Notice

This documentation (the "Documentation") and related computer software program (the "Software") (hereinafter collectively referred to as the "Product") is for the end user's informational purposes only and is subject to change or withdrawal by CA at any time.

This Product may not be copied, transferred, reproduced, disclosed, modified or duplicated, in whole or in part, without the prior written consent of CA. This Product is confidential and proprietary information of CA and protected by the copyright laws of the United States and international treaties.

Notwithstanding the foregoing, licensed users may print a reasonable number of copies of the Documentation for their own internal use, and may make one copy of the Software as reasonably required for back-up and disaster recovery purposes, provided that all CA copyright notices and legends are affixed to each reproduced copy. Only authorized employees, consultants, or agents of the user who are bound by the provisions of the license for the Software are permitted to have access to such copies.

The right to print copies of the Documentation and to make a copy of the Software is limited to the period during which the license for the Product remains in full force and effect. Should the license terminate for any reason, it shall be the user's responsibility to certify in writing to CA that all copies and partial copies of the Product have been returned to CA or destroyed.

EXCEPT AS OTHERWISE STATED IN THE APPLICABLE LICENSE AGREEMENT, TO THE EXTENT PERMITTED BY APPLICABLE LAW, CA PROVIDES THIS PRODUCT "AS IS" WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT WILL CA BE LIABLE TO THE END USER OR ANY THIRD PARTY FOR ANY LOSS OR DAMAGE, DIRECT OR INDIRECT, FROM THE USE OF THIS PRODUCT, INCLUDING WITHOUT LIMITATION, LOST PROFITS, BUSINESS INTERRUPTION, GOODWILL, OR LOST DATA, EVEN IF CA IS EXPRESSLY ADVISED OF SUCH LOSS OR DAMAGE.

The use of this Product and any product referenced in the Documentation is governed by the end user's applicable license agreement.

The manufacturer of this Product is CA.

This Product is provided with "Restricted Rights." Use, duplication or disclosure by the United States Government is subject to the restrictions set forth in FAR Sections 12.212, 52.227-14, and 52.227-19(c)(1) - (2) and DFARS Section 252.227-7013(c)(1)(ii), as applicable, or their successors.

All trademarks, trade names, service marks, and logos referenced herein belong to their respective companies.

Copyright © 2006 CA. All rights reserved.
# OneClick Customization Guide

## Contents

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Polygon</td>
<td>44</td>
</tr>
<tr>
<td>Line</td>
<td>45</td>
</tr>
<tr>
<td>Example of Creating an Icon Shape</td>
<td>45</td>
</tr>
<tr>
<td>X and Y Coordinates</td>
<td>46</td>
</tr>
<tr>
<td>Defining Pipe Connection Location</td>
<td>46</td>
</tr>
<tr>
<td>Defining Image Components</td>
<td>47</td>
</tr>
<tr>
<td>Defining Text Components</td>
<td>51</td>
</tr>
<tr>
<td>Defining Selection Components</td>
<td>55</td>
</tr>
<tr>
<td>Defining Model Icon Tooltips</td>
<td>57</td>
</tr>
<tr>
<td>Chapter 5: Customizing a Model’s Information View</td>
<td>61</td>
</tr>
<tr>
<td>Overview</td>
<td>61</td>
</tr>
<tr>
<td>Creating or Modifying an Information View</td>
<td>63</td>
</tr>
<tr>
<td>Creating an Information Configuration File</td>
<td>63</td>
</tr>
<tr>
<td>Defining the Header</td>
<td>64</td>
</tr>
<tr>
<td>Defining the Subview</td>
<td>65</td>
</tr>
<tr>
<td>Adding a Field Subview</td>
<td>66</td>
</tr>
<tr>
<td>Adding a Table Subview</td>
<td>68</td>
</tr>
<tr>
<td>Adding an Application Subview</td>
<td>69</td>
</tr>
<tr>
<td>Adding a Related Model Subview</td>
<td>70</td>
</tr>
<tr>
<td>Adding a Related Models Table Subview</td>
<td>71</td>
</tr>
<tr>
<td>Defining a Subview Group</td>
<td>73</td>
</tr>
<tr>
<td>Associating the Information Configuration File with the Model Class or Model Type</td>
<td>74</td>
</tr>
<tr>
<td>Chapter 6: Creating Custom Privileges</td>
<td>77</td>
</tr>
<tr>
<td>Overview</td>
<td>77</td>
</tr>
<tr>
<td>Defining a Custom Privilege</td>
<td>77</td>
</tr>
<tr>
<td>Restricting Access to Attribute Values in Model Subviews</td>
<td>79</td>
</tr>
<tr>
<td>Grouping Privileges</td>
<td>80</td>
</tr>
<tr>
<td>Referencing the Privilege when Defining a Menu Item, Column, or Subview</td>
<td>81</td>
</tr>
<tr>
<td>Chapter 7: XML Usage Common to All Customization Files</td>
<td>83</td>
</tr>
<tr>
<td>Using Parameters</td>
<td>83</td>
</tr>
<tr>
<td>Using a Select Case</td>
<td>84</td>
</tr>
<tr>
<td>Manipulating the Output of Attributes with Renderers</td>
<td>84</td>
</tr>
<tr>
<td>Boolean Renderer</td>
<td>85</td>
</tr>
<tr>
<td>Commented Text Renderer</td>
<td>85</td>
</tr>
<tr>
<td>Date Renderer</td>
<td>85</td>
</tr>
<tr>
<td>Enumerated Attribute Renderer</td>
<td>86</td>
</tr>
<tr>
<td>Lists Renderer</td>
<td>86</td>
</tr>
<tr>
<td>Null Renderer</td>
<td>86</td>
</tr>
<tr>
<td>Object ID Renderer</td>
<td>86</td>
</tr>
<tr>
<td>Contents</td>
<td></td>
</tr>
<tr>
<td>------------------------------------------------------------------------</td>
<td>---</td>
</tr>
<tr>
<td>The Round Number Renderer</td>
<td>87</td>
</tr>
<tr>
<td>System Up Time Renderer</td>
<td>87</td>
</tr>
<tr>
<td>Using Expressions</td>
<td>87</td>
</tr>
<tr>
<td>Manipulating Attribute Information</td>
<td>88</td>
</tr>
<tr>
<td>Precautions for Using Expressions</td>
<td>89</td>
</tr>
<tr>
<td>Referencing XML Files</td>
<td>89</td>
</tr>
<tr>
<td>Referencing Images</td>
<td>90</td>
</tr>
<tr>
<td><strong>Index</strong></td>
<td>91</td>
</tr>
</tbody>
</table>
Preface

This guide is intended for administrators who want to customize the OneClick Console using XML. The information and instructions in this document assume that the reader has thorough knowledge of XML and the concepts used to build XML files.

What is in this book

This guide contains the following chapters:

- **Chapter 1: OneClick Directory Structure** - This chapter discusses the directory structure of the XML files used to create the OneClick interface. You must be familiar with the structure to find the files necessary for customization and to implement customizations in directories that will not be overwritten when your installation of SPECTRUM is upgraded.

- **Chapter 2: Customizing the OneClick Console Menu** - This chapter describes how you can to add menus and/or menu items to the console menu in order to launch third-party applications, scripts, or web pages.

- **Chapter 3: Customizing Tables** - This chapter describes how you can customize or add columns to some of the tables available in the OneClick Console.

- **Chapter 4: Adding Support for Model Types** - This chapter describes how to create model icons that can be used with new model types created using SPECTRUM’s GnSNMPDev model type.

- **Chapter 5: Customizing a Model’s Information View** - This chapter describes how to add new information and subviews to a model type’s Information view.

- **Chapter 6: Creating Custom Privileges** - This chapter describes how to define new privileges, which can be applied to menu items or subviews.

- **Chapter 7: XML Usage Common to All Customization Files** - This chapter discusses XML elements and implementation strategies that can be used across all customization files.
Text Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Element</th>
<th>Convention Used</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variables</td>
<td>Courier and Italic in angle brackets (&lt;&gt;)</td>
<td>Type the following: DISPLAY=&lt;workstation name&gt;:0.0 export display</td>
</tr>
<tr>
<td>The directory where you installed SPECTRUM</td>
<td>&lt;$SPECROOT&gt;</td>
<td>Navigate to: &lt;$SPECROOT&gt;/app-defaults</td>
</tr>
<tr>
<td>Solaris and Windows directory paths</td>
<td>Unless otherwise noted, directory paths are common to both operating systems, with the exception that slashes (/) should be used in Solaris paths, and backslashes () should be used in Windows paths.</td>
<td>&lt;$SPECROOT&gt;/app-defaults on Solaris is equivalent to &lt;$SPECROOT&gt;/app-defaults on Windows.</td>
</tr>
<tr>
<td>On-screen text</td>
<td>Courier</td>
<td>The following line displays: path=&quot;/audit&quot;</td>
</tr>
<tr>
<td>User-typed text</td>
<td>Courier</td>
<td>Type the following path name: C:\ABC\lib\db</td>
</tr>
<tr>
<td>References to SPECTRUM documents (title and number)</td>
<td>Italic</td>
<td>SPECTRUM Installation Guide (5136)</td>
</tr>
</tbody>
</table>

Document Feedback

Please send feedback regarding SPECTRUM documents to the following e-mail address:

Spectrum-docs@aprisma.com

Thank you for helping us improve our documentation.

Online Documents

SPECTRUM documents are available online at:

http://support.concord.com/support/secure/products/Spectrum_Doc/

Check this site for the latest updates and additions.
Chapter 1: OneClick Directory Structure

This chapter explains the directory structure of the XML files used to create the OneClick interface. You must be familiar with the structure to find the files necessary for customization and to implement customizations in directories that will not be overwritten when your installation of SPECTRUM is upgraded.

In this chapter

- “Existing OneClick Files” on page 9
- “Using the Custom Directory to Preserve Customizations” on page 10

Existing OneClick Files

There are multiple files that work together to create the OneClick interface. Some of these files are found in various directories under the <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF directory.

When you customize the OneClick interface, you must not modify any of these files directly (see “Using the Custom Directory to Preserve Customizations” on page 10). However, you may need to access some existing files to copy sections of code or to use as examples.

The console/config Directory

The <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config directory contains baseline files that you can use to create customizations. Before modifying these files, you must copy them to the <Install>/WebApps/custom directory. See “Using the Custom Directory to Preserve Customizations” on page 10 for more information.

The topo/config Directory

The files in this directory create the components of the OneClick topology views. This includes files that create icons and subviews. You cannot move or copy the files in this directory to modify them. OneClick reads the files in this directory only, and will not find them if they are in the /custom directory structure.
Chapter 1: OneClick Directory Structure

The common/config Directory

The files in this directory create various topology elements that can be used by all of the other files that create the OneClick interface. This includes colors, columns for tables, and tables.

The alarm/config Directory

The files in this directory create the OneClick Alarm view.

Using the Custom Directory to Preserve Customizations

In order to ensure that customized OneClick support files are preserved during a OneClick upgrade, a special custom directory located at `<Install>/WebApps/custom` is available for user-customized OneClick files.

The OneClick support files are located in the directory structure under `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF`. Most of the files can be copied to the corresponding directory under `<Install>/WebApps/custom/` for modification.

Example:

To create a new OneClick menu item, you must copy the `custom-menu-config.xml` file from the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/` directory to the `<Install>/WebApps/custom/console/config/` directory. You are creating a directory structure that mirrors the directory structure of the original file with respect to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF` directory. Refer to “The custom-menu-config.xml File” on page 11 for details on creating custom OneClick menu items.

You cannot move or copy any of the `*app-config.xml` files to a corresponding custom directory except the `custom-app-config.xml` file. These files differ from the other XML files because OneClick always reads these files from the installed directory only.

The `custom-app-config.xml` file is a template file that you use to customize OneClick functionality. For example you can define your own information view for a particular model type. Refer to “Creating or Modifying an Information View” on page 63 for details.

Before copying the `custom-app-config.xml` file to the `<Install>/WebApps/custom/console/config/` directory, you must first verify a `custom-app-config.xml` file does not already exist in the custom directory.

OneClick creates a copy of this file in the customer directory structure if one does not exist, to store custom search information. This information is lost if you overwrite the file.

**Note:** Customized OneClick support files may be lost in the following situation:

Uninstalling and then reinstalling the same version of SpectroSERVER if you have installed OneClick under the SPECTRUM installation directory. In this case, you should save off the customized files to an area unaffected by the uninstall process, and re-insert them once you have reinstalled SpectroSERVER.
Chapter 2: Customizing the OneClick Console Menu

This chapter describes how to add new menus and new menu items to the OneClick console. You can use new menu items to launch and pass parameters to URLs, third-party applications, and scripts.

In this chapter
- “The custom-menu-config.xml File”
- “Adding a New Menu” on page 12
- “Adding a New Menu Item” on page 13

The custom-menu-config.xml File

The <Install>WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/custom-menu-config.xml file contains examples on how to add custom menus and custom menu items to your OneClick console (see Figure 2-2 on page 12 and Figure 2-1 on page 12).

As explained in the chapter “OneClick Directory Structure” on page 9, you cannot directly modify any of the customizable XML files in their original installed location. Instead, you must define all OneClick customizations in the appropriate subdirectory of the <Install>/WebApps/custom directory. Specific instructions are provided in the follow procedure.

You create OneClick menus and menu items using <menu> and <item> XML elements. The <menu> element can enclose one or more <item> elements, which define the commands that will be available on the menu. The <item> element can enclose several other elements that define how the item appears and behaves.

The following sections describe how to use the <menu> element to create a new menu and how to use the <item> element and its child elements to add menu items to a new or existing menu.
Chapter 2: Customizing the OneClick Console Menu

Figure 2-1: Adding a New Menu

Adding a New Menu

The `<menu>` element is used to create a OneClick console menu as shown in Figure 2-1.

To create a new menu item:

1. Open the `<Install>/WebApps/custom/console/config` directory and check to see if the custom-menu-config.xml file already exists there. If it does not, go to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/` directory and find the custom-menu-config.xml file. Copy it and paste it into the `<Install>/WebApps/custom/console/config` directory. As explained in the chapter “OneClick Directory Structure” on page 9, you must define OneClick customizations in the appropriate subdirectory of the `<Install>/WebApps/custom` directory.

2. Open the `<Install>/WebApps/custom/console/config/custom-menu-config.xml` file. The `<root>` element is the root element for this file. You must define all new menus inside the `<root>` element.

3. Use the `<menu>` element to create each new menu. This element has one attribute, name, which defines the name of the menu. Note that some of the examples in the custom-menu-config.xml file.
file show a fully qualified menu name that references a Java class created by OneClick engineers. For example, com.aprisma.spectrum.app.swing.window.menu.Tool is used as the value for the name attribute in the <menu> element that defines the Tools menu. It is not necessary for you to use a fully qualified name to create a new or refer to an existing menu. Simply use the exact text that you would like to appear as the menu name on the toolbar.

