, and so on, where French is spoken. For country or region specific terms that differ from this translation, you can create smaller translation files.

Localize Resource Bundles

About this task

To localize resource bundles, perform the following steps:

Procedure

1. Extract the package.properties file from the sic.jar file located in the <INSTALL_DIR>/repository/eardata/sic/war/WEB-INF/lib folder and save the file in a temporary folder.
2. The resource bundle contains a <key>=<value> pair, where key is the resource key and value is the literal displayed for the corresponding locale. Replace <value> with the translated value.
3. Save the modified file. If the file is in UTF-8 format, convert it to ASCII by running the native2ascii command as follows:
native2ascii -encoding UTF-8 <source file> <target file>
4. The translated file should be renamed in the following format:
package_<2 letter code for language as given by ISO 639>_<2 letter code for territory as given by ISO 3166>.properties
Here, <2 letter code for territory as given by ISO 3166> is required only if the translation is specific to a language and a country or region.
5. Create a JAR file of the files that have been translated.
6. Run the following command from the <INSTALL-DIR>/bin folder to install the JAR file:
For Windows:
install3rdParty.cmd
For Linux/UNIX:
install3rdParty.sh
For more information about installing third-party JAR files, refer to the *Sterling Selling and Fulfillment Foundation: Installation Guide*.
7. Rebuild the Sterling Selling and Fulfillment Foundation Enterprise ARchive (EAR).
For more information about localizing resource bundles, refer to the *Sterling Selling and Fulfillment Foundation: Localization Guide*.

Localizing Factory Setup Data

Besides storing your transactional data, the database also stores configuration data, such as error codes and item descriptions of various attributes. This means that the database may have to store values in a language-specific format. If these database literals are not localized, screen literals are displayed inconsistently, with some being displayed in the localized language, and others being displayed in English.

For a multilanguage installation, you can localize the database factory default values. For more information about localizing factory setup, refer to the *Sterling Selling and Fulfillment Foundation: Localization Guide*.

Chapter 5. Customizing Sterling Configurator: An Overview

The Sterling Configurator can be customized based on your business requirements. You can make changes in the way information is displayed in the item configuration page in the user interface (UI). For example, you can change the UI look-and-feel or hide certain items that belong to a model when a customer belonging to a particular customer group logs in and navigates to that model.

This topic provides an overview of the types of customization possible in the Sterling Configurator application.

The Sterling Configurator application uses the Apache Struts 2 framework for page construction and request management between pages. Sterling Configurator customizations can be performed by selectively overriding the action definitions. Struts 2 action definitions bind together a collection of resources required to fulfill any type of request from the Web. Custom action definitions can selectively overlay a portion of the action namespace.

The following components can be customized in Sterling Configurator:

- Controls - You can add or modify controls.
- Function Handlers - You can add new function handler classes.
- Struts extensions — Defining new struts and overriding the existing struts.

For more information about creating and extending a struts XML file, refer to the *Sterling Selling and Fulfillment Foundation: Customizing the Web UI Framework*.

- Mashup extensions — Customizing the input XML and the output template of an API call. Additionally, you can perform the following functions with mashups:
 - Define new mashups
 - Override the existing mashup using override extensibility
 - Extend the mashups using differential extensibility

For more information about how to override extensibility and extending mashups using differential extensibility, refer to the *Sterling Selling and Fulfillment Foundation: Customizing the Web UI Framework*.

Notes:

- You can extend the mashups specific to a storefront by defining the mashup in the customized struts action and including the corresponding mashup file to the customization folder created by you. However, you can extend mashups by following the approach specified in the *Sterling Selling and Fulfillment Foundation: Customizing the Web UI Framework*. For more information about extending mashups, refer to the *Sterling Selling and Fulfillment Foundation: Customizing the Web UI Framework*.
- Action class extensions — You can extend action classes. For more information about extending action classes, refer to the topic, “Customizing Action Classes”.
- JSP — Creating new JSPs and overriding the existing JSPs.
- Themes — CSS and image files pertaining to the application's look and feel are organized in a directory hierarchy, with the theme name as the root. By default, each storefront is assigned a theme. The CSS files pertaining to a theme are used in construction of pages.

For information about customization basics, refer to the *Sterling Selling and Fulfillment Foundation: Customization Basics*.

If you add a new field in the JSP, you may have to extend the database table to add a corresponding database column. In such a scenario, the database is extended to include a corresponding column. The new column that is added in the database must be exposed to the corresponding APIs. For more information about extending the database, refer to the *Sterling Selling and Fulfillment Foundation: Extending the Database*.

Chapter 6. Customize Actions

Action Definition Customization Checklist

About this task

When customizing the action definitions that are a part of Sterling Configurator, follow the sequence in which the tasks are listed in the following checklist.

Procedure

1. Create the root folder for the customization project. For more information about how you can structure the folders for customizations, see the topic, "Customization Examples".
2. Under the Customization project create a customization folder, say Cust1.
3. Determine the Struts file you want to customize. To identify the files that require customization, it is important to know the relevant resources such as the action name, mashup XML, and JSP file that are defined as part of the corresponding Struts file.
4. As part of customization, you can either introduce a new Struts file in the customization project or modify the existing Struts file by copying it to the customization project with a different name.
5. To introduce a new Struts file, perform the following steps:
 - a. Add a new action name.
 - b. Add a new JSP page.
 - c. Add a new mashup XML file.
 - d. Add a new XML binding file. The binding files bind the XAPI variables with the JSP components and JAVA attributes with the API attributes.
To modify the existing Struts file, perform the following steps:
 - e. Copy the existing action name to the customization project and modify it.
 - f. Copy the existing JSP page to the customization project and modify it.
 - g. Copy the existing mashup XML file to the customization project and modify it.
 - h. Copy the existing XML binding file to the customization project and modify it.
Based on the requirement, you must either add or update the appropriate files defined in the customized Struts file.
6. Include the newly added or modified Struts file into `custom_struts.xml` file placed under the customization project.
7. Include the `custom_struts.xml` file in to the `sic_struts.xml.sample` file located in the `<INSTALL_DIR>/repository/eardata/sic/extn` folder.
8. Rename the `sic_struts.xml.sample` file to `sic_struts.xml` file.
9. Build the JAR file of the customization project.
10. Install the JAR file using the `installService` utility.
11. Create the EAR and deploy the EAR.

Customize Action Classes

About this task

The Sterling Configurator application enables you to add new action classes. You may need to add new action classes in certain customization scenarios such as adding a new functionality which involves a complex sequence of mashup calls. The Sterling Configurator class files are located in the `<INSTALL-DIR>/repository/eardata/sic/war/WEB-INF/classes` directory.

To add a new action class:

Procedure

1. Create a new class file for the customization, and define the class as part of a package. It is recommended that you include an indication of the enterprise for which the customization is being performed in the new class. For example, if the enterprise is ABC, the class may be defined as `"com.abc.webchannel.orders.OrderChangeAction"`.
2. The new class should extend the `"com.sterlingcommerce.webchannel.core.WCMashupAction"` action.
3. Map the action to the new action class in the `custom_struts.xml` file by following similar steps as described in the section, "Overriding an Existing Functionality".

Chapter 7. Customize Themes

Theme Customization Checklist

Procedure

1. Create the root folder for the customization project. For more information about how you can structure the folders for customizations, see the topic, "Customization Examples".
2. Under the customization project, create a customization folder, say Cust1.
3. Create a new theme, say, theme1 or assign an existing theme to the enterprise that must be customized.
4. Add the customized CSS, js, and image files to the customization folder. Ensure that the files are placed in the appropriate folder. For example, the CSS file must be placed in the customization/src/main/webapp/sic/css/theme folder. For more information about how you must structure the folders for customizations, see the topic, "Customization Examples".
5. Build the JAR file of the customization project.
6. Install the JAR file using the installService utility.
7. Create the EAR and deploy the EAR.

Customization examples

This topic provides a few examples of customization to provide a better understanding of the customization process. Apart from the examples provided here, you can perform customizations based on your requirements.

Theme Customization for a Storefront

About this task

Note: The theme name assigned to a storefront and the storefront ID are case-sensitive.

Alternatively, you can customize the theme for a storefront by adding the appropriate files in the customization project by performing the steps mentioned below:

Procedure

1. Create the root folder for customization say, customization.
2. The name of the customized theme can be abc_cust.
3. In the following folders, replace the {custom_theme} attribute with the storefront theme name:
 - customization/src/main/webapp/sic/css/theme/{custom_theme}
 - customization/src/main/webapp/sic/js/theme/{custom_theme}
 - customization/src/main/webapp/sic/images/theme/{custom_theme}

The following table lists the files and the corresponding location required for implementing the theme customization:

File Name	Location
-----------	----------

theme-1.css

customization/src/main/webapp/swc/css/theme/abc_cust

theme.js

customization/src/main/webapp/swc/js/theme/abc_cust

All the images pertaining to the theme are stored in the customization/src/main/webapp/swc/images/theme/abc_cust folder.

4. Ensure that you deploy the customized JAR by performing the steps mentioned in the “Postcustomization Deployment” topic.

Chapter 8. Customize Controls

Controls are used to determine how the option classes and option items are displayed and behave in the user interface (UI). You can modify an existing control or add a new control.

In the base model, the Sterling Configurator Visual Modeler supports the following choice of controls:

- Radio button
- Checkbox
- Drop down list
- Listbox
- Multiple Selection Listbox
- Display All Children
- User Entered Values
- Tabular Display

When Modelers are creating the model for configurations, they determine which control is used for an option class by selecting it from the Control drop-down list on the Display tab of the option class detail page.

Each control corresponds to a JSP page and the behavior of the option items. This correspondence is defined in the `controls.properties` configuration file under the `Comergent/WEB-INF/properties` folder in the deployment directory.

Following is a sample entry defined in the `controls.properties` file:

```
RADIO.name=Radio Button  
RADIO.jsp=controls/radio.jsp  
RADIO.behavior=single
```

In this example, for the radio button control, the `radio.jsp` JSP page is used to render the option class in the UI. The `behavior` property determines how the Sterling Configurator handles picks in this control. Based on how the `behavior` property is defined, the Sterling Configurator handles picks as follows:

- `entry` — used for user-entered controls.
- `expand` — expand all the children of this control if the control itself is picked.
- `multiple` — allow one or more option items to be picked from this control.
- `single` — if an option item is picked, then remove any previous picks from this option class.

Customizing an Existing Control

You can customize an existing control by modifying the corresponding JSP page or by creating a new JSP page and modifying the `control.properties` file to point to the new JSP page.

Create a new control

About this task

You can define a new control by adding the name of the control to the list of controls declared.

For example, to add a MATRIX_CUSTOM control:

```
controls=MATRIX_CUSTOM,RADIO,CHECKBOX,COMBOBOX,LISTBOX,MULTISELLISTBOX,ALLPICKED,UEV,DISPLAY
```

Then declare the properties of the new control as follows:

```
MATRIX_CUSTOM.name=Matrix Custom Control  
MATRIX_CUSTOM.jsp=controls/MatrixCustom.jsp  
MATRIX_CUSTOM.behavior=single
```

Customizing and modifying controls does not require a server restart because this file is read each time a Sterling Configurator Visual Modeler or Sterling Configurator session is launched.

To customize controls, perform the following steps:

Procedure

1. If you have already customized the controls in the Sterling Configurator Visual Modeler, save the customized `controls.properties` file in the directory where the properties are stored. The path to the location where the properties are stored is configured in the Applications Manager.
2. Edit the `controls.properties` file, as required. For more information about modifying the `controls.properties` file, refer to the *Sterling Configurator Visual Modeler: Implementation Guide*.
3. Save the customized JSP files in the `<INSTALL_DIR>/repository/eardata/sic/extn` directory.

Chapter 9. Customize Control Handlers

Control handlers are a mechanism for invoking Java code to handle special actions that may be difficult to handle in a JSP page alone. For example, the `DynamicInstantiationControlHandler` class dynamically adds child option items to a model when it is retrieved from the cache and removes them (the dynamic items) when the model is returned to the model cache.

The control handlers must implement the `IControlHandler` interface (typically they extend the `StandardControlHandler` class). They implement (or override the implementation of a base class) the methods:

- `public void initializeControl(IModelBean model, IOptionClassBeanoptionClass)`
- `public void resetControl(IModelBean model, IOptionClassBeanoptionClass)`
- `public void handleComergentRequest(IModelBeanmodel,ComergentRequest request,Map picks)`

The ¹ method is called just after the model is fetched from the cache. The ² method is called just before the model is returned to the cache. The³ method is called to construct the picks map used to apply picks.

1. `initializeControl()`
2. `resetControl()`
3. `handleComergentRequest()`

Chapter 10. Customize Function Handlers

About this task

The Sterling Configurator application enables you to customize function handler classes. The function handler classes are Java classes that are used to define custom functions that can be invoked by the Sterling Configurator rule engine.

The function handlers are defined in the `functionHandlers.properties` configuration file inside the `cmgt-configurator.jar` file located under the `<INSTALL-DIR>/jar/smcfs/9.1` directory. This file includes a name for each function handler and the directory in which the function handler class is located.

Following is a sample fragment of the `functionHandlers.properties` file:

```
WEB-INF/classes/com/comergent/apps/configurator/functionHandlers=
CheckLookupFunctionHandler,ChildSum,CountFunctionHandler,IsSelectedHandler,
LengthFunctionHandler,ListFunctionHandler,LookupFunctionHandler,
MaxFunctionHandler,MinFunctionHandler,ParentFunctionHandler,
PropValHandler,SumFunctionHandler,ValueFunctionHandler,WebServiceLookup
CheckLookupFunctionHandler=
com.comergent.apps.configurator.function-Handlers.CheckLookupFunctionHandler
```

To customize the function handler class in the Sterling Selling and Fulfillment Foundation, perform the following steps:

Procedure

1. Create a new Java class with the `com.comergent.apps.configurator.functionHandlers` package declaration. The class declaration must declare that the class extends the `AbstractRuleFunctionHandler` class.

The new Java class should implement the following methods:

- `public String getFuncName():` return the function name, such as "sum" or "max". This is case-sensitive: you can use different function handlers to manage "sum" and "SUM".
 - `public int getType():` return the type of value returned by the function. This should be a constant defined in the `com.comergent.api.appsservices.rulesEngine.Value` class. The `AbstractRuleFunctionHandler` class method returns `Value.STRING`. Therefore, you must override this method if the function returns any other type.
 - `public Value handle(State state, String prop):` return the `Value` calculated for the function.
 - `public boolean isPublicHandler():` return true if the function handler may be used by any client application; otherwise return false. The `AbstractRuleFunctionHandler` class method returns true. Therefore, you must only override this method if the function handler is private.
2. Create a new JAR file containing the customized function handler class and `functionHandlers.properties` file.

Note: The JAR file must be present in the application classpath of the Sterling Selling and Fulfillment Foundation. The `functionHandlers.properties` file is instantiated by a servlet. Therefore, the JAR file must be added to the root of the EAR.

For example, create a new `sic_customization.jar` file containing the following files:

- `com/comergent/apps/configurator/functionHandler/AvgFunctionHandler.class`
 - `com/cust/functionHandlers.properties`
3. Modify the `web.xml` file of the application that is using Sterling Configurator to provide the relative path and name of the customized `functionHandlers.properties` file.

Following is a sample `web.xml` file:

```
<servlet id="Servlet_1">
    <servlet-name>InitializationServlet</servlet-name>
<servlet-class>com.comergent.reference.apps.configurator.InitializationServlet
</servlet-class>
    <init-param>
        <param-name>ObjectMap</param-name>
        <param-value>ObjectMap.properties</param-value>
    </init-param>
    <init-param>
        <param-name>EhCache</param-name>
        <param-value>EhCache.xml</param-value>
    </init-param>
    <init-param>
        <param-name>CacheTypes</param-name>
        <param-value>cacheTypes.properties</param-value>
    </init-param>
    <init-param>
        <param-name>FunctionHandler</param-name>
        <param-value>
            /com/<customization_folder>/functionHandlers.properties
        </param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
```

Function Handler Example

The following example of function handler class implements the “max” function:

```
package com.comergent.apps.configurator.functionHandlers;
import com.comergent.api.appservices.rulesEngine.*;
import com.comergent.apps.configurator.model.*;
import java.util.*;
/**
 * Handles the logics of <i>Max</i> function for a <code>Property
 * </ code>, given the <code>State</code>.
 *
 * @author Comergent Technologies
 * @version 1.0
 *
 * @see Value
 * @see Property
 * @see State
 */
public class MaxFunctionHandler extends AbstractRuleFunctionHandler
{
    /**
     * Name of the function, this particular handler serves.
     */
    private static final String m_name = "max"/*I18NOK:23c81106*/;
    /**
     * Return the name of the function this handler supports
     * @return the function name
     */
    public String getFuncName()
```

```

{
return m_name;
}
/**
 * Return the value type this particular function handler
 * returns.
 * Returns <code>Value.NUMERIC</code>, as the type.
 * @return the numeric value.
 * @see Value the container for different types.
 */
public int getType()
{
return IValue.NUMERIC;
}
/**
 * Return the <i>Maximum</i> value assigned to the property,
 * given <code>State</code>.
 * <code>Value</code> is returned as a result.
 * Extracts all the matching properties given the name, and
 * sorts them and extracts
 * the maximum value.
 *
 * Returns <i>null</i>, if the requested <code>Property</code>
 * does not exist.
 *
 * @param state the property pool
 * @param prop the property to evaluate the function.
 * @return Value the <code>Property</code>, that contains
 * the maximum value.
 */
public IValue handle(IState state, String prop)
{
//double max = 0;
double <] propList = state.getMatchingNumericProperties(prop);
if (propList <= null)
{
Arrays.sort(propList);
return new ConfigValue(new Double(
propList<propList.length -1]),IValue.NUMERIC);
}
return null;
}
}}

```

In this example, the⁴method calculates the maximum value of a property by sorting the list of property values and then returns the last value in the sorted array. The function returns a number of type `IValue.NUMERIC`. It is a public function handler.

Web Service Function Handlers

You can write function handlers that invoke Web services. These classes should extend the `com.comergent.api.apps.configurator.IConfigWebService` interface. They make use of the `webServiceLookup.properties` configuration file: this file specifies how the handler should invoke the Web service. These fragment functions reference Web services:

⁵: this determines whether the correct properties exist to call the Web service

4. handle()

5. checkwslookup()

⁶: invokes the Web service An example function handler is provided by `com.comergent.reference.apps.configurator.SampleWebService`.

6. wslookup()

Chapter 11. Data Validation

Data Validation

The Sterling Configurator application provides the data validation functionality for validating and sanitizing request inputs and outputs. You can use the data validation functionality to, explicitly define the characteristics in the input and output requests, and drop all the other data. You can define your own validation rules for validating different request parameters. You can also encode the data before sending it back to the user interface.

Data validation or sanitization can be performed for various kinds of inputs such as parameter name, parameter value, cookie name, cookie value, and so on. Sterling Configurator supports regular expression based validation.

Input Validator

The Sterling Selling and Fulfillment Foundation Input Validator finds all the validation rules that are registered for a particular input, and invokes the validation. The Sterling Configurator validator is called by a request wrapper to validate request inputs.

Validation Rule

A validation rule performs validation and sanitization of the input. A validation rule contains a property as input identifier for which validation has to happen. A validation is invoked whenever the corresponding input request is accessed. A validation rule must specify the name of the input, it has to validate. For example, if you want to validate the value of a particular parameter, the validation rule must specify the name of that parameter. Multiple inputs with the same name can exist. All the validation rules must be registered with the Input Validator in order to validate the corresponding input.

Note: No validation rules are defined for a given input. The default validation rules specified earlier will be used to validate the input.

You can define the following types of validation rules:

- **Regular Expression-Based Validation Rule**—This type of validation rule is designed to perform regular expression-based validations. This validation rule type supports multiple whitelist and blacklist regular expressions.
- **Java-Based Validation Rule**—This type of validation rule is designed to perform Java-based validation and sanitization of inputs. This validation rule type validates an input and then calls the `getValidInput()` method of the implementation class.

Disabling Data Validation

By default, data validation that is enabled on all input requests will be validated against the registered validation rules. You can disable the data validation on input requests by adding a context parameter in the module configuration file. To disable data validation, in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder add an entry for the `context-param` element as follows:

```

<context-param>
  <param-name>scui-suppress-request-validation</param-name>
  <param-value>TRUE</param-value>
</context-param>

```

Bypassing Data Validation for an URI

By default, data validation is enabled on all input requests. You can, however, bypass data validation on input requests for some specific universal resource indicators (URIs) by adding bypass URI as context parameters in the module configuration file.

To bypass data validation, in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder add an entry for the `config-param` element for each such URI, for example:

```

<context-param>
  <param-name>request.validation.bypass.uri.1</param-name>
  <param-value>/console/login.jsp</param-value>
</context-param>
<context-param>
  <param-name>request.validation.bypass.uri.2</param-name>
  <param-value>/console/start.jsp</param-value>
</context-param>
<context-param>
  <param-name>request.validation.bypass.uri.endswith.1</param-name>
  <param-value>.js</param-value>
</context-param>
<context-param>
  <param-name> request.validation.bypass.uri.regex.1</param-name>
  <param-value>^.*test.jsp$</param-value>
</context-param>

```

These context parameters can have names starting with `request.validation.bypass.uri`, or `request.validation.bypass.uri.endswith`, or `request.validation.bypass.uri.regex`, as described in the following list. You can define multiple entries for these context parameters.

- `request.validation.bypass.uri`—Any request with an URI that is the same as the value specified in the `param-value` element of the context parameter will be bypassed and not validated.
- `request.validation.bypass.uri.endswith`—Any request with an URI that ends with the value specified in the `param-value` element of the context parameter will be bypassed and not validated.
- `request.validation.bypass.uri.regex`—Any URI request that matches the regular expression, as specified in the `param-value` element of the context parameter, will be bypassed and not validated.

Implementing Data Validation

To implement data validation, perform the following tasks:

- Create a Validation Rule
- Register a Validation Rule
- Use an URI-Based Validation Rule
- Defining an Adapter to Find Validation Rules
- Delete the Registered Validation Rules

Create a Validation Rule

A validation rule performs validation and sanitization of the input. A validation rule contains a property as input identifier for which validation has to be performed. Validation is performed whenever a specified input is accessed.

To create a validation rule, implement the `ISCVValidationRule` interface and implement the following interface methods:

- `getName()`—Returns the name of the input for which this validation rule should be applied.
- `isValid(String context, String input)`—should do the actual validation
- `getSafe(String context, String input)`—should return the safe output after removing all the unwanted characters from the input.
- `getTypeName()`

You can use `_global_` and `_default_` as names for the global and default rules respectively.

Sterling Configurator provides two implementation classes of the `ISCVValidationRule` interface. These implementation classes validate input based on regular expressions or Java calls. You can use these implementation classes based on your requirement.

SCRegexValidationRule Implementation Class

This implementation class of the `ISCVValidationRule` interface helps in validating an inputs based on regular expression whitelists and blacklists. You can use this class for creating validation rules that are based on regular expressions.

To create a validation rule using this implementation class:

1. Create an instance of the `SCRegexValidationRule` class. For example:

```
ISCVValidationRule SCRegexValidationRule newValRule =  
new SCRegexValidationRule();
```

2. Set the name of the validation rule as follows:

- If you are defining the validation rule for a parameter value, set the name of the validation rule as parameter name. For example:
`newValRule.setName("<parameter_name>");`
- If you are defining the validation rule for a parameter name, set the name of the validation rule as follows:
`newValRule.setName(newValRule.RULE_NAME_GLOBAL);`

Note: You can use `_global_` and `_default_` as names for the global and default rules respectively.