For example, the following lines of XML would create the Connections menu shown in Figure 2-1 on page 12. If a value for the name attribute is not specified, a menu will not be added to menu bar.

```xml
<menu name="Connections">
    <item name="Ping Local">
    .
    .
    .
    </item>
    <item name="Launch Diagnostics">
    .
    .
    .
    </item>
</menu>
```

4. Once you have created a menu, add the appropriate items to the menu using the <item> element and its available child elements. Instructions for this can be found in "Adding a New Menu Item" on page 13. If no menu items are specified for a menu, the menu will not be visible.

5. Save the changes you have made to the custom-menu-config.xml file.

6. To view the new menus, you must restart the OneClick Client.

Adding a New Menu Item

To add an item to a menu, you must nest an <item> element inside the <menu> element. The <item> element uses the name attribute to define the name of the menu item.

**Note:** The item will also automatically be added to the right-click menu.

1. Open the <Install>/WebApps/custom/console/config directory and check to see if the custom-menu-config.xml file already exists there. If it does not, go to the <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/ directory and find the custom-menu-config.xml file. Copy it and paste it into the <Install>/WebApps/custom/console/config directory. As explained in the chapter “OneClick Directory Structure” on page 9, you must define OneClick customizations in the appropriate subdirectory of the <Install>/WebApps/custom directory.

2. Open the <Install>/WebApps/custom/console/config/custom-menu-config.xml file. Find the <menu> element that defines the menu you would like to add items to. If the <menu> item does not yet exist, add it using the name attribute to define either an existing or a new menu. Note that some of the examples in the custom-menu-config.xml file show a fully qualified menu name that references a Java class created by OneClick engineers. For example, com.aprisma.spectrum.app.swing.window.menu.Tool is used as the value for the name attribute in the <menu> element that defines the Tools menu. It is not necessary for you to use
a fully qualified name to create a new or referring to an existing menu. For example, you can use `<menu name="Tools">` to refer to the Tools menu.

3. Use the `<item>` element to create each new menu item. It is a child element of the `<menu>` element. This element has one attribute, name, which defines the name of the menu item. The following example shows a menu item named “Ping Local” added to a menu called “Connections”.

```xml
<menu name="Connections">
    <item name="Ping Local">
        <accelerator modifiers="2">VK_I</accelerator>
        <action>
            <has-attribute>AttributeID.NETWORK_ADDRESS</has-attribute>
        </filter>
        <context>com.aprisma.spectrum.app.topo.client.render.ModelContext</context>
        <context>com.aprisma.spectrum.app.alarm.client.group.AlarmContext</context>
        <launch-application>
            <platform>
                <os-name>Windows 9x</os-name>
                <command>command.com /c start "Local ping {0}" cmd.exe /c "ping.exe {0} && pause"</command>
            </platform>
            <platform>
                <os-name>Windows</os-name>
                <command>cmd.exe /c start "Local ping {0}" cmd.exe /c "ping.exe {0} && pause"</command>
            </platform>
            <platform>
                <command>/usr/dt/bin/dtterm -e ping -s {0}</command>
            </platform>
        </launch-application>
    </item>
</menu>
```

4. The `<item>` element has a series of child elements that enable you to define how the item behaves. These elements are listed in Table 2-1 on page 15 and are further defined in the rest of this chapter. Use these elements to define the behavior of the menu item you have added.

5. Save the changes you have made to the `custom-menu-config.xml` file.

6. To view the new menus, you must restart the OneClick Client.
Table 2-1: Elements Used to Define a Menu Item

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;menu&gt;</td>
<td>&lt;root&gt;</td>
<td>Defines the menu. The name attribute is used to define the name of the menu. See &quot;Adding a New Menu&quot; on page 12 for more information.</td>
</tr>
<tr>
<td>&lt;separator&gt;</td>
<td>&lt;menu&gt;</td>
<td>Used just before an &lt;item&gt; element to define a separator line as shown in Figure 2-1 on page 12.</td>
</tr>
<tr>
<td>&lt;item&gt;</td>
<td>&lt;menu&gt;</td>
<td>Defines an item on a specific menu. The name attribute is used to define the name of the item.</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;item&gt;</td>
<td>Associates a privilege to the menu item. If the user is not given this privilege, the menu item will not be displayed for that user. See &quot;Creating Custom Privileges&quot; on page 77 for more information.</td>
</tr>
<tr>
<td>&lt;toolbar-image&gt;</td>
<td>&lt;item&gt;</td>
<td>Defines the image that will represent this functionality on the toolbar and the menu. See &quot;Toolbar Images&quot; on page 15 for more information.</td>
</tr>
<tr>
<td>&lt;toolbar-image-rollover&gt;</td>
<td>&lt;item&gt;</td>
<td>Defines the image on the toolbar that will be exposed when the user mouses over the toolbar-image. See &quot;Toolbar Images&quot; on page 15 for more information.</td>
</tr>
<tr>
<td>&lt;toolbar-image-disabled&gt;</td>
<td>&lt;item&gt;</td>
<td>Defines the image on the toolbar that will represent this functionality when the functionality is disabled. See &quot;Toolbar Images&quot; on page 15 for more information.</td>
</tr>
<tr>
<td>&lt;accelerator&gt;</td>
<td>&lt;item&gt;</td>
<td>Defines a keyboard sequence that executes the menu item. See &quot;Keyboard Accelerator&quot; on page 16 for more information.</td>
</tr>
<tr>
<td>&lt;action&gt;</td>
<td>&lt;item&gt;</td>
<td>Defines the action that takes place when the user clicks on the menu item. See &quot;Performing an Action&quot; on page 16 for more information.</td>
</tr>
<tr>
<td>&lt;hot-key&gt;</td>
<td>&lt;item&gt;</td>
<td>Underlines the first instance of the indicated letter and enables the user to activate the menu item using this letter as a keyboard shortcut.</td>
</tr>
</tbody>
</table>

Toolbar Images

The <toolbar-image> element specifies an image for a toolbar button and menu item. If this element is not specified, a toolbar button is not created for the item.

The <toolbar-image-rollover> element specifies an image for the tool bar button when the mouse is moved over it.

The <toolbar-image-disabled> element specifies an image for the tool bar button when it is disabled.

You can use the following image types when defining an image:

- .png
- .gif
- .jpg
- .jpeg
It is recommended that you use an image size of 24 x 24 pixels. You must store images used for customization purposes in the \texttt{/WebApps/tomcat/webapps/spectrum/images} directory. When you reference an image placed in this directory you must express the path from the images directory, i.e. \texttt{images/myimage.png}.

The following example shows the image file \texttt{hints.gif} used to specify an image for the toolbar button. Note that the relative path to the image file is specified.

\texttt{<toolbar-image>images/hints.gif</toolbar-image>}

**Keyboard Accelerator**

The \texttt{<accelerator>} element specifies the accelerator key for the menu item.

\texttt{<accelerator modifiers="2">VK_L</accelerator>}

The text specifies the code for the key. It must contain the capitalized letter preceded by "VK_". The "modifiers" attribute indicates the modifier key combinations as an integer where:

- 1 = Shift
- 2 = Ctrl
- 3 = Ctrl+Shift
- 8 = Alt
- 9 = Alt+Shift
- 10 = Ctrl+Alt

An accelerator does not need to be specified.

In the above example, the menu item's specified action (see “Performing an Action” on page 16) is performed if the 'L' key is pressed while holding down the Control key (Ctrl+L).

**Performing an Action**

The \texttt{<action>} element specifies the action that is performed when the menu item is selected.

You can use the child elements shown in Table 2-2 on page 17 to specify a particular action.
Adding a New Menu Item

### Table 2-2: Elements that Implement an Action

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;context&gt;</code></td>
<td><code>&lt;action&gt;</code></td>
<td>The context in which the menu item will perform the action. See &quot;Contextually Applying the Action&quot; on page 17.</td>
</tr>
<tr>
<td><code>&lt;filter&gt;</code></td>
<td><code>&lt;action&gt;</code></td>
<td>See &quot;Limiting the Availability of the Menu Item&quot; on page 18.</td>
</tr>
<tr>
<td><code>&lt;has-attribute&gt;</code></td>
<td><code>&lt;filter&gt;</code></td>
<td>Used to specify the attribute to filter on. See &quot;Limiting the Availability of the Menu Item&quot; on page 18.</td>
</tr>
<tr>
<td><code>&lt;and&gt;</code>, <code>&lt;or&gt;</code>, <code>&lt;value&gt;</code>, <code>&lt;equals&gt;</code></td>
<td><code>&lt;filter&gt;</code></td>
<td>Used to create an expression that can be used with a filter. See &quot;Limiting the Availability of the Menu Item&quot; on page 18.</td>
</tr>
<tr>
<td><code>&lt;launch-browser&gt;</code></td>
<td><code>&lt;action&gt;</code></td>
<td>Launches a browser. See &quot;Launching a Browser&quot; on page 21 for more information.</td>
</tr>
<tr>
<td><code>&lt;url&gt;</code></td>
<td><code>&lt;launch-browser&gt;</code></td>
<td>Used to specify the URL that should be launched in the browser. See &quot;Launching a Browser&quot; on page 21 for more information.</td>
</tr>
<tr>
<td><code>&lt;launch-application&gt;</code></td>
<td><code>&lt;action&gt;</code></td>
<td>Launches an application. See &quot;Launching an Application&quot; on page 22 for more information.</td>
</tr>
<tr>
<td><code>&lt;launch-web-server-script&gt;</code></td>
<td><code>&lt;action&gt;</code></td>
<td>Launches a script available on the web server. See &quot;Launching a Web Server Script&quot; on page 23 for more information.</td>
</tr>
<tr>
<td><code>&lt;command&gt;</code></td>
<td><code>&lt;launch-application&gt;</code>, <code>&lt;launch-web-server-script&gt;</code>, <code>&lt;platform&gt;</code></td>
<td>Used to specify the application or script that the menu item launches. See &quot;Launching a Browser&quot; on page 21, &quot;Launching an Application&quot; on page 22, and &quot;Launching a Web Server Script&quot; on page 23.</td>
</tr>
<tr>
<td><code>&lt;platform&gt;</code></td>
<td><code>&lt;launch-application&gt;</code></td>
<td>Used in conjunction with <code>&lt;os-name&gt;</code> to specify the application to be launched specific to the operating system of the OneClick client. See &quot;Launching an Application&quot; on page 22.</td>
</tr>
<tr>
<td><code>&lt;os-name&gt;</code></td>
<td><code>&lt;platform&gt;</code></td>
<td>Used in conjunction with <code>&lt;os-name&gt;</code> to specify the application to be launched specific to the operating system of the OneClick client. See &quot;Launching an Application&quot; on page 22.</td>
</tr>
<tr>
<td><code>&lt;param&gt;</code></td>
<td><code>&lt;url&gt;</code>, <code>&lt;command&gt;</code></td>
<td>Used to specify a parameter that is passed to a browser, executable, or script. See &quot;Launching a Browser&quot; on page 21, &quot;Launching an Application&quot; on page 22, and &quot;Launching a Web Server Script&quot; on page 23.</td>
</tr>
<tr>
<td><code>&lt;attribute&gt;</code></td>
<td><code>&lt;param&gt;</code></td>
<td>Used to specify an attribute used as a parameter.</td>
</tr>
</tbody>
</table>

### Contextually Applying the Action

The `<context>` element specifies the context in which the menu item will be active so that the action can be executed. This applies to both the standard and the right-click menu.
You can specify a ModelContext, which indicates that the action should be available when the user selects a model. You can also specify an AlarmContext, which indicates that the action should be available when the user selects an alarm.

```
<context>com.aprisma.spectrum.app.topo.client.render.ModelContext</context>
<context>com.aprisma.spectrum.app.alarm.client.group.AlarmContext</context>
```

You can specify one or both of the contexts. If no contexts are specified then the menu item will be displayed in all contexts. If no specified contexts match the current window context, then the menu item is disabled.

**Limiting the Availability of the Menu Item**

The `<filter>` element specifies a filter that further restricts the enabled state of the menu item. You can filter on any attribute of the selected context.

```
<has-attribute>AttributeID.NETWORK_ADDRESS</has-attribute>
```

In the above example, the action needs the IP address of the alarmed model. Therefore it should only be enabled if the alarmed model has the Network_Address (ID 0x1027f) attribute.

You can specify complex attribute filters with any combination of nested “and” / “or” filters. For example:

```
<filter>
  <and>
    <has-attribute>AttributeID.NETWORK_ADDRESS</has-attribute>
    <equals>
      <attribute id="AttributeID.CONDITION">
        <value>3</value><!--red-->
      </attribute>
    </equals>
  </and>
</filter>
```

The example above enables the item if the selected model has the Network_Address attribute and the Condition (ID 0x1000a) attribute is RED.

The file `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/common/schema/attribute-filter.xsd` contains the complete syntax for attribute filters.
You can use the following constants for commonly-used attributes where an attribute ID is expected:

Table 2-3: Constants for Commonly Used Attributes

<table>
<thead>
<tr>
<th>Constant</th>
<th>Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID.NETWORK_ADDRESS</td>
<td>Network Address (ID 0x1027f)</td>
</tr>
<tr>
<td>AttributeID.MTYPE_ID</td>
<td>Model Type Handle (ID 0x129ab)</td>
</tr>
<tr>
<td>AttributeID.MTYPE_NAME</td>
<td>Model Type Name (ID 0x10000)</td>
</tr>
<tr>
<td>AttributeID.MODEL_OBJECT</td>
<td>Model Handle (ID 0x11f53)</td>
</tr>
<tr>
<td>AttributeID.MODEL_NAME</td>
<td>Model Name (ID 0x1006e)</td>
</tr>
<tr>
<td>AttributeID.MODEL_CLASS</td>
<td>Model Class (ID 0x11ee8)</td>
</tr>
<tr>
<td>AttributeID.CONDITION</td>
<td>Condition (ID 0x1000a)</td>
</tr>
<tr>
<td>AttributeID.DOMAIN_ID</td>
<td>Landscape Handle (ID 0x129ac)</td>
</tr>
<tr>
<td>AttributeID.DOMAIN_NAME</td>
<td>Landscape Name (ID 0x11d42)</td>
</tr>
<tr>
<td>AttributeID.MAC_ADDRESS</td>
<td>MAC Address (ID 0x110df)</td>
</tr>
<tr>
<td>AttributeID.DEVICE_TYPE</td>
<td>Device Type (ID 0x23000e)</td>
</tr>
</tbody>
</table>
You can use the following constants for alarm attributes:

**Figure 2-3: Constants for Commonly Used Alarm Attributes**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Alarm Attribute</th>
</tr>
</thead>
<tbody>
<tr>
<td>AlarmAttrID.ACKNOWLEDGED</td>
<td>Acknowledged (ID 0x11f4d)</td>
</tr>
<tr>
<td>AlarmAttrID.ALARM_FILTER_MH</td>
<td>Alarm Filter (ID 0x12a56)</td>
</tr>
<tr>
<td>AlarmAttrID.ALARM_ID</td>
<td>Full Alarm ID (ID 0x11f9c)</td>
</tr>
<tr>
<td>AlarmAttrID.INT_ALARM_ID</td>
<td>Integer Alarm ID (ID 0x4820067)</td>
</tr>
<tr>
<td>AlarmAttrID.ALARM_SOURCE</td>
<td>Alarm Source (ID 0x11fc4)</td>
</tr>
<tr>
<td>AlarmAttrID.ALARM_STATUS</td>
<td>Alarm Status (ID 0x11fc4)</td>
</tr>
<tr>
<td>AlarmAttrID.CAUSE_CODE</td>
<td>Cause Code (ID 0x11f50)</td>
</tr>
<tr>
<td>AlarmAttrID.CAUSE_LIST</td>
<td>Cause List (ID 0x12a05)</td>
</tr>
<tr>
<td>AlarmAttrID.CAUSE_TITLE</td>
<td>Cause Title (ID 0x4820020)</td>
</tr>
<tr>
<td>AlarmAttrID.CREATION_DATE</td>
<td>Creation Date (ID 0x11f4e)</td>
</tr>
<tr>
<td>AlarmAttrID.CLEARED_BY_USER_NAME</td>
<td>Cleared By User Name (ID 0x11f51)</td>
</tr>
<tr>
<td>AlarmAttrID.IMPACT_SEVERITY</td>
<td>Impact Severity (ID 0x1290d)</td>
</tr>
<tr>
<td>AlarmAttrID.OCCURRENCES</td>
<td>Occurrences (ID 0x11fc5)</td>
</tr>
<tr>
<td>AlarmAttrID.ORIGINATING_EVENT</td>
<td>Originating Event (ID 0x1296e)</td>
</tr>
<tr>
<td>AlarmAttrID.PERSISTENT</td>
<td>Persistent (ID 0x12942)</td>
</tr>
<tr>
<td>AlarmAttrID.PRIMARY_ALARM</td>
<td>Primary Alarm (ID 0x11f54)</td>
</tr>
<tr>
<td>AlarmAttrID.SEVERITY</td>
<td>Severity (ID 0x11f56)</td>
</tr>
<tr>
<td>AlarmAttrID.TROUBLESHOOTER</td>
<td>Troubleshooter (ID 0x11f57)</td>
</tr>
<tr>
<td>AlarmAttrID.TROUBLE_TICKET_ID</td>
<td>Trouble Ticket ID (ID 0x12022)</td>
</tr>
<tr>
<td>AlarmAttrID.USER_CLEARABLE</td>
<td>User Clearable (ID 0x11f9b)</td>
</tr>
</tbody>
</table>

If you need to use an attribute other than one of the attributes listed in Table 2-3 on page 19 or Table 3-2 on page 32, specify the attribute using its hexadecimal attribute ID.
Launching a Browser

The `<launch-browser>` element enables you to launch a specified URL in a browser and pass parameters to the URL. These parameters can be hard-coded values or values from model attributes.

```xml
<launch-browser>
  <url>http://{0}</url>
  <param>
    <attribute>AttributeID.NETWORK_ADDRESS</attribute>
  </param>
</launch-browser>
```

The example above launches the default browser on the client machine. The `<url>` element specifies the URL pattern. You can specify parameters to substitute in the URL pattern by enclosing the parameter number (starting at 0) in curly braces `{}`. You then specify `<param>` elements for each parameter.

SPECTRUM processes the `<param>` elements in order so the first one corresponds to the 0th parameter in the URL pattern. A `<param>` element has a specific syntax, but the most commonly-used is the `<attribute>` element. This element substitutes the value of the specified attribute for the selected context. In the above example, the value of the Network Address attribute is substituted in the URL pattern. For more complex parameters, see the definition of `<param-type>` in `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/common/schema/basic-config.xsd`.

Specifying a Username

Use the following expression to specify the logged-in user’s username:

```xml
<param>
  <expression>
  </expression>
</param>
```

You can use the above example to pass the current user’s username to an application, web-browser, or some other executable that requires a username.

For example, you can launch a browser to a specified URL, and pass the username parameter along to the browser to minimize the amount of logging in or typing in of logon information required of users.

```xml
<launch-browser>
  <url>http://acme.com?user={0}</url>
  <param>
    <expression>
    </expression>
  </param>
</launch-browser>
```
Launching an Application

The `<launch-application>` element enables you to launch a specified executable and pass attribute values to that executable.

```xml
<launch-application>
  <command>myapp {0}</command>
  <param>
    <attribute>AttributeID.NETWORK_ADDRESS</attribute>
  </param>
</launch-application>
```

The above example launches an application on the client machine. The `<command>` element specifies the command or executable to execute. You can provide the path to the command or executable in one of two ways:

- You can specify the path on each client via an environment variable. To do this in the Solaris environment, use the PATH environment variable. To create an environment variable in the Windows environment, select `My Computer > Properties > Advanced` and then choose the Environment Variables button.
- You can specify and absolute path to the executable or command. If you do this, keep in mind that the path must be the same on each OneClick client. Path statements in the Windows environment should use a double backslash instead of a single backslash, for example:

  ```
  C:\\Windows\\system32\\cmd.exe
  ```

As with the `<launch-browser>` action, you can substitute any number of parameters. The above example launches an application called myapp and passes the IP address of the selected model.

The `<command>` element must conform to the following syntax rules:

- The command arguments are delimited by only spaces. If you would like to have a space within an argument, you must either place quotes around the argument or use the escape character `\` prior to the internal space(s).
- If you would like to imbed quotes within an argument, you must place the escape character `\` prior to the quote.
- If any of your command arguments contain commas, SPECTRUM will automatically place the argument within quotes. This is important to know in case you are going to parse an argument that is a numeric value that contains commas.
- SPECTRUM replaces arguments that come back null or have a string length of zero with empty quotes. (i.e. `""`).

The following example uses the `<platform>` element to specify different commands for different platforms. The `<os-name>` element specifies the operating system name and the `<command>` element specifies the command to execute on that operating system. The `<os-name>` element is optional. If you do not specify the `<os-name>`, the associated command is the default such that if no other platforms match, the default command is executed.
Adding a New Menu Item

At runtime, SPECTRUM compares the specified OS names to the OS name returned by the "os.name" Java property. SPECTRUM uses a best-match algorithm so only a prefix of the OS name need be specified. You may specify any of the following OS names:

- SunOS for the Solaris platform
- Windows for all Windows platforms
- Windows 9x for Windows 95/98 only
- Windows 2000 for just Windows 2000 only
- Windows XP for Windows XP only
- Linux for the Linux platform
- Mac for the Macintosh platform

If no specified platforms match, the associated menu item will be disabled.

Launching a Web Server Script

The `<launch-web-server-script>` element launches a script on the web server machine. The `<command>` element specifies the script to execute. As with the `<launch-browser>` action, any number of parameters can be substituted. Since the script resides on the web server, which is restricted to either a Windows or Solaris machine, you do not use a `<platform>` element to denote the platform that the script is running on.

**Note:** This action can only be used to launch a script, it cannot be used to launch a user interface.

The following example launches "myscript" on the web server passing it the model name and model type name of the selected model. Note that the path shown in the `<command>` element is a path for a Windows web server.

```xml
<launch-web-server-script>
  <command>c:/scripts/myscript {0} {1}</command>
  <param>
    <attribute>AttributeID.MODEL_NAME</attribute>
  </param>
  <param>
    <attribute>AttributeID.MTYPE_NAME</attribute>
  </param>
</launch-web-server-script>
```
Chapter 3: Customizing Tables

This chapter discusses various ways in which you can modify existing tables found in the OneClick console.

In this chapter

- “Modifying Table Columns”
- “Making a Column Editable” on page 29
- “Setting up a Default Sort” on page 31
- “Customizing the Port Name Column of the Interface Table” on page 32
- “Sorting the Interfaces Table by ifIndex” on page 33

Modifying Table Columns

If you wish to display additional attributes in one of the tables found in the OneClick console, you can do so by modifying the XML file that constructs the particular table you are interested in modifying.

Procedure

1. Identify the appropriate XML file that is used to build the table. Many of the table files that are used to display data in the OneClick console are located in the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config` directory. All of the table files are named after the functionality that they display. For example, the table used to display interface information for each model is called `table-common-ifconfig-config.xml`.

2. Make sure that the table file you would like to customize does not already exist in the `<Install>/WebApps/custom/topo/config` directory. (It would exist in this directory if previous customizations have been made to this file.) If it does not exist there, copy the table file into the `<Install>/WebApps/custom/topo/config` directory. If this directory structure does not yet exist, you will need to create it. (See “OneClick Directory Structure” on page 9 for more information.)

3. Open the file in a text editor in order to make the appropriate modifications.
4. You can construct a new column using the XML elements defined in Table 3-1 on page 27. The following example shows how some of these elements are used to define a column:

```xml
<column-list>
  <column>
    <name>Interface</name>
    <content>
      <attribute>0x100c4</attribute>
    </content>
    <default-width>30</default-width>
  </column>
</column-list>
```

5. The `<column-list>` element contains all of the `<column>` elements used to define each column in the table. Find the `<column-list>` element in the XML file.

6. Define a `<column>` element within the `<column-list>` element. The columns will display in the table in the order in which they are defined within the `<column-list>` element.

7. Insert the `<name>` element to define the title of the column.

8. Insert a `<content>` and an `<attribute>` element to define what will be displayed in the column.

9. Use the `<default-width>` element to define the default width of the column.

10. Save and close the modified file. You will need to restart the OneClick client for the changes to take effect.
Table 3-1: Elements Used for Modifying a Table

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;table/&gt;</td>
<td>NA</td>
<td>This is the root element, and encloses all child elements used to create a table.</td>
</tr>
<tr>
<td>&lt;swing-table-template/&gt;</td>
<td>&lt;table/&gt;</td>
<td>Used to define the appearance of the table.</td>
</tr>
<tr>
<td>&lt;show-vertical-lines/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines whether to show vertical lines in the table, values are true or false.</td>
</tr>
<tr>
<td>&lt;show-horizontal-lines/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines whether to show horizontal lines in the table, values are true or false.</td>
</tr>
<tr>
<td>&lt;line-color/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines the color of the lines used to create the table. The default value is light-background_color. Other values can be found in the &lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/common/config/common-color-config.xml file.</td>
</tr>
<tr>
<td>&lt;show-tree-lines/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines whether the table will be shown with dashed lines connecting tree nodes, values are true or false.</td>
</tr>
<tr>
<td>&lt;preferred-width/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines (in pixels) the default width of the table.</td>
</tr>
<tr>
<td>&lt;preferred-height/&gt;</td>
<td>&lt;swing-table-template/&gt;</td>
<td>Defines (in pixels) the default height of the table.</td>
</tr>
<tr>
<td>&lt;swing-header-row-template/&gt;</td>
<td>&lt;table/&gt;</td>
<td>Used to define the appearance of the header row of the table.</td>
</tr>
<tr>
<td>&lt;static-color/&gt;</td>
<td>&lt;swing-header-row-template/&gt;</td>
<td>Specifies the color for the header row. Use value idref=row-header-color-config for consistency.</td>
</tr>
<tr>
<td>&lt;swing-row-template/&gt;</td>
<td>&lt;table/&gt;</td>
<td>Used to define the appearance for the body rows in the table.</td>
</tr>
<tr>
<td>&lt;enumerated-color/&gt;</td>
<td>&lt;swing-row-template/&gt;</td>
<td>Used to specify different colors for each row of the table, the default value used is alternatingrow-color-config.</td>
</tr>
<tr>
<td>&lt;static-color/&gt;</td>
<td>&lt;swing-row-template/&gt;</td>
<td>Used to specify a single color used for all of the rows in the table.</td>
</tr>
<tr>
<td>&lt;column-list/&gt;</td>
<td>&lt;table/&gt;</td>
<td>Used to define the list of columns to be used in the table.</td>
</tr>
<tr>
<td>&lt;column/&gt;</td>
<td>&lt;column-list/&gt;</td>
<td>Used to define a column in the column list.</td>
</tr>
<tr>
<td>&lt;name/&gt;</td>
<td>&lt;column/&gt;</td>
<td>The name of the column. The text used here will appear in the table header for this column.</td>
</tr>
<tr>
<td>&lt;editable/&gt;</td>
<td>&lt;column/&gt;</td>
<td>Defines whether the value in the table can be edited. If this value is set to true, a set link appears next to the value.</td>
</tr>
<tr>
<td>&lt;content/&gt;</td>
<td>&lt;column/&gt;</td>
<td>Defines the value placed in the column.</td>
</tr>
<tr>
<td>&lt;renderer/&gt;</td>
<td>&lt;content/&gt;</td>
<td>Specifies the renderer to be used for the content of the column. See &quot;Customizing the Port Name Column of the Interface Table&quot; on page 32 for more information.</td>
</tr>
<tr>
<td>&lt;dynamic-renderer/&gt;</td>
<td>&lt;content/&gt;</td>
<td>Enables you to specify a renderer depending on model class or model type.</td>
</tr>
<tr>
<td>Element</td>
<td>Parent Element</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>&lt;expression&gt;</td>
<td>&lt;content&gt;</td>
<td>Used to define an expression to produce a value for the column. See “XML Usage Common to All Customization Files” on page 83 for more information.</td>
</tr>
<tr>
<td>&lt;message&gt;</td>
<td>&lt;content&gt;</td>
<td>Used for specifying a plain text value for the column.</td>
</tr>
<tr>
<td>&lt;select&gt;</td>
<td>&lt;content&gt;</td>
<td>Used to select a value for the column based on certain criteria. See “XML Usage Common to All Customization Files” on page 83 for more information.</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>&lt;content&gt;</td>
<td>Used to specify an attribute ID. The value of the attribute will be placed in the column.</td>
</tr>
<tr>
<td>&lt;swing-cell-template&gt;</td>
<td>&lt;column&gt;</td>
<td>Used to define how the cell in the column is displayed.</td>
</tr>
<tr>
<td>&lt;image&gt;</td>
<td>&lt;swing-cell-template&gt;</td>
<td>Used to display an image in a cell.</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>&lt;image&gt;</td>
<td>The attribute used to determine the image selection.</td>
</tr>
<tr>
<td>&lt;select&gt;</td>
<td>&lt;image&gt;</td>
<td>Used to define the image that is selected. See “XML Usage Common to All Customization Files” on page 83 for more information.</td>
</tr>
<tr>
<td>&lt;text&gt;</td>
<td>&lt;swing-cell-template&gt;</td>
<td>Used to display a string of text in a cell.</td>
</tr>
<tr>
<td>&lt;renderer&gt;</td>
<td>&lt;text&gt;</td>
<td>The renderer used to manipulate the text. See “Customizing the Port Name Column of the Interface Table” on page 32 for more information.</td>
</tr>
<tr>
<td>&lt;param&gt;</td>
<td>&lt;renderer&gt;</td>
<td>Specifies any parameters to be used by the renderer.</td>
</tr>
<tr>
<td>&lt;renderer-class&gt;</td>
<td>&lt;swing-cell-template&gt;</td>
<td>The class to use for rendering something other than an image or text.</td>
</tr>
<tr>
<td>&lt;editable&gt;</td>
<td>&lt;column&gt;</td>
<td>Allows the user to edit the value in a column. See “Making a Column Editable” on page 29 for more information.</td>
</tr>
<tr>
<td>&lt;hidden-by-default&gt;</td>
<td>&lt;column&gt;</td>
<td>The column does not show up in the table by default. The user must add the column to the table using the preferences dialog.</td>
</tr>
<tr>
<td>&lt;default-width&gt;</td>
<td>&lt;column&gt;</td>
<td>The default width, specified in pixels, of the column.</td>
</tr>
<tr>
<td>&lt;default-sort&gt;</td>
<td>&lt;column-list&gt;</td>
<td>Used in conjunction with the &lt;sort-column-list&gt; element to determine how the table is sorted by default. See ”Setting up a Default Sort” on page 31 for more information.</td>
</tr>
<tr>
<td>&lt;sort-column-list&gt;</td>
<td>&lt;default-sort&gt;</td>
<td>A list of columns that will be sorted on (maximum of three). See ”Setting up a Default Sort” on page 31 for more information.</td>
</tr>
<tr>
<td>&lt;sort-column&gt;</td>
<td>&lt;sort-column-list&gt;</td>
<td>The column to be sorted on. Columns will be sorted in the order that they are listed. See ”Setting up a Default Sort” on page 31 for more information.</td>
</tr>
<tr>
<td>&lt;name&gt;</td>
<td>&lt;sort-column&gt;</td>
<td>The name of the column to sort on. This must match the name defined for the column. See ”Setting up a Default Sort” on page 31 for more information.</td>
</tr>
<tr>
<td>&lt;direction&gt;</td>
<td>&lt;sort-column&gt;</td>
<td>The direction of the sort, values can be ascending or descending. See ”Setting up a Default Sort” on page 31 for more information.</td>
</tr>
</tbody>
</table>
Making a Column Editable