1. Add the whitelist pattern of regular expression against which you want to validate the parameter name. For example:

```
newValRule.addWhitelistPattern("[a-zA-Z0-9.\\-\\+/_+=_ :]*$");
```

2. Set the `AllowNull` value. For example:

```
newValRule.setAllowNull(true);
```

3. Register this new validation rule with the Input Validator. For more information about registering validation rules, refer to the topic, Register a Validation Rule.

SCJavaValidationRule Implementation Class

This abstract class of the `ISCVValidationRule` interface helps in validating an input based on the Java call. You can implement this abstract class for creating a validation rule that requires a Java call to validate inputs.

To create a validation rule using this abstract class, implement this abstract class and the `getValidInput()` and `getSafe()` methods of this abstract class.

Register a Validation Rule

In order to be able to validate inputs, you must register the validation rules that you created in the Input Validator.

Note: Global rules are applied on all inputs for an `inputType`. Default rules are applied on an `inputType` if no other rules either specific or global are found for an `inputType`. If invalid input is found, either an `SCValidationException` is thrown, or the exception is added to the passed `SCValidationErrorList`. For more information about exception handling during data validation, refer to the topic, [Exception Handling](#).

To register a validation rule:

1. Get the Application ID:

- If you are using an application that is based on the Web UI framework, use for the `getApplicationId()` method of the `SCUIUtils` class. For example:

```
String appId = SCUIUtils.getApplicationId(config.getServletConext());
```

If you are using an application that is based on the HTML UI framework, use the `getConsoleApplicationId()` method of the `YFCCContextParams` class. For example:

```
String appId =  
YFCCContextParams.getInstance(config).getConsoleApplicationId()
```

Note: Ensure that the same application ID is defined as context parameter in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder. For example:

```
<context-param>    <param-name>console-application-id</param-name>  
    <param-value><Application ID></param-value> </context-param>
```

- Get an instance of the `SCValidator` using the `getInstanceFor()` method with the Application ID. For example:

```
SCValidator validator = SCValidator.getInstanceFor(appId);
```

2. Register the validation rule with the `SCValidator` instance that you created.

- If you are defining the validation rule for a parameter value, use the `INPUT_TYPE_PARAMETER_VALUE` constant. For example:

```
validator.addRule(new ValRule,  
SCUIWebValidationConstants.INPUT_TYPE_PARAMETER_VALUE);
```

- If you are registering the validation rule for a parameter name, use the `INPUT_TYPE_PARAMETER_NAME` constant. For example:

```
validator.addRule(new ValRule,  
SCUIWebValidationConstants.INPUT_TYPE_PARAMETER_NAME);
```

Note: When you register the validation rules for inputs, the following `inputTypes` constants must be used:

- `INPUT_TYPE_PARAMETER_VALUE`

- INPUT_TYPE_PARAMETER_NAME
- INPUT_TYPE_COOKIE_VALUE
- INPUT_TYPE_COOKIE_NAME
- INPUT_TYPE_HEADER_VALUE
- INPUT_TYPE_HEADER_NAME
- INPUT_TYPE_SCHEME
- INPUT_TYPE_SERVER_NAME
- INPUT_TYPE_CONTEXT_PATH
- INPUT_TYPE_PATH
- INPUT_TYPE_QUERY_STRING
- INPUT_TYPE_URI
- INPUT_TYPE_URL
- INPUT_TYPE_JSESSIONID
- INPUT_TYPE_SERVLET_PATH

Registering a Validation Rule for the Parameter Value Input Using datatypes.xml

This is an optional task. You can also register a validation rule using the datatypes file. This method of registering a validation rule can only be used for parameter value inputs. The datatype for a parameter is deduced using the datatypes map, and the parameter value is validated by using the validation rules registered against that datatype.

Applications based on the HTML UI framework can register a regular expression-based or Java-based validation rule in the datatypes.xml file in the following manner:

```
<DataType Name="Address" Size="70" Type="NVARCHAR">
  <UIType Size="30" UITableSize="30"/>
  <Validation>
    <Regex JavaPattern="<pattern>" JSPattern="<pattern>"/>
    <Impl JavaClass="com.sterlingcommerce.test.MyRuleClass"
      JSFunctionName="myJavascriptFunction"/>
  </Validation>
</DataType>
```

By default, for a <Regex> element, the maximum size of the validation rule is set to the size of the datatype. You can override the maximum size of the validation rule by using the MaxLength attribute. Also, you can set the minimum size of the validation rule by using the MinLength attribute. For example:

```
<DataType Size="5" Name="Foo" Type="Bar">
  <Validation>
    <Regex MaxLength="200" MinLength="3"
      JavaPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"
      JSPattern="^[a-zA-Z0-9.,!\-/+=_ :]*$"/>
  </Validation>
</DataType>
```

Note: A Java-based validation rule class must have a fully qualified class name, and this class must implement the ISCValidationRule interface.

In the datatypes.xml file, you can also define Javascript patterns and functions to validate the input on the client itself. These client side validations will be fetched on the client and all the corresponding inputs will be validated against these client side validations.

Use an URI-Based Validation Rule

An URI-based validation rule can be used to suppress certain rules for certain input requests. To use an URI-based validation rule, perform the following tasks:

1. Create a validation rule. For more information about creating a validation rule, refer to the topic [Create a Validation Rule](#).
2. Create an instance of the `ISCValidationRuleKey` using the `SCUIURIContextValidationRuleKey(String uri, String name)` class. For example:
`SCUIURIContextValidationRuleKey key =`

```
new SCUIURIContextValidationRuleKey(uri, name);
```

Here, URI refers to the input path on which the rule should be applied. The URI should not contain the context path and should start with "/". name is the name of the input that has to be validated. You can use `ISCValidationRule.RULE_NAME_GLOBAL` and `ISCValidationRule.RULE_NAME_DEFAULT` as names to make a rule either a global rule or a default rule respectively.

3. Register the validation rule that you created along with the validation rule key. For example:

```
SCValidator.getInstanceFor(SCUIWebValidationUtils.  
getApplicationID(servletContext)).addRule(key, rule,  
SCUIWebValidationConstants.INPUT_TYPE_PARAMETER_VALUE);
```

Defining an Adapter to Find Validation Rules

This is an optional task. You can define an adapter to find the validation rules that will be used to validate the parameter values. This adapter class must implement the `ISCUInputValidationAdapter` interface, and must be registered with the application as a context parameter. For example:

```
<context-param>  
  <param-name>scui-param-value-validation-adapter</param-name>  
  <param-value>test.MyParamValueValidationAdapter</param-value>  
</context-param>
```

You must implement the `getValidationRules()` method of the `ISCUInputValidationAdapter` interface and pass the parameter name in the name argument.

When validating a parameter value, the system will call the registered adapter to find the rules against which the parameter value should be validated. The `getValidationRules()` method can either return all the rules registered for the passed parameter name, or have some logic to find other rules too. If no adapter is registered, the system will use all the rules registered for the given parameter name, along with the global rules or the default rules, to validate the parameter value.

Delete the Registered Validation Rules

You can delete the registered validation rules by calling any of the following methods:

- `removeDefaultRules(String inputType)`
- `removeGlobalRules(String inputType)`
- `removeRules(String name, String inputType)`
- `removeRules(ISCValidationRuleKey name, String inputType)`

Exception Handling

While validating a request, if an invalid input is found, an `SCUIRequestValidationException` is thrown. You can override this default behavior by adding the `scui-suppress-validation-exception` context parameter with the value as `TRUE` in the `web.xml` file located in your `EARFILE/WARFILE/WEB-INF` folder. For example:

```
<context-param>
  <param-name>scui-suppress-validation-exception</param-name>
  <param-value>TRUE</param-value>
</context-param>
```

When you set this parameter's value as `TRUE`, all the validation exceptions are added to a list, the list can be accessed by running:

```
ArrayList< SCUIRequestValidationException>
SCUIWebValidationUtils.getValidaionErrorList(HttpServletRequest request)
```

You can also define a global exception handler. If any validation exception has not been detected, and it goes back to the `SCUISafeRequestFilter`, the request is sent to the corresponding global error handler servlet container.

This global exception handler and the request method can be defined as context parameters in the `web.xml` file, located in your `EARFILE/WARFILE/WEB-INF` folder. For example:

```
<context-param>
  <param-name>scui-global-validation-exception-handler-path</param-name>
  <param-value><path_to_global_exception_handler></param-value>
</context-param>
<context-param>
  <param-name>scui-global-validation-exception-handler-method</param-name>
  <param-value>FORWARD|INCLUDE|REDIRECT</param-value>
</context-param>
```

For applications that are based on the Web UI framework, Sterling Configurator provides `/jsps/datavalidationerror.jsp` as the default exception handler.

For applications that are based on the HTML UI framework, Sterling Configurator provides `/sic/sic_app/jsp/datavalidationerror.jsp` as the default exception handler.

The Web UI framework has also added a Struts action result, "DATAVALIDATIONERROR", which is returned to the exception handler if the request is invalid. You can define this result type and the corresponding path, say, `/jsps/datavalidationerror.jsp`, for the Struts actions.

By default, the global exception handler method is set to `FORWARD`.

Default Validation Rules in Sterling Configurator

By default, Sterling Configurator provides and registers validation rules for specific parameters and rules pertaining to requests, based on specific inputs.

To validate the request inputs (such as parameter value, parameter name, etc.), the input validator uses following regular expressions:

- `HTTPScheme=^(http|https)$`
- `HTTPServerName=^[a-zA-Z0-9_\\.\\-]*$`

- HTTPParameterName=`^[a-zA-Z0-9\\-\\.]*$`
- HTTPParameterValue=`^[a-zA-Z0-9.!\\-\\/+=_]*$`
- HTTPCookieName=`^[a-zA-Z0-9\\-\\.]*$`
- HTTPCookieValue=`^[a-zA-Z0-9.!\\-\\/+=_:]*$`
- HTTPHeaderName=`^[a-zA-Z0-9\\-_*]*$`
- HTTPHeaderValue=`^[a-zA-Z0-9()\\-\\.*\\.\\?;+\\/:&_"]*$`
- HTTPContextPath=`^[a-zA-Z0-9\\.\\-_/]*$`
- HTTPPath=`^[a-zA-Z0-9\\.\\-\\/_!]*$`
- HTTPURI=`^[a-zA-Z0-9()\\-\\.*\\.\\?;+\\/:&_!\\$]*$`
- HTTPURL=`^.*$`
- HTTPSESSIONID=`^[a-zA-Z0-9!:\\-]*$`
- HTTPQueryString=The HTTPQueryString will be validated based on the individual parameter names and values.

These validation rules are invoked for all the inputs of the same kind. For example, all HTTP Header names are validated against "HTTPHeaderName" regular expression.

Sterling Configurator also overrides the above validation rules for specific parameters and specific input-types by providing entries in the following property files:

- `sic_ParamValue_ValidationRules.properties`: This file is located in the `EARFILE/sic.jar/com/comergent/security/xss` folder. It contains validation rules that are specific to the request-input HTTPParameterValue for specific request parameter.
- `sic_InputType_ValidationRules.properties`: This file is located in the `EARFILE/sic.jar/com/comergent/security/xss` folder. It contains validation rules that are specific to any of the request inputs.
- `sicapp_ParamValue_ValidationRules.properties`: This file is located in the `EARFILE/sic_app.jar/com/sterlingcommerce/webchannel/core/security/xss` folder. It contains validation rules that are specific to the request-input HTTPParameterValue for specific request parameter.
- `sicapp_InputType_ValidationRules.properties`: This file is located in the `EARFILE/sic_app.jar/com/sterlingcommerce/webchannel/core/security/xss` folder. It contains validation rules that are specific to any of the request inputs.

Data Validation Failure

Data validation fails can fail if a customer provides invalid input. In such a scenario, Sterling Configurator enables a customer to identify it by the way of displaying appropriate error message.