In some cases, you can allow the user to edit the value found in a particular table cell in the column. The Acknowledge column in the Alarm table and the Admin Status column in the Interface Configuration table are examples of columns that you might want to allow your user to edit. Figure 3-1 shows the default behavior of the Acknowledge field in the Alarm table and the new behavior if the column has been configured to be editable.

Figure 3-1: The Acknowledge Column in the Alarm Table

To make the Acknowledge column of the Alarm table editable:

1. Create the `<Install>/WebApps/custom/alarm/config` directory if it does not yet exist.

2. If the `<Install>/WebApps/custom/alarm/config` already exists, check to see if the `alarm-table-config.xml` file is in that directory. (It would exist in this directory if previous customizations have been made to this file.) If not, go to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/alarm/config` directory, copy the `alarm-table-config.xml` file and paste it into the `<Install>/WebApps/custom/alarm/config` directory.

3. Open the file and find the following section of XML, which creates the Acknowledge column:

   ```xml
   <column idref="column-alarmacknowledge-config">
   <default-width>37</default-width>
   </column>
   ```

4. Add the `<editable>` element as shown below:

   ```xml
   <column idref="column-alarmacknowledge-config">
   <default-width>37</default-width>
   <editable/>
   </column>
   ```

This shows how the Acknowledge field in the Alarm table behaves by default.

If you make the column editable, the user can set the value to Yes or No.
<column idref="column-alarmacknowledge-config">
  <editable/>
  <default-width>37</default-width>
</column>

5. Save and close the alarm-table-config.xml file.

6. Restart the OneClick client for the changes to take effect.

To make the Admin Status column of the Interface Configuration table editable:

1. Create the <Install>/WebApps/custom/topo/config directory if it does not yet exist.

2. If the <Install>/WebApps/custom/topo/config already exists, check to see if the table-common-ifconfig-config.xml file is in that directory. (It would exist in this directory if previous customizations have been made to this file.) If not, go to the <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config directory, copy the table-common-ifconfig-config.xml file and paste it into the <Install>/WebApps/custom/topo/config directory.

3. Open the file and find the following section of XML, which creates the Admin Status column:

   <column>
     <name>com.aprisma.spectrum.app.topo.client.AdminStatus</name>
     <content>
       <attribute>0x100ca</attribute>
       <renderer>
         <param name="attrID">0x100ca</param>
         com.aprisma.spectrum.app.util.render.EnumeratedAttrRenderer
       </renderer>
     </content>
     <default-width>65</default-width>
   </column>

4. Add the <editable> element as shown below:

   <column>
     <name>com.aprisma.spectrum.app.topo.client.AdminStatus</name>
     <content>
       <attribute>0x100ca</attribute>
       <renderer>
         <param name="attrID">0x100ca</param>
         com.aprisma.spectrum.app.util.render.EnumeratedAttrRenderer
       </renderer>
     </content>
     <editable/>
     <default-width>65</default-width>
   </column>

5. Save and close the table-common-ifconfig-config.xml file.

6. Restart the OneClick client for the changes to take effect.
Setting up a Default Sort

You can create a default sort criteria for a table using the `<default-sort>` element and its child elements (see Table 3-1 on page 27). You can sort on a maximum of three columns. SPECTRUM sorts the columns in the order in which they are listed in `<sort-column-list>.

To create a default sort for a table:

1. Identify the appropriate XML file that is used to build the table. All of the table files that are used to display data in the OneClick console are located in the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config` directory. All of the table files are named after the functionality that they display. For example, the table used to display interface information for each model is called `table-common-ifconfig-config.xml`.

2. Make sure that the table file you would like to customize does not already exist in the `<Install>/WebApps/custom/topo/config` directory. (It would exist in this directory if previous customizations have been made to this file.) If it does not exist there, copy the table file into the `<Install>/WebApps/custom/topo/config` directory. If this directory structure does not yet exist, you will need to create it. (See “OneClick Directory Structure” on page 9 for more information.)

3. Open the file in a text editor in order to make the appropriate modifications.

4. Find the closing tag for the column-list element, `</column-list>`.

5. Insert the XML to create the sort as shown in the example below:

   ```xml
   <table>
   <column-list>
   .
   .
   </column-list>
   <default-sort>
   <sort-column-list>
   <sort-column>
   <name>Name_of_first_column_to_sort</name>
   <direction>ascending</direction>
   </sort-column>
   <sort-column>
   <name>Name_of_second_column_to_sort</name>
   <direction>ascending</direction>
   </sort-column>
   <sort-column>
   <name>Name_of_third_column_to_sort</name>
   <direction>ascending</direction>
   </sort-column>
   </sort-column-list>
   </default-sort>
   <table>
   ```

6. Save and close the XML file.

7. Restart the OneClick client for your changes to take effect.
Customizing the Port Name Column of the Interface Table

You can change the tree column in the interface table to show something other than model name.

Procedure

1. If the `<Install>/WebApps/custom/topo/config` directory already exists, check to see if the `content-iftree-config.xml` file is in that directory. (It would exist in this directory if previous customizations have been made to this file.) If not, go to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/` directory and find the `content-iftree-config.xml` file. Copy it and paste it into the `<Install>/WebApps/custom/topo/config` directory.

   By default, the `content-iftree-config.xml` file contains the following:

   ```xml
   <attribute>AttributeID.MODEL_NAME</attribute>
   <!-- If the model name is not filled in or it's a GnPort, use Component_OID-->
   <expression>((String)value()).length() == 0 ||
   ((String)value()).equals("GnPort") ?
   attr(0x1006a).toString(): value()
   </expression>
   </content>
   ```

2. Modify the contents of the `<attribute><value></attribute>` element in the copied `<Install Area>/WebApps/custom/topo/config/content-iftree-config.xml` file to contain the attribute you want to display. By default the Model_Name attribute (`<attribute>AttributeID.MODEL_NAME</attribute>`) is displayed. To display a different attribute, change `<value>` to the desired attribute ID. You can specify the integer ID (for example, 0x129e0) or a predefined constant. Table 3-2 on page 32 displays the predefined constants with their integer IDs.

<table>
<thead>
<tr>
<th>Predefined Attribute Value Constant</th>
<th>Integer ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>AttributeID.PORT_DESCRIPTION</td>
<td>0x129e0</td>
</tr>
<tr>
<td></td>
<td>(ifDescr)</td>
</tr>
<tr>
<td>AttributeID.PORT_TYPE</td>
<td>0x129ed</td>
</tr>
<tr>
<td></td>
<td>(ifType)</td>
</tr>
<tr>
<td>AttributeID.COMPONENT_OID</td>
<td>0x1006a</td>
</tr>
</tbody>
</table>

   **Note:** The contents of the `<expression>` element are specific to the use of the Model_Name attribute and should be removed when specifying other attributes.
3. Close and restart the OneClick client to see the changes take effect.

Example

To display the port description, edit the file as follows:

```xml
<content id="content-iftree-config"
  xmlns ="http://www.aprisma.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.aprisma.com
  ../../../common/schema/column-config.xsd">
  <attribute>AttributeID.PORT_DESCRIPTION</attribute>
</content>
```

Sorting the Interfaces Table by ifIndex

SPECTRUM sorts the interfaces in the Interfaces Table by the first column, which is the interface model name. SPECTRUM sorts this field alphabetically based on the textual values.

Procedure

The following procedure gives instructions on how to sort the table numerically by ifIndex value.

1. If the `<Install>/WebApps/custom/topo/config` directory already exists, check to see if the `content-iftree-config.xml` file is in that directory. (It would exist in this directory if previous customizations have been made to this file.) If not, go to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/` directory and find the `content-iftree-config.xml` file. Copy it and paste it into the `<Install>/WebApps/custom/topo/config` directory.

2. Find the line that reads:

   `<attribute>AttributeID.MODEL_NAME</attribute>`

3. Change this line to read:

   `<attribute>0x1006a</attribute>`

   The 0x1006a attribute is Component_OID (which is rolled from ifIndex).

   **Note:** The contents of the `<expression>` element are specific to the use of the Model_Name attribute and should be removed when specifying other attributes.

4. Save your changes to the `content-iftree-config.xml` file.

5. Open the `<Install>/WebApps/custom/topo/config` directory and check to see if the `column-iftree-config.xml` file already exists there. If it does not, go to the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/` directory and find the `column-iftree-config.xml` file. Copy it and paste it into the `<Install>/WebApps/custom/topo/config` directory.

6. Find the line that reads:

   `<name>com.aprisma.spectrum.app.util.render.ModelNameColumn</name>`

7. Change this line to read:

   `<name>Index</name>`

8. Save your changes to the `column-iftree-config.xml` file.
9. Open the <Install>/WebApps/custom/topo/config directory and check to see if the interfaces-table-config.xml file already exists there. If it does not, go to the <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/ directory and find the interfaces-table-config.xml file. Copy it and paste it into the <Install>/WebApps/custom/topo/config directory.

10. Find the line that reads:

   <name>com.aprisma.spectrum.app.util.render.ModelNameColumn</name>

11. Change this line to read:

   <name>Index</name>

12. Save your changes to the interfaces-table-config.xml file.

13. Close all of the text files and restart the OneClick console in order for the changes to take effect.

If you would like to retain the interface model name in the table, implement the following additional instructions:

1. Open the <Install>/WebApps/custom/topo/config/interfaces-table-config.xml file.

2. Find the following entry within the <column-list> element:

   <column idref="column-iftree-config">
   <default-width>150</default-width>
   </column>

3. Add the following three lines to the end of this entry.

   <column idref="column-ifmodelname-config">
   <default-width>150</default-width>
   </column>

4. Close and restart the OneClick console in order for the changes to take effect.
Chapter 4: Adding Support for Model Types

The following chapter describes how to add OneClick topology support for a new management module that you have created with SPECTRUM’s Generic SNMP Toolkit.

In this chapter

- “Creating a New Icon for a Model Type”
- “Registering a Model Icon” on page 36
- “Configuring the Icon Used for Each OneClick Theme” on page 37
- “Designing On and Off Page Reference Icons” on page 39
- “Defining Model Icon Tooltips” on page 57

Creating a New Icon for a Model Type

If you have created a new model type or model class using SPECTRUM’s Generic SNMP toolkit, you can use the XML elements described in this chapter to represent this new model type in the OneClick interface.

You define new icons in the `custom-app-config.xml` file. Table 4-1 on page 36 shows all of the parent elements you can use in this file to create customized icons. The following procedure outlines how to use each of these elements to define a model’s appearance in the OneClick user interface. You can find a detailed description of each of these steps in rest of the sections of this chapter.

Procedure

1. If it does not already exist there, copy the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/custom-app-config.xml` file to the `<Install>/WebApps/custom/console/config` directory.
2. Open the file with a text editor.
3. Use the instructions found in “Registering a Model Icon” on page 36 to register the model icon with the OneClick interface.
4. Use the instructions found in “Configuring the Icon Used for Each OneClick Theme” on page 37 to define the icon’s appearance for each OneClick theme.
5. Use the instructions found in “Designing On and Off Page Reference Icons” on page 39 to design the appearance of the icon.

6. Use the instructions found in “Defining Model Icon Tooltips” on page 57 to define the icon’s tooltips.

7. Once you have created support for a model’s appearance, you may want to create an Information view for your new model type or model class. Instructions for doing this can be found in the “Customizing a Model’s Information View” on page 61 section.

8. Save and close the \texttt{<Install>/WebApps/custom/console/config/custom-app-config.xml} file.

9. Restart the OneClick client for your changes to take effect.

### Table 4-1: Parent Elements Used to Create an Icon in the custom-app-config.xml File

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{&lt;app-config&gt;}</td>
<td>NA</td>
<td>The root element for the custom-app-config.xml file.</td>
</tr>
<tr>
<td>\texttt{&lt;contents-registry&gt;}</td>
<td>\texttt{&lt;app-config&gt;}</td>
<td>Registers a custom icon for use in OneClick. See &quot;Registering a Model Icon&quot; on page 36 for a complete description.</td>
</tr>
<tr>
<td>\texttt{&lt;theme-config&gt;}</td>
<td>\texttt{&lt;app-config&gt;}</td>
<td>Defines the icon used for each OneClick theme.</td>
</tr>
<tr>
<td>\texttt{&lt;tooltip-config&gt;}</td>
<td>\texttt{&lt;app-config&gt;}</td>
<td>Creates the text that will be used for the icon’s tooltip. See &quot;Defining Model Icon Tooltips&quot; on page 57 for a complete description.</td>
</tr>
<tr>
<td>\texttt{&lt;view&gt;}</td>
<td>\texttt{&lt;app-config&gt;}</td>
<td>Creates subviews for the icon. See &quot;Customizing a Model’s Information View” on page 61 for a complete description.</td>
</tr>
</tbody>
</table>

### Registering a Model Icon

In order to link a model type or model class with the OneClick interface, you must add a registry entry to the custom-app-config.xml file.

The registry entry defines the icon registration ID, which is used to identify the section of XML code used for icon configuration for the model type or model class. The registry entry also defines the section of code used to define a model’s tooltip and the file used to create the model’s information view.

The following example shows a registry entry for a model class and model type.

```xml
<contents-registry>
  <icon-reg-id>mydevice-icon-config</icon-reg-id>
  <model-class>7</model-class>
  <model-type>0xffff0000</model-type>
  <tooltip-config>mydevice-tooltip-config</tooltip-config>
  <information-config>view-mydevicedetails-config</information-config>
</contents-registry>
```

This example assigns the registration ID \texttt{mydevice-icon-config} for all models with model class 7 and all models of type 0xffff0000. Note that you can specify multiple model classes and model types per icon registration.
Configuring the Icon Used for Each OneClick Theme

The model’s icon appearance will be defined by the icon configuration affiliated with the designated icon registration ID. Syntax for this portion of the custom-app-config.xml is described in the “Designing On and Off Page Reference Icons” on page 39 section.

The model’s icon tooltips will be defined by the Tooltip Configuration file specified by the <tooltip-config> element. Syntax for this portion of the custom-app-config.xml is described in the “Defining Model Icon Tooltips” on page 57 section.

The model’s Information view will be defined in the Information Configuration file specified by the <information-config> element (see “Customizing a Model’s Information View” on page 61 for more details).

### Table 4-2: Registering an Icon

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;app-config&gt;</td>
<td>NA</td>
<td>The root element for the custom-app-config.xml file.</td>
</tr>
<tr>
<td>&lt;contents-registry&gt;</td>
<td>&lt;app-config&gt;</td>
<td>Registers a custom icon for use in OneClick.</td>
</tr>
<tr>
<td>&lt;icon-reg-id&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>The icon registration ID, which registers the icon with OneClick and identifies the icon configuration to use. The icon configuration defines the appearance of the icon, see “Designing On and Off Page Reference Icons” on page 39 for more information.</td>
</tr>
<tr>
<td>&lt;model-class&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>Registers a model class for the ID &lt;icon-reg-id&gt;.</td>
</tr>
<tr>
<td>&lt;model-type&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>Registers a model-type for the ID &lt;icon-reg-id&gt;. Fill in the model type handle for your model type.</td>
</tr>
<tr>
<td>&lt;is-derived-from&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>Registers a parent model and all of its derived models. All of these models will have the appearance defined by the icon configuration.</td>
</tr>
<tr>
<td>&lt;tooltip-config&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>The XML that will define the topology icons tooltip content. See ”Defining Model Icon Tooltips” on page 57 for more information.</td>
</tr>
<tr>
<td>&lt;information-config&gt;</td>
<td>&lt;contents-registry&gt;</td>
<td>The XML that will define the content of the model’s Information tab. See “Customizing a Model’s Information View” on page 61 for more information.</td>
</tr>
</tbody>
</table>

As of SPECTRUM version 7.1 SP2, the <contents-registry> element replaces the <icon-registry> element and the <information-config> element replaces the <details-config> element. These new elements should be used going forward, however SPECTRUM OneClick will maintain backwards compatibility with the older <icon-registry> and <details-config> elements.

Configuring the Icon Used for Each OneClick Theme

A theme configuration defines the model’s icon appearance in each of the different OneClick themes. You create three theme configurations for each icon in the custom-app-config.xml file. In each theme configuration, you define the supporting XML to create the model’s standard and off-page reference icon. If you use different supporting XML files to define the icons for each theme, your model icon to have a different appearance for each OneClick theme. The elements that you use to create a theme configuration are shown in Table 4-3 on page 39.

The <theme-config> element’s id attribute links the theme configuration to a specific icon registration ID. Since model types and model classes are associated with a particular icon
registration ID as shown in “Registering a Model Icon” on page 36, the <theme-config> element’s id links specific model types and model classes to a defined icon appearance.

You must define a separate theme configuration for each of the desired OneClick console themes, “Classic”, “OneClick”, and “Utility”. The <theme> element specifies which theme the icon configuration is defined for. If you do not define an theme configuration for a specific theme, the icon will not be able to be represented in that theme.

The <on-page> and <off-page> elements name the icon configuration file that is used to define how the standard and off-page reference icon for the model appears.

There are three possible types of <*-cellicon> elements: <enumerate-cellicon>, <dynamic-cellicon>, and <static-cellicon>. The idref attribute of the <*-cellicon> element defines the image to be used in the model’s hierarchy.

The following sample XML shows the theme configuration for the icon registration ID mydevice-icon-config. Notice that each of the icon’s appearance is defined for each of the OneClick themes: Utility, Classic, and OneClick.

```xml
<theme-config>
  <icon-reg-id>mydevice-icon-config</icon-reg-id>
  <theme>Utility</theme>
  <on-page idref="mydeviceutil-iconbase-config"/>
  <off-page idref="mydeviceutil-oprbase-config"/>
  <enumerated-cellicon idref="component-cellicon-config"/>
</theme-config>

<theme-config>
  <icon-reg-id>mydevice-icon-config</icon-reg-id>
  <theme>Classic</theme>
  <on-page idref="mydeviceclassic-iconbase-config"/>
  <off-page idref="mydeviceclassic-oprbase-config"/>
  <enumerated-cellicon idref="component-cellicon-config"/>
</theme-config>

<theme-config>
  <icon-reg-id>mydevice-icon-config</icon-reg-id>
  <theme>OneClick</theme>
  <on-page idref="mydeviceoc-iconbase-config"/>
  <off-page idref="mydeviceoc-oprbase-config"/>
  <enumerated-cellicon idref="component-cellicon-config"/>
</theme-config>
```

**Note:** As of SPECTRUM version 7.1 SP2, the <theme-config> element has replaced the <icon-config> element in the custom-app-config.xml file. The <icon-config> element is now only used as the root element of the icon configuration file. See “Designing On and Off Page Reference Icons” on page 39 for information on icon configuration files.
Designing On and Off Page Reference Icons

The on-page and off-page references define how the standard and referenced icons will look in the OneClick interface. You must construct a new XML file, called an icon configuration file, to define each of these icons. Place the icon configuration files in the `<Install>/WebApps/custom/topo/config` directory. You must use the names of these files as the values for the idref attributes of the `<on-page>` and `<off-page>` elements (see “Configuring the Icon Used for Each OneClick Theme” on page 37).

For example, if you create an icon configuration file called, `mydevice-utility-iconbase-config.xml`, which defines the on-page reference for your icon in the Utility them, you must define the `<on-page>` reference element in the appropriate theme configuration as follows:

```xml
<theme-config>
  <icon-reg-id>mydevice-icon-config</icon-reg-id>
  <theme>Utility</theme>
  <on-page idref="mydevice-utility-iconbase-config"/>
  <off-page idref="mydevice-utility-oprbase-config"/>
</theme-config>
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;theme-config&gt;</code></td>
<td><code>&lt;content-registry&gt;</code></td>
<td>Defines the theme configuration for a specific model or model class.</td>
</tr>
<tr>
<td><code>&lt;icon-reg-id&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>The icon registration ID.</td>
</tr>
<tr>
<td><code>&lt;enumerated-cellicon&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>Specifies that an enumerated cell icon image should be used for the model hierarchy tree. The idref attribute of this element defines the id of the chosen cellicon. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;dynamic-cellicon&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>Specifies that a dynamic cell icon image should be used for the model hierarchy tree. The idref attribute of this element defines the id of the chosen cellicon. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;static-cellicon&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>Specifies that a static cell icon image should be used for the model hierarchy tree. The idref attribute of this element defines the id of the chosen cellicon. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;theme&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>The theme this registration is for. Possible values are OneClick, Classic, or Utility.</td>
</tr>
<tr>
<td><code>&lt;on-page&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>The icon configuration for the standard icon for the designated theme. See “Designing On and Off Page Reference Icons” on page 39 for more information.</td>
</tr>
<tr>
<td><code>&lt;off-page&gt;</code></td>
<td><code>&lt;theme-config&gt;</code></td>
<td>The icon configuration for the referenced icon for the designated theme. See “Designing On and Off Page Reference Icons” on page 39 for more information.</td>
</tr>
</tbody>
</table>
Chapter 4: Adding Support for Model Types

<enumerated-cellicon idref="component-cellicon-config"/>
</theme-config>

**Table 4-4: Element Used to Create On-Page and Off-Page Reference Icons**

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;icon-config&gt;</code></td>
<td>NA</td>
<td>This is the root element for the icon configuration file.</td>
</tr>
<tr>
<td><code>&lt;static-cellicon&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>A single color foreground/background or image. You can use the idref attribute to refer to another file that defines the image or color. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;dynamic-cellicon&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>More than one color or image. You can use the idref attribute to refer to another file that defines the image or color. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;enumerated-cellicon&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>Displays an image or color based on the value of an attribute. You can use the idref attribute to refer to another file that defines the image or color. Predefined icons exist in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/WEB-INF/console/topo/config</code> directory and follow the naming pattern <code>*-color-config.xml</code>.</td>
</tr>
<tr>
<td><code>&lt;shape&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>See <a href="#"><em>Defining the Icon Shape</em> on page 43</a>.</td>
</tr>
</tbody>
</table>
### Designing On and Off Page Reference Icons

#### <stroke> <icon-config>

The type of line used to create the shape. The following values can be used:
- **Bump1**: Bump stroke of width 1.
- **Bump2**: Bump stroke of width 2.
- **Dash1x1x1**: Dashed Stroke of width of 1, and repeats dash of length 1 followed by a blank of length 1.
- **Dash1x1x3**: Dashed Stroke of width of 1, and repeats dash of length 1 followed by a blank of length 3.
- **Dash1x2x2**: Dashed Stroke of width of 1, and repeats dash of length 2 followed by a blank of length 2.
- **Dash1x3x1**: Dashed Stroke of width of 1, and repeats dash of length 3 followed by a blank of length 1.
- **Dash2x1x1**: Dashed Stroke of width of 2, and repeats dash of length 1 followed by a blank of length 1.
- **Dash2x1x3**: Dashed Stroke of width of 2, and repeats dash of length 1 followed by a blank of length 3.
- **Dash2x2x2**: Dashed Stroke of width of 2, and repeats dash of length 2 followed by a blank of length 2.
- **Dash2x3x1**: Dashed Stroke of width of 2, and repeats dash of length 3 followed by a blank of length 1.
- **DashDot**: Dashed Stroke of width of 1, and repeats dash of length 3 followed by a blank of length 1 followed by a dot of length 1 followed by a blank of length 1.
- **DashDotDot**: Dashed Stroke of width of 1, and repeats dash of length 3 followed by a blank of length 1 followed by a dot of length 1 followed by a blank of length 1 followed by a dot of length 1 followed by a blank of length 1.
- **DashedSeparator**: Dashed Stroke used for display a separator lines for the SPECTRUM look and feel.
- **PenStroke**: Ditch stroke with a width of 1.
- **Ditch2**: Ditch stroke with a width of 2.
- **Ditch4**: Ditch stroke with a width of 4.
- **Hole1**: Hole stroke of width 1.
- **Hole2**: Hole stroke of width 2.
- **Invisible**: Invisible Pen Stroke.
- **Ridge2**: Ridge stroke with a width of 2.
- **Solid1**: Solid Pen Stroke with a width of 1.
- **Solid2**: Solid Pen Stroke with a width of 2.
- **Solid3**: Solid Pen Stroke with a width of 3.
- **StepBump1**: Step Bump with a width of 1.
- **StepBump2**: Step Bump with a width of 2.
- **StepHole2**: Step Hole with a width of 2.

#### <pipe-connection> <icon-config>

See "Defining Pipe Connection Location" on page 46.

#### <components> <icon-config>

This element encloses the image, text, and selection components that define the icon. The image, text and selections components are layered on top of each other and the index value for each of these components determines the drawing order. The lowest number (1) will be drawn first. See "Defining Image Components" on page 47, "Defining Text Components" on page 51 and "Defining Selection Components" on page 55.
Chapter 4: Adding Support for Model Types

The "Sample Icon Configuration File #1" on page 42 shows a sample icon configuration file using the XML elements shown in Table 4-4 on page 40. These elements are broken down and explained in the following sections:

- "Defining the Icon Shape" on page 43
- "Defining Pipe Connection Location" on page 46
- "Defining Image Components" on page 47
- "Defining Text Components" on page 51
- "Defining Selection Components" on page 55

Sample Icon Configuration File #1

```xml
<?xml version="1.0" encoding="UTF-8"?>
<icon-config id="mydevice-utility-iconbase-config">
  <static-color idref="default-iconbase-color-config"/>
  <shape-rectangle>
    <x>0</x>
    <y>0</y>
    <width>139</width>
    <height>84</height>
  </shape-rectangle>
  <stroke>Invisible</stroke>
  <!-- Specify the location of where pipes will connect to the icon. -->
  <pipe-connection>
    <x>73</x>
    <y>36</y>
  </pipe-connection>
  <components>
    <!-- Definition of the model text background. -->
    <image-component index="1">
      <x>0</x>
      <y>75</y>
      <width>94</width>
      <height>40</height>
      <image>
        <select>
          <default>
            images/icon_text_background.png
          </default>
        </select>
      </image>
    </image-component>
    <!-- Definition of the model's base image. The color of this image is determined by the condition of the model. -->
    <image-component index="2">
      <x>0</x>
      <y>0</y>
      <width>139</width>
    </image-component>
  </components>
</icon-config>
```

OneClick Customization Guide
Defining the Icon Shape

The base shape of the icon represents the active (click) area for the icon. This is the area in which the user will have to click the mouse in order to activate the model. The base shape can be a rectangle, rounded rectangle, ellipse, polygon, or line. You define each of these shapes within the `<icon-config>` element as shown in the following sections:

**Rectangle**