For example, if a customer enters `<script>` as one of the basic search criteria in the Sterling Configurator Home page, the basic search field is validated. In such a scenario, the input validation fails because `<script>` is not a valid input. The details pertaining to validation failure are logged in the server log file. A customer can view the server log file to understand the cause of validation failure, and verify whether it was an authentic validation failure. If required, a user can create validation rules based on the customer's requirements.

The following is an example of an error trace generated in the server log:


```
3415041 [qtp0-5] WARN DataValidationLogger - SECURITY-FAILURE -
  Input does not conform to pattern:
context=HTTP parameter value (Name:searchTerm),
pattern=[a-zA-Z0-9\:\.\-\/\+=_\@\x11\x12\x13\x14\=&\!?\;"'(\)\*\'\[\] ]*$,
Input=<script/>
  ValidationException @ com.sterlingcommerce.security.dv.SCValidationException.
<init>(SCValidationException.java:76)
```

This is a minimal response page with the 500 response status and Multipurpose Internet Mail Extensions (MIME) type set to either text or plain. Because the response status is sent as 500 and the MIME type is set as either text or plain, some versions of the Internet Explorer browser may replace this error page with the browser's custom error page. This is because of the preferences that have been set using **Internet Options > Advanced** tab to display user friendly HTTP error messages.

Note: By default, Sterling Configurator provides validation rules that are specific to en_US locale. To support validation rules that are specific to a different locale, the corresponding user must customize the validation rules.

Customize Data Validation

About this task

You can customize data validation rules by creating new validation rules and deleting existing validation rules. Ensure that data validation is enabled for Sterling Web in the web.xml file located in the EARFILE/WARFILE/WEB-INF folder.

To customize data validation rules:

Procedure

1. Include the following property files in the class path of the application server. These files are an extension to the property files provided by Sterling Web:
 - swc_InputType_ValidationRules_extn.properties
 - swc_ParamValue_ValidationRules_extn.propertiesFor more information about the property files with default validation rules provided by Sterling Web, refer to the *Sterling Web: Implementation Guide*
2. Define customized validation rules by providing entries in the property files mentioned previously. For information about implementing data validations, refer to the *Sterling Web: Implementation Guide*.
3. Restart the application server.

Note: By default, Sterling Web provides validation rules specific to the en_US locale. If you want to support a different locale, you must add characters pertaining to that locale. Validation rules are not specific to a locale and are generic and common across all the locales.

Chapter 12. Deploying your customizations

About this task

After the customizations are performed, you must deploy the changes in the application by performing the following steps:

- Generate the customization JAR file.
- Install the customization JAR file using the installService utility. In the customization JAR file, ensure that the customized files have been placed in the appropriate locations, as described in the tables below.
- Deploy the customizations by creating and deploying the SWC EAR file.

For example, if you are adding or overriding a functionality, the customized files must be placed in the appropriate locations as mentioned below:

Files Location

3rd party jar files

The path generated by install3rdparty utility. For more information about the install3rdparty utility, refer to the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

Java Server Page

`\sic\files\repository\eardata\sic\war\`

Javascript

`\sic\files\repository\eardata\sic\war\`

Mashup definitions

`\sic\files\repository\eardata\sic\war\mashupxmls\webchannel\`

Static image files

`\sic\files\repository\eardata\sic\war\`

If you are customizing the theme and logo, the customized files must be placed in the appropriate locations as mentioned below:

Files Location

Image for logo

`\sic\files\repository\eardata\sic\war\swc\images\logo`

CSS files

`\sic\files\repository\eardata\sic\war\sic\css\theme`

JS files

`\sic\files\repository\eardata\sic\war\sic\js\theme`

The DCL.xml file is the common file required for all customizations. This file is included in the `\swc\jars\` path in the customization JAR. The DCL.xml file must have an entry of the JAR file comprising the customized Java classes and resources. The name of the JAR files is case-sensitive. For example, the customization JAR file generated for the storefront ABC will be `ABC_CUST.jar` and the jar file comprising the customized Java classes and resources will be `abc_cust.jar`. You must generate the `abc_cust.jar` file and add it to the `\swc\jars\abc_cust\ path in the ABC_CUST.jar file.`

In the above example, the DCL.xml file has the entry for the abc_cust.jar as follows:

```
<dc1>
<vendor>
<name>abc_cust/<user_defined_version>/abc_cust.jar</name>
<target>DCL|APP</target>
</vendor>
</dc1>
```

To deploy the customizations to the application, you must perform the following steps:

Procedure

1. To generate the customization jar file, run the following command from the path to the directory where the customization script file is located. Ensure that the customization jar file comprises the DCL.xml file:

```
ant -f <customization script file> <target>
```

2. To install the customization jar file, run the following command from the <INSTALL_DIR>/bin folder:

For Windows:

```
InstallService.cmd <location of customized jar>
```

For Linux or Unix:

```
InstallService.sh <location of customized jar>
```

3. Navigate to the <INSTALL_DIR>/repository/eardata/{custname}/extn folder and rename the struts.xml.sample file as struts.xml.
4. Add the entry of the custom_struts.xml in the struts.xml file. This must be done for all the new customized action definitions.

```
<struts>
<include file="sic_struts.xml"/>
<include file="scuiimpl_struts.xml"/>
...
<include file="custom_struts.xml"/>
...
</struts>
```

5. Build and deploy the SWC EAR file. For more information about deploying EAR and WAR files, refer the *Sterling Selling and Fulfillment Foundation: Installation Guide*.

Chapter 13. Sterling Configurator Best Practices

Mapping eBusiness requirements to the features provided by Sterling Configurator can be daunting. Understanding how Sterling Configurator works in a variety of circumstances and studying examples demonstrating common usage patterns can help.

About Absolute and Relative Paths

This chapter refers to the use of absolute or relative paths to specify entities such as properties and rules. Paths have the following form: <model group root node>.<path to the option item that has the property or rule>.<property name or rule name>

For example, consider the following absolute path to the property `memoryProvided`:

```
MXDS-7500.memory.sim256.memoryProvided
```

The model group's root node is `MXDS-7500`. The path to the option item that has the property `memoryProvided` is `memory.sim256` and `memoryProvided` is the property name.

If you plan to use a property or rule in more than one model, you can use special symbols to specify relative paths. For example, the "*" in the following path indicates that the path begins at the root of the model group hierarchy:

```
*.memory.sim256.memoryProvided
```

Beginning the path with a period (.) indicates "from the attachment point of the rule". For example, the "." in the following path indicates that "an option item called `sim256` in an option class called `memory` in the current model".

```
.memory.sim256
```

Model Planning Considerations

A model represents a configurable item. When you sit down to plan a Sterling Web implementation, you start by considering how to design a model of the item. There is no one "right" way to model a particular item. On the other hand, there are an endless number of ways to create models that, while technically correct, are inefficient and hard to maintain.

The following are among the trade-offs to keep in mind:

- **Cost factors:** creation, maintenance, and performance. You must balance the cost of creating the model, compared with the cost of maintaining it, compared with its expected performance. The cost of creating the model represents the one-time effort expended to develop it; the cost of maintaining the model represents the effort expended over time to maintain and enhance it, while the performance represents the execution speed of the model on a particular hardware platform. You can optimize for any one of these factors, but be mindful that trying to optimize for more than one of them means, in most cases, that you have

competing goals. For example, a complex model might run fast, but would be hard to maintain. For an overview of models and best practice design principles, refer to the "Designing Models" topic.

- **Implementers' roles:** model builder only, model builder and maintainer, model maintainer only. The implementer's role influences their bias. For example, a consultant given a month to implement a model which will then be handed off to another group for maintenance may focus on designing the model quickly, rather than designing a model that will be easy to maintain. If the implementer will also maintain the model, the model may take longer to design and perhaps not run as fast, but will be easy to maintain long-term. Whatever your role, your goal should be to set up a model that will avoid problems down the road.

Designing Models

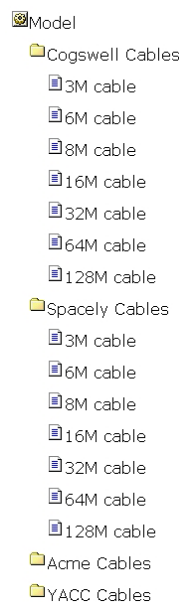
This section covers model design considerations and provides examples of designs that work, compared with designs that work best.

Make Models as Small and Simple as Possible

Model size matters. Large models take longer to render in the browser, are harder to maintain, and during configuration the Sterling ConfiguratorConfigurator "walks" the model structure a number of times to get prices, fire rules, and so on. Keeping the model size as small as possible through the use of subassemblies and other techniques described in this chapter improves performance and decreases maintenance costs.

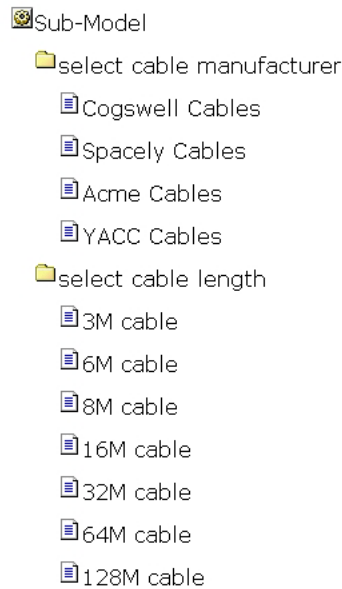
For example, suppose that you have a number of cable suppliers, each of which supplies a number of different lengths of cables. You want to allow the user to select the quantity, length and cable supplier.

One way to do this is to create option items in your model to represent every single one of the available options, as shown in the following figure:

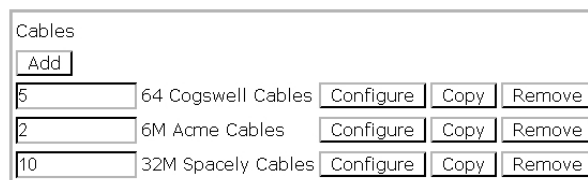


This approach will work, although it may be a bit tedious to implement and maintain and creates a model with many options that will never be of interest to the end user. An alternative approach might be to implement the different cable

length option items as an option item group, then include the cable length option item group under each of the manufacturers. This would ease maintenance: the modeler would have to look in only one place to update the cable option item information. However, this approach presents the end user with a huge list of cables to choose from and does not improve performance since there's still a large model to walk. A better alternative is to create a submodel that allows the user to select a cable manufacturer and length, then use dynamic instantiation to let the user add as many different cable types and lengths as necessary, as shown in the following figure:



The following figure shows a sample cable selection UI that uses dynamic instantiation to allow end users to configure their cable selections.



As you can see, this approach keeps the model small. Maintaining this model will be easier since the modeler no longer has to deal with a huge number of duplicate option items, performance is better since the model is smaller, and configuration is easier for the end user since there isn't a long list of cable types and manufacturers to look through in order to find the right one.