```xml
<shape-rectangle>
  <x>26</x>
  <y>24</y>
  <width>45</width>
  <height>14</height>
</shape-rectangle>
```

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;shape-rectangle&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>Defines a rectangle.</td>
</tr>
<tr>
<td><code>&lt;x&gt;</code></td>
<td><code>&lt;shape-rectangle&gt;</code></td>
<td>The upper left X coordinate of the rectangle. See &quot;X and Y Coordinates&quot; on page 46.</td>
</tr>
<tr>
<td><code>&lt;y&gt;</code></td>
<td><code>&lt;shape-rectangle&gt;</code></td>
<td>The upper left Y coordinate of the rectangle. See &quot;X and Y Coordinates&quot; on page 46.</td>
</tr>
<tr>
<td><code>&lt;width&gt;</code></td>
<td><code>&lt;shape-rectangle&gt;</code></td>
<td>The width of the rectangle.</td>
</tr>
<tr>
<td><code>&lt;height&gt;</code></td>
<td><code>&lt;shape-rectangle&gt;</code></td>
<td>The height of the rectangle.</td>
</tr>
</tbody>
</table>

**Rounded Rectangle**

```xml
<shape-roundrectangle>
  <x>0</x>
  <y>0</y>
  <width>89</width>
  <height>68</height>
  <arcwidth>12</arcwidth>
  <archeight>12</archeight>
</shape-roundrectangle>
```

Table 4-5: Elements Used to Define a Rectangle
Table 4-6: Elements Used to Define a Rounded Rectangle

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;shape-roundrectangle&gt;</td>
<td>&lt;icon-config&gt;</td>
<td>Defines a rounded rectangle.</td>
</tr>
<tr>
<td>&lt;x&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The upper left X coordinate. See “X and Y Coordinates” on page 46.</td>
</tr>
<tr>
<td>&lt;y&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The upper left Y coordinate. See “X and Y Coordinates” on page 46.</td>
</tr>
<tr>
<td>&lt;width&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The width.</td>
</tr>
<tr>
<td>&lt;height&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The height.</td>
</tr>
<tr>
<td>&lt;arcwidth&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The width of the arc that defines the corners.</td>
</tr>
<tr>
<td>&lt;arceheight&gt;</td>
<td>&lt;shape-roundrectangle&gt;</td>
<td>The height of the arc that defines the corners.</td>
</tr>
</tbody>
</table>

**Ellipse**

<shape-ellipse>
  <x>26</x>
  <y>24</y>
  <width>45</width>
  <height>14</height>
</shape-ellipse>

Table 4-7: Elements Used to Define an Ellipse

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;shape-ellipse&gt;</td>
<td>&lt;icon-config&gt;</td>
<td>Creates an ellipse.</td>
</tr>
<tr>
<td>&lt;x&gt;</td>
<td>&lt;shape-ellipse&gt;</td>
<td>The upper left X coordinate of the ellipse. See “X and Y Coordinates” on page 46.</td>
</tr>
<tr>
<td>&lt;y&gt;</td>
<td>&lt;shape-ellipse&gt;</td>
<td>The upper left Y coordinate of the ellipse. See “X and Y Coordinates” on page 46.</td>
</tr>
<tr>
<td>&lt;width&gt;</td>
<td>&lt;shape-ellipse&gt;</td>
<td>The width of the ellipse.</td>
</tr>
<tr>
<td>&lt;height&gt;</td>
<td>&lt;shape-ellipse&gt;</td>
<td>The height of the ellipse.</td>
</tr>
</tbody>
</table>

**Polygon**

<shape-polygon>
  <point x="0" y="28" />
  <point x="10" y="10" />
  <point x="28" y="0" />
  <point x="116" y="0" />
  <point x="134" y="10" />
  <point x="144" y="28" />
  <point x="144" y="65" />
  <point x="134" y="80" />
  <point x="116" y="92" />
  <point x="28" y="92" />
  <point x="10" y="80" />
</shape-polygon>
Designing On and Off Page Reference Icons

Table 4-8: Elements Used to Define a Polygon

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;shape-polygon&gt;</td>
<td>&lt;icon-config&gt;</td>
<td>Defines a polygon</td>
</tr>
<tr>
<td>&lt;point&gt;</td>
<td>&lt;shape-polygon&gt;</td>
<td>A point defining an edge of the polygon. The x attribute defines the x coordinate position of the point. The y attribute of this element defines the y coordinate position of the point.</td>
</tr>
</tbody>
</table>

Line

<shape-line>
  <x1>5</x1>
  <y1>5</y1>
  <x2>40</x2>
  <y2>40</y2>
</shape-line>

Table 4-9: Elements Used to Define a Line

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;shape-line&gt;</td>
<td>&lt;icon-config&gt;</td>
<td>Defines a line</td>
</tr>
<tr>
<td>&lt;x1&gt;</td>
<td>&lt;shape-line&gt;</td>
<td>X coordinate for the start of the line.</td>
</tr>
<tr>
<td>&lt;y1&gt;</td>
<td>&lt;shape-line&gt;</td>
<td>Y coordinate for the start of the line.</td>
</tr>
<tr>
<td>&lt;x2&gt;</td>
<td>&lt;shape-line&gt;</td>
<td>X coordinate for the end of the line.</td>
</tr>
<tr>
<td>&lt;y2&gt;</td>
<td>&lt;shape-line&gt;</td>
<td>Y coordinate for the end of the line.</td>
</tr>
</tbody>
</table>

Example of Creating an Icon Shape

The following example shows a portion of the XML from an icon configuration file. This XML defines the rounded-rectangle shown in Figure 4-1 on page 46.

<icon-config id="device-iconbase-config">
  <static-color idref="device-iconbase-color-config"/>
  <shape-roundrectangle>
    <x>0</x>
    <y>0</y>
    <width>89</width>
    <height>68</height>
    <arcwidth>12</arcwidth>
    <archeight>12</archeight>
  </shape-roundrectangle>
</icon-config>
Chapter 4: Adding Support for Model Types

Figure 4-1: Defining Click Zones

X and Y Coordinates

The x and y coordinates define where the upper left hand corner of the image begins. When the value of x is increased, the upper left hand corner of the image is moved from the left to the right. When the value of y is increased, the upper left hand corner of the image is moved from the top to the bottom.

Figure 4-2: x and y Coordinates

Defining Pipe Connection Location

The pipe location indicates the location on the icon where OneClick connects the pipes to the icon.

```xml
<pipe-connection>
  <x>73</x>
  <y>36</y>
</pipe-connection>
```
Defining Image Components

Image components enable you to add static or dynamic images to a model based on the model attributes.

The following example shows you how to define an image component within your icon configuration file:

```xml
<image-component index="1">
  <x>0</x>
  <y>0</y>
  <width>139</width>
  <height>84</height>
  <image idref="oneclick-orgservice-iconbase-image-config"/>
</image-component>
```

The `<image>` element shown above references the file defined in the "Sample Image Definition File" on page 49 section.

This sample XML generates the image component shown in Figure 4-3 on page 47 assuming that the condition of the model is CRITICAL (3).

![Figure 4-3: Image Component](image.png)
### Table 4-11: Elements Used for Defining Image Components

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;components&gt;</code></td>
<td><code>&lt;icon-config&gt;</code></td>
<td>Defines all components for the icon.</td>
</tr>
<tr>
<td><code>&lt;image-component&gt;</code></td>
<td><code>&lt;components&gt;</code></td>
<td>Defines the image component. The index attribute of this element defines the order in which the defined image is drawn relative to the other image, text, and selection components defined for this icon. Each index value must be unique, for example you cannot have two <code>&lt;*-component&gt;</code> elements with an index value of 1 in the same file. If you do, only the first <code>&lt;*-component&gt;</code> element will be used. Index values must begin at 1.</td>
</tr>
<tr>
<td><code>&lt;x&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>The x coordinate of the upper left corner of image.</td>
</tr>
<tr>
<td><code>&lt;y&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>The y coordinate of the upper left corner of image.</td>
</tr>
<tr>
<td><code>&lt;width&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>The width of the image. There are no size restrictions on the image, however, it is recommended that you use a size relative to the other images used in OneClick.</td>
</tr>
<tr>
<td><code>&lt;height&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>The height of the image. There are no size restrictions on the image, however, it is recommended that you use a size relative to the other images used in OneClick.</td>
</tr>
<tr>
<td><code>&lt;image&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>The image to be rendered. The idref attribute references the XML file that builds the image. It is put in a separate file for organizational purposes only. Images should be in png file format and should be placed in the <code>&lt;Install&gt;/WebApps/tomcat/webapps/spectrum/images</code> directory.</td>
</tr>
<tr>
<td><code>&lt;shape-*&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>Shape component to be drawn with image.</td>
</tr>
<tr>
<td><code>&lt;enumerated-color&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>Colors used for the shape.</td>
</tr>
<tr>
<td><code>&lt;static-color&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>Color used for the shape.</td>
</tr>
<tr>
<td><code>&lt;selection-component&gt;</code></td>
<td><code>&lt;image-component&gt;</code></td>
<td>Indicates whether this is to be shown only when the user selects the image component.</td>
</tr>
</tbody>
</table>
Sample Image Definition File

The following XML code defines the image used for `<image-component index="2">` in the "Sample Icon Configuration File #1" on page 42. The example uses the `<select>`, `<case>`, and `<expression>` elements to conditionally select an image based on the value of the condition attribute. The example uses the `<yield>` to define which image should be used when a condition is met. Note that the image is in PNG file format. Images used for customization purposes should be stored in the `<Install>/WebApps/tomcat/webapps/spectrum/images` directory. When you reference an image placed in this directory you will need to express the path from the images directory, i.e. `images/myimage.png`.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<image id="oneclick-orgservice-iconbase-image-config">
  <select>
    <case>
      <expression>
        attrInt(AttributeID.CONDITION) == 0
      </expression>
      <yield>
        images/oneclick_org_service_green.png
      </yield>
    </case>
    <case>
      <expression>
        attrInt(AttributeID.CONDITION) == 1
      </expression>
      <yield>
        images/oneclick_org_service_yellow.png
      </yield>
    </case>
    <case>
      <expression>
        attrInt(AttributeID.CONDITION) == 2
      </expression>
      <yield>
        images/oneclick_org_service_orange.png
      </yield>
    </case>
    <case>
      <expression>
        attrInt(AttributeID.CONDITION) == 3
      </expression>
      <yield>
        images/oneclick_org_service_red.png
      </yield>
    </case>
    <case>
      <expression>
        attrInt(AttributeID.CONDITION) == 4
      </expression>
      <yield>
        images/oneclick_org_service_brown.png
      </yield>
    </case>
  </select>
</image>
```
<expression>
    attrInt(AttributeID.CONDITION) == 5
</expression>
<yield>
    images/oneclick_org_service_grey.png
</yield>
</case>
<case>
    <expression>
        attrInt(AttributeID.CONDITION) == 6
    </expression>
    <yield>
        images/oneclick_org_service_blue.png
    </yield>
</case>
<default>
    images/oneclick_org_service_blue.png
</default>
</select>
</image>

**Icon Configuration Example #2**

This icon configuration example generates the image shown in Figure 4-4 on page 51, assuming that the condition of the model is CRITICAL (3).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<icon-config id="oneclick-orgservice-iconbase-config">
    <static-color idref="oneclick-default-iconbase-color-config"/>
    <shape-rectangle>
        <x>0</x>
        <y>0</y>
        <width>139</width>
        <height>84</height>
    </shape-rectangle>
    <stroke>Invisible</stroke>
    <!-- --------------------------------------------------------
    - Specify the location of where pipes will connect to the icon.
    ---------------------------------------------------------->
    <pipe-connection>
        <x>73</x>
        <y>36</y>
    </pipe-connection>
    <components>
        <!-- --------------------------------------------------------
        - Definition of the model text background.
        ---------------------------------------------------------->
        <image-component index="1">
            <x>0</x>
            <y>75</y>
            <width>94</width>
            <height>40</height>
            <image>
                <select>
                    <default>
                        images/icon_text_background.png
                    </default>
                </select>
            </image>
        </image-component>
    </components>
</icon-config>
```
Defining Text Components

Text components enables you to display static text, or dynamic text based on attributes on the model icon.

The following example shows you how to define a text component within your icon configuration file:

```xml
<text-component idref="oneclick-default-textfield-config" index="5">
  <x>5</x>
  <y>97</y>
  <width>85</width>
  <height>13</height>
  <horizontal_alignment>left</horizontal_alignment>
  <text>
    <attribute>0x1006e</attribute>
  </text>
</text-component>
```
**Table 4-12: Elements Used to Define Text Components**

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;components&gt;</td>
<td>&lt;icon-config&gt;</td>
<td>Defines all components for the icon.</td>
</tr>
<tr>
<td>&lt;text-component&gt;</td>
<td>&lt;components&gt;</td>
<td>Defines the text component. The index attribute of this element defines the order in which the defined image is drawn relative to the other image, text, and selection components defined for this icon. Each index value must be unique, for example you cannot have two &lt;<em>-component&gt; elements with an index value of 1 in the same file. If you do, only the first &lt;</em>-component&gt; element will be used. Index values must begin at 1.</td>
</tr>
<tr>
<td>&lt;x&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The x coordinate of the lower left baseline of text</td>
</tr>
<tr>
<td>&lt;y&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The y coordinate of the lower left baseline of text</td>
</tr>
<tr>
<td>&lt;width&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The width of the text area</td>
</tr>
<tr>
<td>&lt;height&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The height of the text area.</td>
</tr>
<tr>
<td>&lt;text&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The text to be displayed within the text area.</td>
</tr>
<tr>
<td>&lt;horizontal_alignment&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The alignment of the text horizontally Possible values: left, center, right</td>
</tr>
<tr>
<td>&lt;vertical_alignment&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The alignment of the text vertically Possible values: top, center, bottom</td>
</tr>
<tr>
<td>&lt;font&gt;</td>
<td>&lt;text-component&gt;</td>
<td>The font of the displayed text.</td>
</tr>
<tr>
<td>&lt;enumerated-color&gt;</td>
<td>&lt;text-component&gt;</td>
<td>Enumerated colors used for the text.</td>
</tr>
<tr>
<td>&lt;static-color&gt;</td>
<td>&lt;text-component&gt;</td>
<td>Static colors used for the text.</td>
</tr>
<tr>
<td>&lt;selection-component&gt;</td>
<td>&lt;components&gt;</td>
<td>Indicates whether this is to be shown only during selection. See &quot;Defining Selection Components&quot; on page 55.</td>
</tr>
</tbody>
</table>

**X, Y, Height, and Width**

The x and y location of the text component is relative to the upper left hand corner of the model icon being developed. This coordinate marks the left baseline of the text. The height is measured upward from the y coordinate, and the width is measured to the right of the X component.

**Justification:**

The horizontal and vertical orientation of the text can be set via the horizontal_alignment and vertical_alignment elements. The possible values for the horizontal_alignment element are: left, right, and center. The possible values for the vertical_alignment element are top, center, and bottom. The following demonstrates the dimensions of a text area along with the justification options. The justification is displayed as Horizontal/Vertical:
**Sample Icon Configuration #3**

The following icon configuration sample shows the use of text components. The resulting icon looks like the one shown in Figure 4-6, assuming that the condition is CRITICAL, model name is Aprisma, and model type name is Service_Owns:

![Figure 4-6: Text Components](image)

```xml
<?xml version="1.0" encoding="UTF-8"?>
<icon-config id="oneclick-orgservice-iconbase-config">
  <static-color idref="oneclick-default-iconbase-color-config"/>
  <shape-rectangle >
    <x>0</x>
    <y>0</y>
    <width>139</width>
    <height>84</height>
  </shape-rectangle>
  <stroke>Invisible</stroke>
  <!-- Specify the location of where pipes will connect to the icon. -->
  <pipe-connection>
    <x>73</x>
    <y>36</y>
  </pipe-connection>
  <components>
    <!-- Definition of the model text background. -->
    <image-component index="1">
      <x>0</x>
      <y>75</y>
    </image-component>
  </components>
</icon-config>
```
<width>94</width>
<height>40</height>
<image>
  <select>
    <default>
      com/aprisma/spectrum/app/topo/images/icon_text_background.png
    </default>
  </select>
</image>
</image-component>

<!-------------------------------------------------------------------------------------------------------------------------------------
  - Definition of the model's base image. The color of this
  - image is determined by the condition of the model.
  - ------------------------------------------------------------------------------------------------------------------------------------- -->

<image-component index="2">
  <x>0</x>
  <y>0</y>
  <width>139</width>
  <height>84</height>
  <image idref="oneclick-orgservice-iconbase-image-config"/>
</image-component>

<!-------------------------------------------------------------------------------------------------------------------------------------
  - Definition of the model name text field.
  - ------------------------------------------------------------------------------------------------------------------------------------- -->

<text-component idref="oneclick-default-textfield-config" index="3">
  <x>5</x>
  <y>97</y>
  <width>85</width>
  <height>13</height>
  <horizontal_alignment>left</horizontal_alignment>
  <text>
    <attribute>0x1006e</attribute>
  </text>
</text-component>

<!-- -------------------------------------------------------------------------------------------------------------------------------------
  - Definition of the model type name text.
  - ------------------------------------------------------------------------------------------------------------------------------------- -->

<text-component idref="oneclick-default-textfield-config" index="4">
  <x>5</x>
  <y>107</y>
  <width>85</width>
  <height>12</height>
  <horizontal_alignment>left</horizontal_alignment>
  <text>
    <attribute>AttributeID.DEVICE_TYPE</attribute>
    <expression>
      ( value() == null || ((String)value()).length() == 0 ) ? attr(AttributeID.MTYPE_NAME ) : value()
    </expression>
  </text>
</text-component>

</components>
</icon-config>
Defining Selection Components

In order to make the icon standout when selected, you may want to specify images that will appear only during selection. You do this via the `<selection-component>` element. In the example below there are two components added that are defined as Selection Components (image component #2 and #4). If the user selected the component, these image components would become visible. Otherwise, they remain invisible. Figure 4-7 shows the two selection components:

![Selection Components](image.png)

<?xml version="1.0" encoding="UTF-8"?>
<icon-config id="oneclick-orgservice-iconbase-config">
  <static-color idref="oneclick-default-iconbase-color-config"/>
  <shape-rectangle >
    <x>0</x>
    <y>0</y>
    <width>139</width>
    <height>84</height>
  </shape-rectangle>
  <stroke>Invisible</stroke>
  <!-- =============================================================
  - Specify the location of where pipes will connect to the icon. -
  ============================================================== -->
  <pipe-connection>
    <x>73</x>
    <y>36</y>
  </pipe-connection>
  <components>
    <!-- =============================================================
    - Definition of the model text background. -
    ============================================================== -->
    <image-component index="1">
      <x>0</x>
      <y>75</y>
      <width>94</width>
      <height>40</height>
      <image>
        <select>
          <default>
            com/aprisma/spectrum/app/topo/images/icon_text_background.png
          </default>
        </select>
      </image>
    </image-component>
  </components>
</icon-config>
Chapter 4: Adding Support for Model Types

<!-- ============================================================ - Definition of the model's base image. The color of this image is determined by the condition of the model. -->

<image-component index="2">
  <x>0</x>
  <y>75</y>
  <width>94</width>
  <height>40</height>
  <select-component>true</select-component>
  <select>
    <default>
      com/aprisma/spectrum/app/topo/images/icon_selected_text_background.png
    </default>
  </select>
</image-component>

<!-- ============================================================== - Definition of the image to show when the model is selected. -->

<image-component index="3">
  <x>0</x>
  <y>0</y>
  <width>139</width>
  <height>84</height>
  <image idref="oneclick-orgservice-iconbase-image-config"/>
</image-component>

<!-- ============================================================== - Definition of the model name text field. -->

<text-component idref="oneclick-default-textfield-config" index="5">
  <x>5</x>
  <y>97</y>
  <width>85</width>
  <height>13</height>
  <horizontal_alignment>left</horizontal_alignment>
</text-component>
Defining Model Icon Tooltips

You can configure the content of a tooltip that will appear when hovering over the model icon. The file that defines the tooltip is designated by the `<tooltip-config>` element from icon-registry in the `custom-app-config.xml` file (see “Registering a Model Icon” on page 36). The example below shows that the tool-tip configuration for models of model class 2, 5, 11, and 12 will be registered to use the tooltip defined within the `mydevice-tooltip-config.xml` file.

```xml
<contents-registry>
  <reg-id>device-icon-config</reg-id>
  <tooltip-config>mydevice-tooltip-config</tooltip-config>
  <model-class>2</model-class>
  <model-class>5</model-class>
  <model-class>11</model-class>
  <model-class>12</model-class>
</contents-registry>
```

The following is an example shows the contents of a tooltip file. Each element is explained in Table 4-13 on page 59.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<tooltip-config id="mydevice-tooltip-config"
```
Chapter 4: Adding Support for Model Types

xmlns ="http://www.aprisma.com"
xmns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.aprisma.com../common/schema/column-config.xsd">
<format><![CDATA[
    <html><table>
        <tr>
            <td><b>{0}</b></td>
            <td>{1}</td>
        </tr>
        <tr>
            <td><b>{2}</b></td>
            <td>{3}</td>
        </tr>
        <tr>
            <td><b>{4}</b></td>
            <td>{5}</td>
        </tr>
    </table></html>
]]></format>
<param>
    <localize>com.aprisma.spectrum.app.util.render.ModelNameColumn</localize>
</param>
<param>
    <attribute>AttributeID.MODEL_NAME</attribute>
    <renderer>com.aprisma.spectrum.app.util.render.NullRenderer</renderer>
</param>
<param>
    <localize>com.aprisma.spectrum.app.util.render.NetworkAddressColumn</localize>
</param>
<param>
    <attribute>AttributeID.NETWORK_ADDRESS</attribute>
    <renderer>com.aprisma.spectrum.app.util.render.NullRenderer</renderer>
</param>
<param>
    <localize>com.aprisma.spectrum.app.util.render.MACAddressColumn</localize>
</param>
<param>
    <attribute>AttributeID.MAC_ADDRESS</attribute>
    <renderer>com.aprisma.spectrum.app.util.render.NullRenderer</renderer>
</param>
</device-tooltip-config>

The numbers used in the curly brackets reference the parameters defined by the <param> elements below. {0} references the first parameter defined, {1} references the second parameter defined, etc.
Table 4-13: Elements Used to Create Tooltips

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tooltip-config&gt;</td>
<td>NA</td>
<td>The root element for the file that defines the tooltip. The id attribute for this element must be set equal to the value used for the &lt;tooltip-config&gt; element in the &lt;content-registry&gt; found in the custom-app-config.xml file.</td>
</tr>
<tr>
<td>&lt;format&gt;</td>
<td>&lt;tooltip-config&gt;</td>
<td>Use this to define how the data will be displayed in the tooltip. In the above example, an HTML table is used. The number in the curly brackets, e.g. {3}, references the corresponding parameter, e.g. the third parameter defined in the file.</td>
</tr>
<tr>
<td>&lt;param&gt;</td>
<td>&lt;tooltip-config&gt;</td>
<td>Use this to define the value to be displayed.</td>
</tr>
<tr>
<td>&lt;localize&gt;</td>
<td>&lt;param&gt;</td>
<td>Converts the string specified in the parameter to a localized value. Use this if you are using a parameter value obtained from a OneClick XML file that shipped with SPECTRUM and begins with &quot;com.aprisma.spectrum&quot;.</td>
</tr>
<tr>
<td>&lt;renderer&gt;</td>
<td>&lt;param&gt;</td>
<td>See &quot;Customizing the Port Name Column of the Interface Table&quot; on page 32.</td>
</tr>
<tr>
<td>&lt;attribute&gt;</td>
<td>&lt;param&gt;</td>
<td>Use this to identify the attribute you want to be displayed.</td>
</tr>
<tr>
<td>&lt;message&gt;</td>
<td>&lt;param&gt;</td>
<td>Use this for specifying a plain text value for a parameter.</td>
</tr>
</tbody>
</table>
Chapter 5: Customizing a Model’s Information View

This chapter describes how to add and edit information displayed in the Information view for a particular model type or model class.

In this chapter

• “Overview”
• “Creating or Modifying an Information View” on page 63
• “Creating an Information Configuration File” on page 63
• “Associating the Information Configuration File with the Model Class or Model Type” on page 74

Overview

Each model displayed in OneClick has an Information view as shown in Figure 5-1 on page 62. You access this view via the Information tab in the Component Detail panel.
The Information views for existing model types are constructed in separate XML files called Information Configuration files. The `<information-config>` element in a model type’s `<contents-registry>` section of the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/topo-app-config.xml` file links the appropriate Information Configuration file to the appropriate model type.

For example, the following `<contents-registry>` can be found in the topo-app-config.xml file. It links the 0x2100c (Rtr_Cisco) model type with the view-devicedetails-config.xml file, which specifies the format for the Information view.

```
<contents-registry>
  <reg-id>router-icon-config</reg-id>
  <tooltip-config>device-tooltip-config</tooltip-config>
  <information-config>view-devicedetails-config</information-config>
  <performance-config>performance-data-rtrcisco-config</performance-config>
</contents-registry>
```

**Note:** Several model classes and model types may be specified within the contents or component details registries. Information views can be reused in one or more `<contents-registry>` elements.
Creating or Modifying an Information View

When you create or modify an Information view, you must create a new Information Configuration file and place it in the `<Install>/WebApps/custom/console/config/` directory. You must then associate the Information Configuration file with the appropriate model type using the `<Install>/WebApps/custom/console/config/custom-app.config.xml` file.

**Note:** You must use the `custom-app-config.xml` that is installed in `<Install/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/` directory. As with the other custom files, it should be first copied to the custom directory. See "Using the Custom Directory to Preserve Customizations" on page 10

**Procedure**

1. If you are modifying an Information view for an existing model type or model class, you must identify the current Information Configuration file used to create the Information view.
   a. Open the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config/topo-app-config.xml` file and find the `<contents-registry>` element for the appropriate model type or model class.
   b. Find the `<information-config>` element within the `<contents-registry>` element, and note the name of the Information Configuration file described by this element.
   c. All of the existing Information Configuration files are located in the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/topo/config` directory. Find the appropriate Information Configuration file in this directory and copy it to the `<Install>/WebApps/custom/topo/config` directory.
   d. Rename the file and note this new name.

2. If you are creating an entirely new Information Configuration file. Create a new text file and name it appropriately. Note this new name.

3. Open the file using a text editor and use the XML syntax outlined in "Customizing a Model’s Information View” on page 61 to build the file.

4. Save and close the file

5. Associate the Information Configuration file with the appropriate model types or model classes using the instructions found in "Associating the Information Configuration File with the Model Class or Model Type” on page 74.

**Creating an Information Configuration File**

The XML that defines the Information Configuration is split up into two major sections: the header definition and the subview definition. Each define a portion of the model’s information tab.
Figure 5-2: The Information Tab

![Diagram of the Information Tab]

Table 5-1: Elements that Define an Information View

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;view&gt;</td>
<td>NA</td>
<td>This is the root element for the Information Configuration file. The ID attribute for this element defines the value that should be used for the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;information-config&gt; element in the custom-app-config.xml file. See &quot;Associating the Information Configuration File with the Model Class or Model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Type&quot; on page 74.</td>
</tr>
<tr>
<td>&lt;view-header&gt;</td>
<td>&lt;view&gt;</td>
<td>Defines the header portion of the view. See &quot;Defining the Header&quot; on page 64 for more information.</td>
</tr>
<tr>
<td>&lt;subviews&gt;</td>
<td>&lt;view&gt;</td>
<td>Defines the available subviews. See &quot;Defining the Subview&quot; on page 65 for more information.</td>
</tr>
</tbody>
</table>

Defining the Header

The Information tab header is identified by the <view-header> element. This header provides the information for the model’s graphical depiction and textual information as shown in Figure 5-2 on page 64.

```xml
<view-header>
  <show-icon>true</show-icon>
  <show-labels>false</show-labels>
  <field-column>
    <column idref="column-modelname-config"/>
    <column idref="column-modeltype-config"/>
  </field-column>
</view-header>
```
Defining the Subview

The subview section defines one or more subviews that will be available from the Information tab as shown in Figure 5-2 on page 64. You can define one or more subviews using the <subviews> element and the child elements shown in Table 5-3. As shown in the “Example Subview Definition” on page 65, all subview definitions are enclosed within one <subviews> element.