Use popup-qty Controls for Entering Quantity

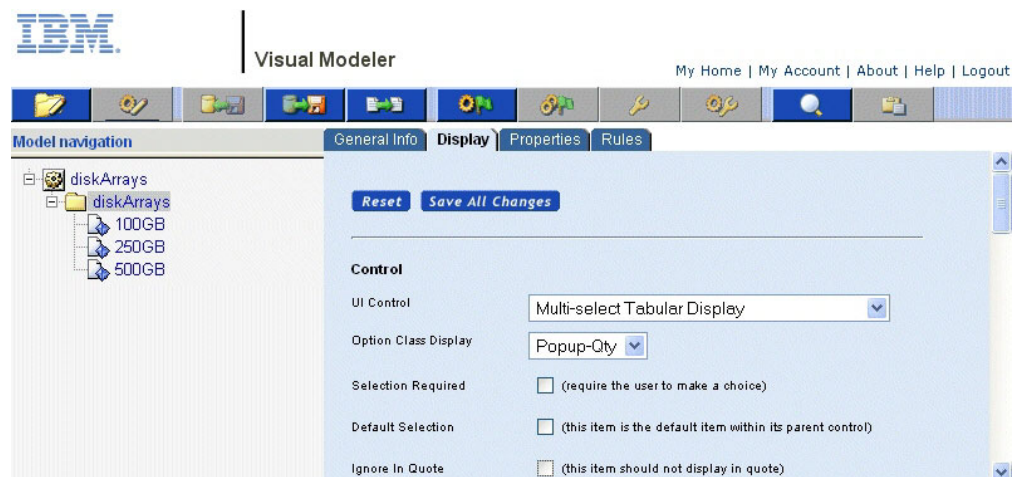
Sometimes the modeler wants to allow users to select an item, then enter the number of items they want. The best way to do this is to set the Option Class Display to **popup-qty**. When the end user selects an item, a quantity box will display, allowing the end user to enter the number of items they want. Some modelers do not like the placement of the quantity box, so instead use the User Entered Values (UEV) control to display an edit field next to the item in which

users can enter a quantity. The problem is that the behavior of the **popup-qty** control differs significantly from the behavior of UEV controls: the **popup-qty** control has quantity processing built in, while UEV controls require additional work.

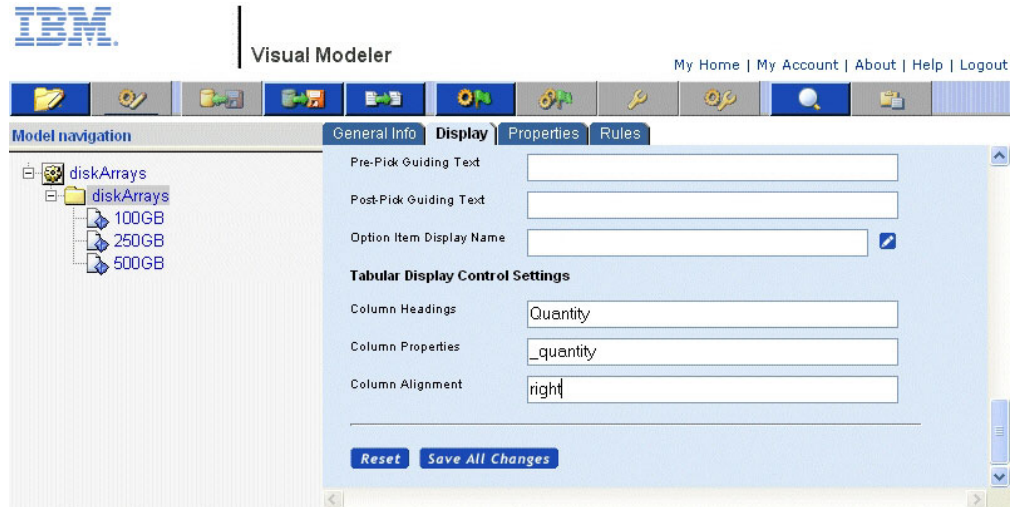
When an end user enters a quantity in a **popup-qty** box, the application automatically selects the quantity of the selected item. Any properties attached to the item are included in the configurator state (property pool), and the values of any numeric properties are multiplied by the quantity entered. UEV controls were designed to capture some additional information from the user. To get the UEV to behave as a quantity, the modeler must write an expansion rule that takes the value entered in the UEV control and picks that many of the selected item. Using the value entered in the UEV control to set the `_quantity` property using an assignment rule will not work as expected, since this does not automatically create instances of the item's properties in the property pool.

To display a **popup-qty** box beside the item selected, use the **popup-qty** Option Class Display style and one of the tabular displays with quantity controls. This will ensure the correct number of items are selected and the correct properties are copied to the property pool.

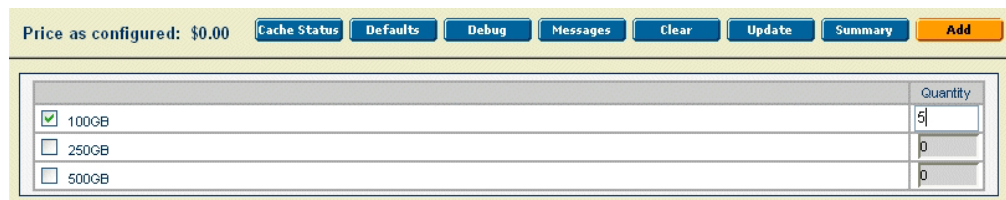
For example, the following figure shows how to set up a **popup-qty** control using the Visual Modeler. From the Models and Groups panel, select the model you wish to modify, then click the **Edit Model** icon. The Model navigation page is displayed. Click the option group you wish to modify, click the **Display** tab, then select **Multi-select** Tabular Display from the UI Control drop-down list, as shown in the following figure:



Scroll to the bottom of the page and enter the Column Headings, Column Properties, and Column Alignment settings. The following figure shows sample settings:



Finally, compile and test the model. You should see a popup-qty control placed as you specified on the Product Configurator page.










Properties

Properties are common in the Sterling Configurator. Modelers attach properties to models, option classes and option items and then write rules that work on these properties in order to display messages, show, hide, or select items, and even set the values of other properties. Considering the important part that properties play in modeling a configurable item, some care should be given to how properties are defined and used. This section outlines some useful tips and procedures to follow when defining and attaching properties.

Use Meaningful Property Names

When developing a model, under severe time constraints, the modeler is tempted to take shortcuts in order to speed the development process. One of the most common shortcuts is to create properties with short, and often vague or cryptic, names. This may speed the development of the model in the short term, but dramatically increases the amount of effort required to maintain the model. The modeler should always design their models so that it is immediately obvious what a given property represents. The more meaningful the name you give to a property, the easier it will be to search, debug and maintain the model.

Consider the following example:

model node	properties
 Model	<pre>tmr=sum(mr) tmp=sum(mp) amr=value(tmr)-value(tmp)</pre>
 OC1	
 oi1	mr=1
 oi2	mr=2
 OC2	
 oi3	mp=2
 oi4	mp=1

At first glance it may not be obvious what the properties assigned to this model are trying to accomplish. With a little more time spent creating meaningful names it becomes much easier to grasp the essence of all the properties and how they relate to one another.

Do Not Use the Same Property to Mean Two Different Things

Modelers may reuse an existing property instead of creating a new property designed specifically for the problem at hand. This has two possible implications:

- The model may be harder to understand if the existing property name bears no relation to the problem at hand.
- Re-use of the property name may actually cause errors if the re-use conflicts with the property's original use.

Let us revisit our example from the previous section. Suppose that the modeler created a property to store memory required and called it memory (see previous section for why that was a bad choice to begin with). The modeler needs a property to store memory provided, and notices a property called memory and decides to use it instead of creating a new property. Now, at first glance, it looks like all option items require some amount of memory, instead of two items requiring memory and two providing it. Not only that, but our total_memory_required property will no longer have the correct value, since it now performs a sum of both memory required and memory provided. If modeled in this fashion, then the modeler will have to do extra work to separate out the specific instances of the properties required: such as using full or relative paths to the items containing the appropriate property instances. (See Rules on why using paths to specific instances of properties can be a bad idea.)

Define Properties at the Appropriate Level in the Model Hierarchy








Properties may be defined at any level in the model group hierarchy, from the root model group level to the individual model level. Where a property is defined determines which models can see and make use of the property. A little thought during the design of your models will speed model development and help prevent property clutter. Use the following guidelines to determine where a property should be defined:

- If a property will only be used in a particular model, then define the property at the model level.

- If a property will be used in more than one model within a particular model group, then define the property at that model group level.
- If a property will be used in models that span model groups, then define the property in the first model groups that contains all of the model groups whose models will use the property.
- As a last resort, define the property at the root model group node.

Using Multiple Properties with the Same Value

Multiple properties with the same value can sometimes make a model easier to build and maintain. For example, suppose that you are building a model that allows the user to choose from a selection of disks arrays. Each type of disk array has some number of disks associated with it. The user can choose multiple disk arrays of any type. One of the pieces of information that you need to calculate is the total number of disks that the user has selected (see below).

model node	properties
 DiskArray SubModel	total_disks=sum(disks)
 100GB disk arrays	
 blue	disks=4
 mauve	disks=6
 250GB disk arrays	
 blue	disks=8
 mauve	disks=12

Now let us assume that you realize that you also need to know the number of 100GB disk arrays and the number of 250GB disk arrays. Instead of calculating these values by specifying item paths to the properties that we want, or writing rules that have to be attached at a particular point in the model, or re-working all the disk and total_disk properties, we can simply define a couple of new properties that have the same values as our old disk property (see below).

model node	properties
DiskArray SubModel	total_disks=sum(disks) total_100GBdisks = sum(100GB_disks) total_250GBdisks = sum(250GB_disks)
100GB disk arrays	
blue	disks=4 100GB_disks=4
mauve	disks=6 100GB_disks=6
250GB disk arrays	
blue	disks=8 250GB_disks=8
mauve	disks=12 250GB_disks=12

Now, if we want the total disks, we can still get sum(disks). If we want the individual values, then we can get those without specifying paths to individual properties or modifying the work that we had already done.

Use Worksheets to Simplify Property Assignment

When developing a model, it is often necessary to assign the same set of properties to multiple option classes or option items. Worksheets are very useful in this case, since they allow you to rapidly set the values for a particular property on any number of option classes or items. This is especially true when using a formula to set the value of a property in multiple places. The modeler can simply copy and paste the formula onto all the items required. For example, we have some display properties that are set for each item within a tabular display. We use a worksheet to allow us to easily cut and paste the formulas for col1 and col2 to each item in the option class.

Worksheet: modelDisplay <input type="button" value="Select"/> <input type="button" value="New..."/>		
modelDisplay		
Item	col1	col2
U100	$\{\text{expand}(\text{"min_array_disk"})\} / \{\text{expand}(\text{"max_array_disk"})\}$	$\{\text{expand}(\text{"min_cache_memory"})\} \text{ GB} / \{\text{expand}(\text{"max_cache_memory"})\}$
U600	$\{\text{expand}(\text{"min_array_disk"})\} / \{\text{expand}(\text{"max_array_disk"})\}$	$\{\text{expand}(\text{"min_cache_memory"})\} \text{ GB} / \{\text{expand}(\text{"max_cache_memory"})\}$
U1100	$\{\text{expand}(\text{"min_array_disk"})\} / \{\text{expand}(\text{"max_array_disk"})\}$	$\{\text{expand}(\text{"min_cache_memory"})\} \text{ GB} / \{\text{expand}(\text{"max_cache_memory"})\}$

An added benefit of using worksheets is that can provide a concise picture of a section of the model. With a little thought and planning, a worksheet can provide an overview of a particular section of the model or a complete representation of the solution to a particular problem. Below is a different view of the same option class. In this case, we are interested in seeing all the min and max properties that are set for each of the option items.

Worksheet: modelMinMax				
Item	min_array_disk	max_array_disk	min_cache_memory	max_cache_memory
U100	4	252	4	64
U600	64	508	6	=(value(exp_cache) == 0) ? 64 : 128
U1100	128	1148	6	128

Avoid Chaining Property Formulas

Properties attached to an item do not have any notion of sequence. By this we mean that, when using formulas to set property values, we cannot rely on any particular order of evaluation of the formulas. If property A contains a formula and property B contains a formula that relies on property A, then we have no guarantee that the rule created from formula B will fire after the rule created for formula A. In order to get around this issue, the modeler has two choices:

- Turn the first formula into a rule that fires before the second formula is evaluated. All rules generated from formulas have a priority of 50. By creating a rule for the first formula, and setting its priority to be less than 50, we ensure that the value of property A will be set before the value of property B is calculated.
- Turn on repeat rule firing. In this case the first phase of rule-firing will calculate the value for property A. The second pass of the rule-firing loop will calculate the value of property B based on the value of property A computed in the first pass.

Note: Massive amounts of chaining of formulas in this way may result in degradation of performance due to the number of passes through the rule-firing loop necessary to satisfy all the conditions. For this reason, we recommend the first alternative and advocate limiting formula chaining as much as possible.

Rules

Rules affect the efficiency and ease of maintenance of your model. This section describes considerations to keep in mind when writing rules.

Rule firing conditions

Rule conditions are created by applying boolean operations to relational expressions. A relational expression is the comparison of one function/property pair with another function/property pair using relational operations such as less than, equal to, greater than, in, not in, and so on. The result is either true or false. Boolean operators like AND and OR wrap sets of these relational expressions. The relational expressions are called fragments, as they are fragments of a rule. The left-hand-side of the relational operator is often abbreviated LHS, while RHS stands for right-hand-side.

Order rule fragments so that rules fire only when necessary

The evaluation of rule fragments determines when a rule fires, so the order in which fragments appear in a rule is important. The more quickly the model can determine whether a rule is true or false, the more efficient the model can be. Placing rule fragments in order, from most likely to prevent the rule from firing to least likely to prevent the rule from firing, can improve performance.

Always test your rules to ensure that they fire only when appropriate. Knowing under what circumstances a rule's results are or are not be used is also important. For example, an expansion rule that always fires but the quantity formula results in zero, or if there are not any matches for the formula in the > and <= fields in the expansions section, is very inefficient.

Create general-purpose rules

Whenever possible, write rules that are as general as possible. Consider where the messages should be displayed when writing rules. A rule attached directly to a product will trigger at that Option Item and the message will be displayed in that Option Class.

For example, the following rule can be attached to any item to which the productType and handsetType properties are attached:

```
If propval(productType) != value(selectProductType)
and propval(handsetType) != value(phonePreference)
set _isVisible=0
```

This rule fires only for items where the productType property is attached AND does not match the selected product types AND if the selected phone preferences do not match the current item's preferences. A general rule such as this one can replace dozens of other specific rules such as the following specific ones:

```
If propval(productType) == literal("handset")
and propval(handsetType) != literal("camera")
and value(phonePreference) == literal("camera")
set _isVisible=0
If propval(productType) == literal("handset")
and propval(handsetType) != literal("flip")
set _isVisible=0
```

Use formulas where appropriate

In many circumstances, formulas can be used instead of rules. During modeling, formulas are maintained as attached properties that have as their value an expression that is evaluated at runtime. If any of the functions referenced in the expression cannot be evaluated, the formula acts like a rule that hasn't fired. If multipass rule firing is turned on, the formula is reevaluated during each firing pass until rule firing ends or until the formula produces its result. Use a formula rather than a rule when the only condition for requiring that you compute a result is that the function/properties used in the formula have values.

For example, suppose that you want to compute the turning radius for truck components such as axel and wheelbase to ensure that a customer's choice of truck components makes sense. You might attach a formula to the relevant truck components to compute the turningRadius as follows:

```
turningRadius = value(axelTurnFactor) * value(wheelBaseTurnFactor)* sum(turningElements)
```

This formula fires when each of the value(axelTurnFactor), value(wheelBaseTurnFactor), and sum(turningElements) expressions all produce numeric results. The equivalent rule is as follows:

```
if (value(axelTurnFactor) >= 0 or value(axelTurnFactor) < 0)
and (value(wheelBaseTurnFactor)>= 0 or value(wheelBaseTurnFactor)< 0)
and (sum(turningElements) >=0 or sum(turningElements) <0)
turningRadius = value(axelTurnFactor) * value(wheelBaseTurnFactor)* sum(turningElements)
```

The condition portion of the rule is quite long and seems to always evaluate to true. However, functions can return NULL if a property that they reference does not exist, so this rule is really checking that the result is non-NULL by evaluating whether a returned value is ≥ 0 or < 0 .

Avoid writing rules

Writing rules can be avoided in the following situations:

- Pricelist eligibility conditions
If the requirement is to make certain option available or hidden, or to make different prices available based on customer profile, pricelists may be used instead of rules
- Model Structure
While restructuring the option class, building block, or submodel.

Avoid specifying paths to instances of items or properties

The left hand side (LHS) and right hand side (RHS) of a rule fragment consist of a function and a property name. The property name can contain both relative and absolute path information. However, specifying a property's path information in a rule fragment can result in the rule becoming inoperable if the path information or option classes change.

For example, the following rule references wheelSize and wellSize using fully specified path information. If the modeler ever needs to rename either the wheels or fender option classes, or wishes to reuse the rule in some other model, the rule may not operate correctly.

```
If value(*.wheels.wheelSize) == literal("17in")
and value(*.fender.wellSize) < literal(17)
set _isVisible=0
```

Use path information only if you want to access one specific instance of a property, and then only if it isn't possible to make a new property type to hold this value. If you must reference a property's path name, it is often better to use relative pathnames rather than absolute pathnames.

Constraint tables vs. rules

This section explains the trade-off between using constraint tables to limit customer choices versus using rules. Constraint tables limit a customer's choice of one or more option items based on the customer's choice of another option item. For example, the choice of an exterior color for a car might limit the choice of interior colors.

Constraint tables work best for simple validation. For example, an option item does or doesn't work with another option item. Simple constraint tables are easier to maintain than rules. However, large, complex constraint tables can be hard to maintain and can lead to performance issues.

Constraint tables are turned into rules internally.

Rules are best for expressing complex validation issues, and are more versatile than constraint tables. While both constraint tables and rules can display error messages, you can also create rules to set properties or make choices.

Modular Development

This section explains some of the techniques for simplifying model creation and maintenance. Selecting the appropriate technique may have a significant impact on model performance.

- Using Option Class Groups, Option Item Groups, and Sub-assemblies:

This technique works well when a group of options is repeated in many different models. For example, suppose that every computer you sell includes a list of hard drives that the user can choose from. Creating Option Class Groups, Option Item Groups, and Sub-assemblies allows the modeler to create and maintain common information in one place, then use it in many places.

One drawback is that this technique can lead to overly large models if a sub-assembly is included in the same model many times.

- Sub-model punch-in and punch-out:

This technique is useful when a configuration contains a selection that is also configurable. You can use sub-model punch-in and punch-out to nest complex configurations within one overall selling model.

One drawback is that all copies of the configured item will have the same configuration.

- Dynamic instantiation:

This technique allows multiple instances of a configured item within a single model. Each instance can have a different configuration.

Tools

Modeling can be a time consuming and tedious exercise, but in the end the correctness of the modeling and the scalability of the created solution are key to the success of the project. To aid in creating scalable and correct models, we have developed a collection of tools that can be used in various phases of development to guide the modeler. During development, the trace log and the model reporting tool can help the modeler determine which models to debug. Before pushing models into production, their scalability and stability can be tested using the load testing platform. Finally, during execution, the model cache status page can provide insights into the model's usage of the system, and the log analyzer can be used to make sense out of megabytes worth of log information.

Using the Trace Log

The trace log shows the execution of the rules engine. This is often, though not always, the most time consuming part of each request that the configurator makes to the server. The trace log is designed to provide the information necessary to debug rules that are misbehaving and to track the execution time of rules, so always start your debugging by reviewing the trace log. You create trace logs using the Visual Modeler. To do so:

1. Go to Model Group navigation and navigate to the model you wish to debug.
2. Select the model from the Models and Groups panel.
3. The model displays in the Model Preview tab.
4. Click the Test icon.
5. Click Debug.

The trace log appears in a separate window. The log consists of two sections. The first section is the rule firing trace and the second section is the property pool as it exists at the end of rule firing. The following illustration shows a section of a sample rule firing trace.

Rule Firing Trace		
#	(ms)	Result
0	0	Applying picks
1	0	Firing phase [0]:begin
2	0	Firing rules on MXDS-7500.Disk Drives
3	0	MSG_E_Available_HDD_Slots ==> fires on TRUE - priority = 50
4	0	Property not found [MX75_HDD_Ordered or MX75_Bays_Available], taking null action
5	0	took 0ms.
6	0	Firing rules on MXDS-7500.Memory
7	0	MSG_E_MX75_Memory_Software_Check ==> fires on TRUE - priority = 50
8	0	TESTING:sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
9	0	FALSE: 0.0 < 0.0
10	0	FALSE: sum(MX75_Mem_Ordered) <sum(MX75_Mem_Required) [nullreturn=false]
11	0	took 0ms.
12	0	Firing rules on MXDS-7500.Placeholder for auto memory selection
13	0	ASG make placeholder invisible ==> fires on TRUE - priority = 50
14	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
15	0	took 0ms.
16	0	Firing rules on MXDS-7500.AutoMemory
17	0	EXP_MX75_Automatic_Memory_Selection ==> fires on TRUE - priority = 50
18	0	Left side property [MX75_Mem_Auto_Select] not found, taking null action
19	0	took 0ms.
20	0	Firing rules on MXDS-7500.Software.Application
21	15	EXP_MX75_Fire_Wire ==> fires on TRUE - priority = 50
22	15	Left side property [MX75_Video_Editing] not found, taking null action
23	15	took 15ms.

The rule firing trace has three columns:

- A sequence number, useful for communicating with others about rule issues. It's easy to tell someone, "See line 42 where it says Xxx?"
- Elapsed time. This logs how long it took from the time the log entry was made until the start of rule firing.
- The body of the trace log. This shows aspects of the rule firing, such as a condition being evaluated, an assignment occurring, the start of a rule or the conclusion of a rule, and so on. The log shows the number of milliseconds needed to fire a rule after each rule firing entry. The total number of milliseconds needed to run the model is logged at the end of the rule firing trace.

The property pool trace also presents three columns:

- Name is the full path name to the item and the property on that item.
- Type is the property type for the named property, such as Numeric, List, or String.
- Value is the value of the property after the rule has fired.

The following illustration shows a section of a sample property pool trace:

Property Pool		
Name	Type	Value
MXDS-7500.CONFIG: FIRST FIRE	Numeric	1.0
MXDS-7500.MX75_Bays_Available	Numeric	2.0
MXDS-7500.MX75_Card_Slot_Available	Numeric	4.0
MXDS-7500.MX75_Mem_Ordered	Numeric	0.0
MXDS-7500.MX75_Mem_Required	Numeric	0.0
MXDS-7500.Service Options.View Service.UI: COLUMN SPAN	Numeric	2.0
MXDS-7500.Placeholder for auto memory selection.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONFIG CELL HTML CLASS	String	configsubcell_plain
MXDS-7500.Microprocessor.UI: CONSTANT GUIDING TEXT	String	Dual processor capable motherboard, supporting Intel processors
MXDS-7500.Microprocessor.UI: CONTROL	String	RADIO
MXDS-7500.Disk Drives.UI: CONTROL	String	RADIO
MXDS-7500.Placeholder for auto memory selection.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Software.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory Cards Message.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: CONTROL	String	controls/allpicked.jsp
MXDS-7500.Accessory.Graphic Cards.UI: CONTROL	String	RADIO
MXDS-7500.Accessory.Cards.UI: CONTROL	String	CHECKBOX
MXDS-7500.Accessory.Network Cards.UI: CONTROL	String	RADIO
MXDS-7500.Service Options.View Service.UI: CONTROL	String	CHECKBOX
MXDS-7500.Placeholder for auto memory selection.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.UI: DEFAULT SELECTION	String	no
MXDS-7500.Accessory Cards Message.Accessory Card Image.UI: DEFAULT SELECTION	String	no

Use this log "single user" to get a feel for how extensive the rules are per click. Check how long is it taking to fire the rules. If the answer is more than 100-200ms you may have scalability problems. If you do, use the trace log to figure out if any particular rules are performing badly. Make sure there are no java exceptions in the debug trace. A java exception causes all rules to stop firing at that point forward in the sub-assembly or model.

Using the Model Reporting Tool

The model reporting tool can provide an overview of a model's size relative to other models. Use it to help make decisions about which models to test. You can track the test results over time so that you can determine the amount of change to the model.

Using Load Testing Tools

Load testing tools help you determine how your model will perform once deployed. Before using the load testing tools:

- Understand what is being tested.

- Isolate your test cases so that you know what the impact means (local vs. remote LAN testing with and without clustering, with and without web fronting, and so on).
- Understand that as models change, so must any scripts that you use to perform testing and replay test scenarios.
- `cmd=configstatus` shows the current contents of the cache.

Performance

Rules

- A rule that adds memory by:


```
totalMem = value(*.adapter.1.memory) + value(*.adapter.2.memory)
          + value(*.adapter.3.memory) + value(*.adapter.4.memory)
```

 will perform much more slowly than:


```
totalMem = sum(memory)
```
- If the memory property exists in other places for other uses so that `sum(memory)` would produce the wrong value, then introduce additional properties on the adapter items 1-4 called `adapterMemory`, and use:


```
totalMem = sum(adapterMemory).
```

 This is much less maintenance effort that maintaining:


```
totalMem = value(*.adapter.1.memory) + value(*.adapter.2.memory)
          + value(*.adapter.3.memory) + value(*.adapter.4.memory)
```
- Write rules to fire only when they are needed:
 A rule that assigns `totalMem = sum(mem)` only needs to fire if `count(mem) > 0`

Properties

- Define properties at the correct position in the model group hierarchy:
 - If they are local only to this model, then define them in the model.
 - If they are more global than the current model group, then define them at the lowest point in the model group tree that is an ancestor of a model where you wish to use the property.

Development and Maintenance

There could be several modelers in a team sharing the same model, have multiple releases to address and many models to work on. This section outlines some useful tips to follow during development and maintenance:

Personal Environment

In personal environment modelers cannot share their models. They work on a local copy of models and export them. Problems occur when the same model is modified by other modelers. Hence, personal environments are preferred by modelers for training and experimental purposes.

Shared Environment

In shared environments modelers can share the models for a release. Conflicts occur when many modelers make changes to the same model. It could also occur when rules, properties are used by many models, or by attaching models to other models. Conflicts are avoided by assigning specific models to each modeler, avoid sharing rules and properties. It is a good practice to keep the XML files separate.

Real-time Version Control System

A real-time version control system is useful in situations when we want to move to a previous version. For example, if a modeler deletes a model by mistake we can step back to the previous version.

Chapter 14. Definitions for Out-of-the-box Functions

The following table lists the functions supported by Visual Modeler:

Table 1. Functions supported by Visual Modeler

Functions	Definition
checkwslookup	Check if the correct properties exist to invoke a Web service.
childsum	Sum of the specified property values defined at this node and any of the node's children (and recursively down to children of child nodes and so on).
count	Counts the number of objects (selected option items, models, or groups) having the specified property.
isselected	Returns true if the option item is selected; otherwise returns false.
length	Returns the length of a string property. If there are more than one instance of that property in the property pool, length function returns the length of the first one it finds. To restrict the search to the parent hierarchy of the item to which the rule is attached, use the parentlength function.
list	Used with the operators in and not in to check if a property value is included or not in a specified list of values.
literal	Exact match of the literal value.
lookup	Used to look up values specified by name from a properties file: the lookupValues.properties configuration file. A string property containing the key is used to find the entry in the property file. This entry defines the properties for this key and the values that each of these properties should be set to when the function is invoked. For example, Suppose the following is defined in the properties file: <pre>Color=blue,green,red Color.blue=#0000FF Color.green=#00FF00 Color.red=#FF0000</pre> Then if the lookup<Color> function is invoked, the corresponding properties blue, green, and red will be attached at the appropriate node.
max	Returns the maximum property value from all selected items with the specified property.
min	Returns the minimum property value from all selected items with the specified property.
parent	This function walks up the tree from the current location to see if the property has been defined anywhere at or above the current location. For example, if the rule is attached at an option item level, then the property will be looked for on the option item itself. If it is not defined there, then the option class to which the option item belongs will be looked at to see if the property is defined there.

Table 1. Functions supported by Visual Modeler (continued)

Functions	Definition
path	<p>Returns the relative path of a property from the root node of the model. The root node is replaced with an asterix (*).</p> <p>For example: if the path to the property is MXDS7500.Software.Applications.MSOffice.MX75_Mem_Required path returns *.Software.Applications.MSOffice.MX75_Mem_Required.</p>
parentlength	<p>Returns the length of a string property, after checking for properties in the current item's parent hierarchy. For example, In the following scenario:</p> <pre> Model -- OC1 - prop = "short" -- item1 (rule if (parentLength(prop) > 6), do something) -- OC2 - prop = "string that is exactly 42 characters long!" -- item2 (rule if (parentLength(prop) > 6), do something) </pre> <p>When the rule attached to item2 is evaluated, parentLength will return 42 and, since 42 is greater than 6, the action "do something" will be taken.</p>
propval	Returns the value of a property even if the option item has not been selected.
propval	Returns the value of a property even if the option item has not been selected.
rawpath	Returns the actual path of the property without stripping off the root node. For example: If the path to the property is MXDS7500.Software.Applications.MSOffice.MX75_Mem_Required , rawpath returns MXDS7500.Software.Applications.MSOffice.MX75_Mem_Required.
sum	Sum of the property values from all selected items having the specified property
value	Uses the property value for comparison. If multiple items exist on the order with the given property, then the maximum value is used.

Chapter 15. Definitions for Out-of-the-box Configurator Properties

Built-In Properties

The Built-In properties are listed in the table below:

Table 2. Built-In Properties

Property	Data Type	Description
CONFIG: FIRST FIRE	numeric	1 if this is the first time firing rules, 0 otherwise. Value will be 1 at the start of the user session, and therefore when used in rules fragments a high priority (i.e. a low number such as "5" or "10") will ensure that the rule is run at the beginning of the session.
CONFIG: POOL SIZE	numeric	This property is used to assign the number of copies of a model to keep in the model pool.
CONFIG: REPEAT FIRING	string	"yes" or "true" turns on looping in the rule engine, causing rules to fire as long as the current state keeps changing. Since rules are removed from the rule list whenever they fire, this is not an infinite loop.
CONFIG: SUBMODEL NAME	string	The encoded name of another model. Encoding replaces potentially unsafe file system characters with _XXXX where XXXX is the hex representation of their Unicode character code. For example, a space is represented by "_0020".
CONFIG: SUBMODEL RETURN	string	"yes" or "true": When punching into a submodel specified by CONFIG: SUBMODEL NAME, returns back the the parent model the BOM of the child model
_cacheKey	string	Use on a model node to contain the key used to store the model in the model cache.
_description	string	The property has the description of an item.
_errorCount	numeric	Number of errors encountered during rule firing.
_fileSize	string	Size of the model XML file (a Long value represented as a String)
_lastModified	string	Last modified date for a model as a string (number of seconds since some important date).
_modelTabs	list	List of tab names for the model.
_name	string	The name of an option item, option class, or model.
_parent. <item names>	varies	Properties inherited by a submodel from the parent.
_pickItems	list	Used internally to keep track of picked items.
_pickmap. <itemKey>	string	Mapping of an item to an option class.
_picks	list	Used internally to keep track of picked items.
_quantity	integer	Quantity selected, if >0 the item is picked. Note: Setting _quantity in the assignment action of a rule is prohibited.
_sequence	numeric	Rule firing sequence. If 0, this is the first time through the loop, 1 is the second, and so on.

Table 2. Built-In Properties (continued)

Property	Data Type	Description
_sku	string	The item ID assigned to the model, option class, or option item in the Configurator model group. Note: When used in the "for each" rule fragment, the loop will return more items that we often want it to. Instead, use a property other than _sku that can filter out more items.
_tabMembers < tab number >	list	Where <#> is a tab number (0...N), these properties contains the names of the root level option classes that are part of the tab whose index is <#>
CONFIG: ON UNSELECT SKIP CHILD RULE AND CURR RULE FIRING	string	"yes" or "true" skips the rules attached to the node and its subtree, if the node is not selected.
CONFIG: ON UNSELECT SKIP CHILD RULE FIRE BUT FIRE CURR RULE	string	"yes" or "true" skips the rules attached to the node's subtree but fires the rules attached to the node, if the node is not selected.
CONFIG: DISABLE SUBMODEL VALIDATION ON SUBMODEL RETURN	string	"yes" or "true" skips the validation of cached sub-model configuration present in the parent model on sub-model return. Relevant only if CONFIG: SUBMODEL RETURN is enabled.

User Interface (UI) Properties

The User Interface (UI) properties are listed in the table below:

Table 3. User Interface Properties

Property	Data Type	Description
UI: ADDITIONAL DESCRIPTION	string	You can use this property to add additional descriptive text to an option class. Use this property in conjunction with the UI: DISPLAY RESULTS property.
UI: ALIGNMENT	string	The "Horizontal" or "Vertical" options control layout of radio buttons and check box controls.
UI: AUTOMATIC POST	string	Selecting "yes" or "true" turns on automatic posting for an option class. After a customer makes a pick of an option item, you want the server to re-display the page so that rules can be fired and any changes to the available option classes can be displayed. However, if you do not want picks in an option class to cause a re-display, then set this property to "no" or "false". This is equivalent to selecting On User Request from the Submit to Server Display property drop-down list. The option class is displayed with Update button; after making a pick in this option class, a user can click the Update button to request a re-display of the page from the server.

Table 3. User Interface Properties (continued)

Property	Data Type	Description
UI: CLASS DISPLAY NAME	string	Use this property at the model level to determine what is displayed as the displayed name of option classes. By default, this property takes the value <code>\${expand("_description")}</code> which means that the value of the option class's Description field is displayed. For example, if you want to display option class names instead of descriptions, then set this property to <code>\${expand("_name")}</code> . You can overwrite this value at a single class by using the UI: DISPLAY NAME property.
UI: COLUMN ALIGNMENT	string	Use this property in the tabular display control to specify the alignment of the values in the column. The tabular display control uses the ";" character to separate entries from each other. The format of this column is something like: "left;left:center;right".
UI: COLUMN HEADINGS	string	Use this property in the tabular display control to specify the titles of columns. Each title is separated from each other with the ";" character. For example: "Speed;Pins;Manufacturer".
UI: COLUMN PROPERTIES	string	A semi-colon-separated list of property names used in the tabular display of properties. For example: "SPEED;NOPINS;SUPPLIER", where SPEED, NOPINS, and SUPPLIER are properties defined on option items in an option class
UI: COLUMN SPAN	numeric	This property controls how many columns an option class requires for its display in the customer-facing display of the model. This is the same as entering a number for the Number of Columns field on the Display tab.
UI: CONFIG CELL HTML CLASS	string	This sets the CSS class attribute in the HTML. Use this property to control the look-and-feel of cells. The Visual Modeler uses the internal.css CSS file when you test models.
UI: CONSTANT GUIDING TEXT	string	This property defines the guiding text that will always be shown for an option class. This is the same as entering text for the Constant Guiding Text field on the Display tab.
UI: CONTROL	string	This is the name of the JSP fragment used to render an option class. Do not use UI: JSP FILENAME at the option class level.
UI: DEFAULT SELECTION	string	Selecting "true" or "yes" on an item makes the item a default selection within its parent option class
UI: DISPLAY ADDITIONAL INFO	string	When using dynamic instantiation, displays in a parent model user interface some information from the submodel. UI: DISPLAY ADDITIONAL INFO be used to provide descriptive information specific to each instance of a sub-model. For example, attach this property to the root node of a submodel, and pass it as an output property to the parent model. The information will be displayed next to the item in the parent model. Because each instance of a dynamically instantiated submodel is configured differently, you may want to provide some distinguishing information about each instance upon returning from the submodel to the parent.

Table 3. User Interface Properties (continued)

Property	Data Type	Description
UI: DISPLAY NAME	string	Use this property to determine what is displayed as the displayed name of the option class. By default, this property takes the value <code>#{expand("_description")}</code> which means that the value of the option class's Description field is displayed.
UI: DISPLAY RESULTS	string	This property is deprecated. A property that is displayed along with the description of items. This special property also allows the usage of text expansion macros. Currently we support: <code>#{expand(propname[,defaultValue[,pictureString]])}</code> but the name of this "function", expand in this case, is accessed via the object manager. An example usage is to set a description string in the UI: ADDITIONAL DESCRIPTION property, and then set the value of this property to <code>#{expand("UI: ADDITIONAL DESCRIPTION")}</code> . To add additional macros, define a new class that implements the IExpansionHandler interface, and put a reference to it into the object manager.
UI: HELP URL	string	This URL is used to turn an option class description into a hyperlink, typically used to provide additional information about what that option class is for. It could also be a datasheet or any other hyperlink. Clicking on the hyperlink will bring up the page in a new window. This is the same as entering text for the Help URL field on the Display tab
UI: ICON GRAPHIC	string	Use this property with an option class to display a picture along with the description of the option class. This is the same as entering text for the Image field on the Display tab
UI: IGNORE IN QUOTE	string	When set to "yes" or "true", the item to which this property is attached to is filtered out of the summary page. It is also flagged as not visible in the BOM transfer to the shopping cart. This is the same as checking Ignore in Quote on the Display tab. This field is used to ensure that only selected option items are displayed in shopping carts. It suppresses option classes in the list of items in a shopping cart.
UI: ITEM DISABLE	string	"yes", "no". Used to prevent the user from selecting an option item while still displaying it on the UI
UI: JSP FILENAME	string	The name of the JSP page that will render the model: Configurator_Tabbed.jsp or configurator.jsp. This property is added to support easier customization and eventually to allow different presentations per model. Using the built-in customization elements of Sterling Configurator, it is possible to dynamically change pages as well.
UI: LEAD TIME	numeric	This property is attached to items in the model. It is used to build a maximum lead time for the entire model by finding the largest lead time of all items currently selected.
UI: NUMBER OF COLUMNS	numeric	This property shows the number of columns to divide the end-user configurator presentation. It is defined at the model level to manage how many columns are used to display the option classes for a model. This property in conjunction with UI: COLUMN SPAN, UI: ROW SPAN, and UI: SKIP COLUMNS controls how option classes are arranged on the page. This property is the same as setting the Number of Columns property in the Display tab.

Table 3. User Interface Properties (continued)

Property	Data Type	Description
UI: OPTION CLASS REQUIRED	string	Setting this property to "yes" or "true" causes Sterling Configurator to require that a selection be made for an option class. For radio buttons this causes the None selection to be removed.
UI: OPTION CLASS SELECT	string	This property is used to specify what UI control should be used when no specific UI: CONTROL value is specified. It is used to support importing models from external configuration systems or from earlier releases of the application. It takes "single" or "multiple" as values. It is only used in the absence of a UI: CONTROL property to determine if a radio button or check box control should be shown for an option class.
UI: OPTION CLASS VIEW	string	This controls the display behavior of an option class. POPUP: Show a standard option class. POPUP-QTY: Show a quantity box for each selected item within that control. INVISIBLE: Prevent the display of the control. Use INVISIBLE to hide option classes until other picks made by the customer requires the class to be displayed.
UI: POPUP-QTY ALLOWED VALUES	string	This controls what values are available for a selection in a popup drop-down list. Use this at the option class level, in conjunction with setting UI: OPTION CLASS VIEW to POPUP-QTY. A "," separated list of allowed values. Ranges can be specified with "-", so 1-4,7-9 is the same as 1,2,3,4,7,8,9. If you leave this field blank, then a text field is displayed with the current value; otherwise a drop-down list with the allowed values is displayed.
UI: POST PICK GUIDING TEXT	string	A guiding text message displayed with an option class description if the user has made at least one pick from within the option class. This is the same as entering text for the Pre-Pick Guiding Text field on the Display tab. This property is not displayed until a customer makes a pick.
UI: PRE PICK GUIDING TEXT	string	A guiding text message displayed with an option class description if the user has not made a pick from within the option class. This is the same as entering text for the Post-Pick Guiding Text field on the Display tab. Once a pick has been made, then this property is no longer displayed.
UI: PREVENT SELECTION	string	Selecting "yes" or "true" causes the Sterling Configurator to prevent the user from selecting items that would violate a constraint table rule. If the Constraint Selections display property is set to "Hide constrained items", then this property is set to "yes".
UI: PRICE	numeric	The price for an item that will be used if STATIC_PRICING or OVERRIDE_PRICING is set in the business rules. In the case of OVERRIDE_PRICING, this value will be used if a price cannot be found for the item in the price list.
UI: PRICING SKU	string	The SKU to use when looking up the item in the price list. Note that if you set a product ID value for this property, then it overrides the value of the Assigned Product ID in determining prices.

Table 3. User Interface Properties (continued)

Property	Data Type	Description
UI: PRICING STYLE	string	<p>This property is used at the option class level. It controls how prices of option items are displayed to the end user as follows:</p> <p>NONE: Do not display prices as user configures product.</p> <p>ABSOLUTE: Display prices next to option items as absolute prices.</p> <p>DELTA: Display prices next to option items as their effect on the price of the whole configured product.</p> <p>This property is the same as setting Pricing Style in the Display tab.</p>
UI: PRODUCT ID	string	If you associate a product with the node of a model, this property can be used to retrieve the product ID of the associated product. The value of this property is resolved at compile time. If the product ID is changed, you must re-compile the model for the change to take effect.
UI: PRODUCT NAME	string	If you associate a product with the node of a model, this property can be used to retrieve the product name of the associated product. The value of this property is resolved at compile time. If the product name is changed, you must re-compile the model for the change to take effect.
UI: PRODUCT DESCRIPTION	string	If you associate a product with the node of a model, this property can be used to retrieve the description of the associated product. The value of this property is resolved at compile time. If the product description is changed, you must re-compile the model for the change to take effect.
UI: ROW SPAN	numeric	This property controls how many rows an option class requires for its display in the end-user presentation of the page. In conjunction with UI: NUMBER OF COLUMNS and UI: COLUMN SPAN, this property controls the layout of the page viewed by end-users. This is the same as entering a number for the Number of Rows field on the Display tab.
UI: SHOW ITEM IMAGES	string	Selecting "yes" or "true" controls whether item images are shown.
UI: SKIP COLUMNS	numeric	This shows the number of columns to skip after this class. It is used to add to the count variable that is tracking how many cells are being used to lay out the option classes. This is the same as entering a number for the Number of Columns to Skip field on the Display tab. If you have used the UI: COLUMN SPAN property or UI: ROW SPAN for another option class, then use this property to account for table cells in the layout that the multiple span class uses.
UI: SUPPRESS NAME DISPLAY	string	Selecting "yes" or "true" causes Sterling Configurator to not display the names of option classes
UI: SUPPRESS NONE SELECTION	string	Selecting "yes" or "true" suppresses the NONE selection value for radio buttons..

Table 3. User Interface Properties (continued)

Property	Data Type	Description
UI: SUPPRESS UEV NONE VALUE	string	Selecting "yes" or "true" suppresses the NONE selection for UEV combo boxes. Use this in conjunction with UI: UEV ALLOWED VALUES property. For example, you have specified that a user-entered value field can only take the values Red, Green and Blue. If the value of this property is set to "yes", then None will not appear in the drop-down list of selectable values. If you set the value of this property to "no", or do not attach this property, then None will be a selectable value.
UI: UEV ALLOWED VALUES	string	Comma-separated list of values for a combination box UEV control. Suppose that you want to allow customers to enter only one color from a small list of colors. Enter the list as follows: Black,Blue,Green,Red,White When this property is set, the user-entered value option item is displayed as a drop-down list of these values. None is also displayed as a selectable option, unless you set the UI: SUPPRESS UEV NONE VALUE property to "yes". This property is the same as setting values in the Allowed Values display property.
UI: UEV ASSIGNMENT PROPERTY	string	The name of a property where a UEV will store its value. This property should be of the correct type to contain the UEV. Numeric properties can be used to hold INTEGER UEVs as well as NUMERIC UEVs. If the value of this property is just a property name, then the property will be set on the current item. If the value contains a path to a property as well as the property name, then the property will be set on the item referenced by the path if it exists. Once a user makes their pick in the user-entered value field, then the assigned property can be used by rules or in the display of the model, just like any other property. This property is the same as setting a value in the Assign Value to Property display property.
UI: UEV INTEGER VALUE	integer	The engine fills in this value when an integer UEV has a value in it. This provides you with a way to reference the value of the field without assigning it to another property.
UI: UEV LIST VALUE	list	The engine fills in this value when a list UEV has a value in it (not currently used). This provides you with a way to reference the value of the field without assigning it to another property.
UI: UEV NUMERIC VALUE	numeric	The engine fills in this value when a numeric UEV has a value in it. This provides you with a way to reference the value of the field without assigning it to another property.
UI: UEV POSTFIX	string	A string of text displayed after the UEV entry field. This property is the same as setting a value in the Text After Entry Field display property.
UI: UEV PREFIX	string	A string of text displayed before a UEV entry field. This property is the same as setting a value in the Text Before Entry Field display property.
UI: UEV STRING VALUE	string	Filled in by the engine when a string UEV has a value in it. This provides you with a way to reference the value of the field without assigning it to another property. See UI: UEV ASSIGNMENT PROPERTY to use another property.
UI: UEV TYPE	string	These are the options for the types of UEV control; "string", "integer", or "numeric".

Miscellaneous Properties

The following table summarizes some of the available properties for assignment :

Table 4. Miscellaneous Properties

Property	Data Type	Description
_constraintMessage	string	A message on an item because it is constrained
_constraintType	integer	Type of constraint; 0 is suggest, 1 is warn, and 2 is error
_description	string	An items description
_amEntitled	integer	0 false, 1 true
_isConstrained	integer	0 false, 1 true
_isSelected	integer	0 false, 1 true
_isViewable	integer	0 false, 1 true
_itemKey	integer	Database key of the item
_pickOverride	integer	0 false, 1 true; pick was overridden by a rule
_quantity	integer	Quantity; 0 quantities are not in the rule pool. Note: Setting _quantity in the assignment action of a rule is prohibited.
_ratio	numeric	Ratio of this item to its children, computed if nested within another parent .
_rawRatio	numeric	Raw ratio used in previous computation
_rulePick	integer	0 false, 1 true
_tabLevel	integer	Depth of this item

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing

IBM Corporation

North Castle Drive

Armonk, NY 10504-1785

U.S.A.

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing

Legal and Intellectual Property Law

IBM Japan Ltd.

19-21, Nihonbashi-Hakozakicho, Chuo-ku

Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be

incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licenses of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation

J46A/G4

555 Bailey Avenue

San Jose, CA 95141-1003

U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© IBM 2013. Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. 2013.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and [ibm.com](http://www.ibm.com)[®] are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at <http://www.ibm.com/legal/copytrade.shtml>.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of the Office of Government Commerce, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium and the Ultrium Logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Connect Control Center[®], Connect:Direct[®], Connect:Enterprise[®], Gentran[®], Gentran[®]:Basic[®], Gentran:Control[®], Gentran:Director[®], Gentran:Plus[®], Gentran:Realtime[®], Gentran:Server[®], Gentran:Viewpoint[®], Sterling Commerce[™], Sterling Information Broker[®], and Sterling Integrator[®] are trademarks or registered trademarks of Sterling Commerce[®], Inc., an IBM Company.

Other company, product, and service names may be trademarks or service marks of others.

Index

A

action classes
 extending 24

C

customization
 examples 25
customization checklist
 action definition 23
 theme and logo 25

D

data validation
 customizing 43

F

factory setup
 localizing 19

M

Multipurpose Internet Mail Extensions
(MIME) 43

P

post customization
 deployment 45

R

resource bundle localization 17
resource bundles
 localizing 18

S

Sterling Web EAR
 creating, deploying 13

U

users
 Sterling Configurator 16

V

validation
 data 35
validation rules 41
 customizing 43



Printed in USA