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;subviews&gt;</td>
<td>&lt;view&gt;</td>
<td>Encloses all of the elements which define each type of subview.</td>
</tr>
<tr>
<td>&lt;field-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines a field subview. See “Adding a Field Subview” on page 66.</td>
</tr>
<tr>
<td>&lt;table-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines a table subview. See “Adding a Table Subview” on page 68.</td>
</tr>
<tr>
<td>&lt;application-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines an application subview. See “Adding an Application Subview” on page 69.</td>
</tr>
<tr>
<td>&lt;related-model-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines a related model subview. See “Adding a Related Model Subview” on page 70.</td>
</tr>
<tr>
<td>&lt;related-model-table-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines a related model table subview. See “Adding a Related Models Table Subview” on page 71.</td>
</tr>
<tr>
<td>&lt;subview-group&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Groups subviews together under one subview. See “Defining a Subview Group” on page 73.</td>
</tr>
</tbody>
</table>

Example Subview Definition

<subviews>
Adding a Field Subview

Field subviews are used to display a list of non-list attributes available on the selected device model. The following is an example of XML syntax used to create a field subview.

```xml
<field-subview>
  <title>General Information</title>
  <privilege>
    <name>GeneralInfo</name>
  </privilege>
  <field-column>
    <column idref="column-condition-config"/>
    <column idref="column-contactstatus-config"/>
    <column idref="column-networkaddress-config"/>
    <column idref="column-ismanaged-config">
      <editable/>
    </column>
    <column idref="column-securitystring-config">
      <editable verifier="com.aprisma.spectrum.app.swing.widget.SecStringInputVerifier"/>
    </column>
  </field-column>
  <field-column>
    <column idref="column-modelcreationdate-config"/>
    <column idref="column-modeltypename-config"/>
    <column idref="column-modelclass-config"/>
    <column idref="column-lastsuccessfulpoll-config"/>
    <column idref="column-landscape-config"/>
    <column idref="column-modelnotes-config">
      <editable/>
    </column>
  </field-column>
</field-subview>
```

This would generate a subview similar to the one shown in Figure 5-3 on page 67.
You can use the elements shown in Table 5-4 on page 67 to create a field subview.

**Table 5-4: Elements Used to Create a Field Subview**

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;field-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Defines a field subview. If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>The title of the subview. In our example above, the title is “General Information”.</td>
</tr>
<tr>
<td>&lt;display-if&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>Enables the author to specify whether the subview should be displayed via an expression.</td>
</tr>
<tr>
<td>&lt;display-if-app-installed&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>Enables the author to specify that the defined view only be added if the specified application is installed.</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>Associates a privilege to the subview. If the user is not given this privilege, the subview will not be displayed for that user. See “Creating Custom Privileges” on page 77 for more information.</td>
</tr>
<tr>
<td>&lt;show-labels&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>Indicates whether or not to show field labels. Values: true or false</td>
</tr>
<tr>
<td>&lt;field-column&gt;</td>
<td>&lt;field-subview&gt;</td>
<td>Constructs a column of information.</td>
</tr>
<tr>
<td>&lt;column&gt;</td>
<td>&lt;field-column&gt;</td>
<td>Defines the data for the field column. The idref attribute enables you to associate another XML file, which will define the data for the column. The data for the column does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;editable&gt;</td>
<td>&lt;column&gt;</td>
<td>Specifies if the column is editable.</td>
</tr>
</tbody>
</table>
Adding a Table Subview

A table subview enables you to display a group of list attributes available from the selected model. These list attributes are displayed in table format. The following is an example of XML syntax used to create a table subview.

```
<table-subview>
  <title>Interface Configuration Table</title>
  <privilege>
    <name>InterfaceConfigurationTable</name>
  </privilege>
  <swing-header-row-template>
    <static-color idref="row-header-color-config"/>
  </swing-header-row-template>
  <swing-row-template>
  </swing-row-template>
  <column-list>
    <column>
      <name>Interface</name>
      <content>
        <attribute>0x100c4</attribute>
      </content>
      <default-width>30</default-width>
    </column>
    <column>
      <name>Type</name>
      <content>
        <attribute>0x100c6</attribute>
        <renderer>
          <param name="attrID">0x100c6</param>
          com.aprisma.spectrum.app.util.render.EnumeratedAttrRenderer
        </renderer>
      </content>
      <default-width>100</default-width>
    </column>
    <column>
      <name>IF Speed</name>
      <content>
        <attribute>0x100c8</attribute> <!-- IfSpeed -->
        <renderer>com.aprisma.spectrum.app.topo.client.
          interfaces.render.IfSpeedRenderer
        </renderer>
      </content>
      <default-width>60</default-width>
    </column>
    <column>
      <name>Physical Address</name>
      <content>
        <attribute>0x100c9</attribute>
      </content>
      <default-width>90</default-width>
    </column>
  </column-list>
</table-subview>
```
This generates a subview similar to the one shown in Figure 5-4.

**Figure 5-4: Table Subview Example**

You can use the elements shown in Table 5-5 to create a table subview.

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;table-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Adds a table subview. If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;table-subview&gt;</td>
<td>The title for the table.</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;table-subview&gt;</td>
<td>Associates a privilege to the subview. If the user is not given this privilege, the subview will not be displayed for that user. See &quot;Creating Custom Privileges&quot; on page 77 for more information.</td>
</tr>
</tbody>
</table>

All of the elements shown in Table 3-1: Elements Used for Modifying a Table page 27 can be used to create the table for the table subview.

### Adding an Application Subview

An application subview enables you to display attributes affiliated with an application model type related to the selected model type by a specified criteria. The example below uses SPECTRUM’s PossPrimApp (0x230000) relation.

```xml
<application-subview>
  <title>SNMP2 IP Routing Table</title>
  <model-type>0x230010</model-type>
  <subviews>
    <table-subview idref="table-ip2-ip-routingtable-config"/>
  </subviews>
</application-subview>
```

The attribute used within the `<model-type>` element defines the model type to which the subview pertains. In the example above, the value 0x230010 is used. This attribute value corresponds to the SNMP2-Agent Application model type. This means that this particular table definition applies...
only to the SNMP2_Agent application. If the current device does not implement this application, this table will not be visible.

Table 5-6: Elements Used for Creating an Application Subview

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;application-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>Creates and application subview. If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;application-subview&gt;</td>
<td>The title of the subview</td>
</tr>
<tr>
<td>&lt;model-type&gt;</td>
<td>&lt;application-subview&gt;</td>
<td>The model type to which the subviews will pertain.</td>
</tr>
<tr>
<td>&lt;subviews&gt;</td>
<td>&lt;application-subview&gt;</td>
<td>Enables you to add a prebuilt table-subview or field-subview to the detail components that reside within the application subview. Values: table-subview and/or field-subview</td>
</tr>
<tr>
<td>&lt;criteria&gt;</td>
<td>&lt;application-subview&gt;</td>
<td>The search criteria used to find the related models.</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;application-subview&gt;</td>
<td>Associates a privilege to the subview. If the user is not given this privilege, the subview will not be displayed for that user. See “Creating Custom Privileges” on page 77 for more information.</td>
</tr>
</tbody>
</table>

Adding a Related Model Subview

A related model subview enables the user to display the attributes of models related to the current, selected model via a search criteria, for example, models that are related by a specific association. The user can then display attributes of the found models in field or table format. The section “Adding a Related Models Table Subview” on page 71 shows how you can display the attributes in table format.
Table 5-7: Elements Used for Creating a Related Model Subview

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;related-model-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;related-model-subview&gt;</td>
<td>The title of the subview.</td>
</tr>
<tr>
<td>&lt;model-type&gt;</td>
<td>&lt;related-model-subview&gt;</td>
<td>The model type to which the subviews will pertain.</td>
</tr>
<tr>
<td>&lt;subviews&gt;</td>
<td>&lt;related-model-subview&gt;</td>
<td>Enables you to add a prebuilt table-subview or field-subview to the detail components that reside within the subview. Values: table-subview and/or field-subview</td>
</tr>
<tr>
<td>&lt;criteria&gt;</td>
<td>&lt;related-model-subview&gt;</td>
<td>The search criteria used to find the related models.</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;related-model-subview&gt;</td>
<td>Associates a privilege to the subview. If the user is not given this privilege, the subview will not be displayed for that user. See “Creating Custom Privileges” on page 77 for more information.</td>
</tr>
</tbody>
</table>

Adding a Related Models Table Subview

You can use the <related-models-table-subview> to display a table of models that are associated with the current selected model based on a specified search criteria.

In the example below, the <subviews> element is used to place a view within a view allowing multiple views to be nested within each other. Each column in the table represents the value of an attribute for each of the models that have passed the search criteria.
Chapter 5: Customizing a Model’s Information View

Table 5-8: Elements Used for Creating a Related Model Table Subview

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;related-model-table-subview&gt;</td>
<td>&lt;subviews&gt;</td>
<td>If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;related-model-table-subview&gt;</td>
<td>The title of the subview</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;related-model-table-subview&gt;</td>
<td>Associates a privilege to the subview. If the user is not given this privilege, the subview will not be displayed for that user. See &quot;Creating Custom Privileges&quot; on page 77 for more information.</td>
</tr>
<tr>
<td>&lt;table&gt;</td>
<td>&lt;related-model-table-subview&gt;</td>
<td>This elements and its sub-elements define the table. See Table 3-1: Elements Used for Modifying a Table page 27 for a list of sub-elements.</td>
</tr>
<tr>
<td>&lt;criteria&gt;</td>
<td>&lt;related-model-table-subview&gt;</td>
<td>The search criteria used to find the related models.</td>
</tr>
</tbody>
</table>

Related-Model-Table Subview Example (demo-details-config.xml)

```xml
<subviews>
  <related-models-table-subview>
    <title>Demo Table Title</title>
    <criteria>demo-search-criteria</criteria>
    <table>demo-table-config</table>
  </related-models-table-subview>
</subviews>
```

Referenced Criteria XML (demo-search-criteria.xml)

```xml
<search-criteria id="demo-search-criteria"
  xmlns:api="http://www.aprisma.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.aprisma.com
  ../../../common/schema/search-criteria-config.xsd">
  <child-models>
    <application-specified-models/>
    <relation>Collects</relation>
  </child-models>
</search-criteria>
```

Referenced Table XML (demo-table-config.xml)

```xml
<?xml version="1.0" encoding="utf-8"?>
<table id="demo-table-config"
  xmlns:api="http://www.aprisma.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.aprisma.com
  ../../../common/schema/table-config.xsd">
  <swing-header-row-template>
```
Creating an Information Configuration File

Defining a Subview Group

You can group together one or more subviews under a single collapsible group using the `<subview-group>` element.

```
<subview-group>
  <title>Subview Group Title</title>
  <display-if>
    <expression>
      attrInt(AttributeID.MTYPE_HANDLE) == 0x3cc0002
    </expression>
  </display-if>
  <subviews>
    <table-subview idref="example-table1-config">
      <title>Example Sub View #1</title>
    </table-subview>
    <table-subview idref="example-table2-config">
      <title>Example Sub View #2</title>
    </table-subview>
  </subviews>
</subview-group>
```

This example generates a subview group similar to the one shown in Figure 5-5 on page 74.
Figure 5-5: A Subview Group

Use the elements shown in Table 5-9 on page 74 to create a subview group.

Table 5-9: Elements Used to Create a Subview Group

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;subview-group&gt;</td>
<td>&lt;subviews&gt;</td>
<td>If you set the expanded attribute of this element to true, the subview will be expanded by default. The idref attribute enables you to associate an XML file that defines the data for this subview. The data for the subview does not have to be in another file, it is done for organizational purposes only.</td>
</tr>
<tr>
<td>&lt;title&gt;</td>
<td>&lt;subview-group&gt;</td>
<td>The title of the subview group. In the example above, this is Subview Group Title</td>
</tr>
<tr>
<td>&lt;privilege&gt;</td>
<td>&lt;subview-group&gt;</td>
<td>Associates a privilege to the subview group. If the user is not given this privilege, the subview group will not be displayed for that user. See &quot;Creating Custom Privileges&quot; on page 77 for more information.</td>
</tr>
<tr>
<td>&lt;display-if&gt;</td>
<td>&lt;subview-group&gt;</td>
<td>Adds an expression that will determine whether or not the group will be visible.</td>
</tr>
<tr>
<td>&lt;subviews&gt;</td>
<td>&lt;subview-group&gt;</td>
<td>Adds any type of subview (besides group) to this subview group.</td>
</tr>
</tbody>
</table>

Associating the Information Configuration File with the Model Class or Model Type

Once you have created the Information view with an Information Configuration file, you must follow the instructions below to associate Information Configuration file with the model type or model class.
Procedure

1. If it does not already exist there, copy the `<Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/custom-app-config.xml` to the `<Install>/WebApps/custom/console/config` directory.

2. Open this file with a text editor.

3. Add the following block of XML code to link the appropriate model type(s) or model class(es) to the Information Configuration file. Use the XML elements shown to define the information appropriate to your model type or model class. For a definition of each of these elements, see “Registering a Model Icon” on page 36.

   ```xml
   <contents-registry>
      <icon-reg-id>your-icon-registration</icon-reg-id>
      <tooltip-config>your-tooltip-config</tooltip-config>
      <information-config>your-information-config-file</information-config>
      <model-class>your-model-class</model-class>
   </contents-registry>
   ``

4. Save and close the `custom-app-config.xml` file.

5. Restart the OneClick client for your changes to take effect.
Chapter 6: Creating Custom Privileges

This chapter describes how to restrict access to menu items, attributes, and subviews.

In this chapter

- “Overview”
- “Defining a Custom Privilege” on page 77
- “Referencing the Privilege when Defining a Menu Item, Column, or Subview” on page 81

Overview

You define each new privilege in the custom-privileges.xml file. This file registers custom privileges that can be applied to menu items (see “Customizing the OneClick Console Menu” on page 11), columns (see “Customizing Tables” on page 25), or subviews (see “Customizing a Model’s Information View” on page 61). If an Administrator has not assigned the privilege to a user, that user will not be able to access the menu item, column, or subview.

Defining a Custom Privilege

You can define one or more custom privileges in the custom-privileges.xml file using the following procedure:

Procedure

1. If it does not already exist there, copy the <Install>/WebApps/tomcat/webapps/spectrum/WEB-INF/console/config/custom-privileges.xml to the <Install>/WebApps/custom/console/config directory.

2. Open this file with a text editor.

3. All new privileges must be defined inside the <privileges> element. This is the root element for the file.

4. Create your new privilege using the elements show in Table 6-1 on page 78. The following example defines the launch-app privilege, as shown in Figure 6-1 on page 79.

```xml
<privileges>
  <launch-app type="write">
```
Note that when you create a privilege, you are creating a new XML element. In the example above, the `<launch-app>` element creates the launch-app privilege. The `type` attribute defines which default role the privilege will be assigned to. There are two possible values, “read” and “write”. A privilege with the “read” type will be assigned to the OperatorRO role, and a privilege with the “write” type will be assigned to the OperatorRW role.

5. Save and close the `custom-privileges.xml` file.

6. In order for changes made to the `custom-privileges.xml` file to be available in OneClick. You must restart the Tomcat Web Server software and then restart OneClick.

7. You can now use this privilege when creating a menu item, column, or subview. See “Referencing the Privilege when Defining a Menu Item, Column, or Subview” on page 81 for instructions on linking the privilege with the menu item, column, or subview.

### Table 6-1: Elements Used to Create a Privilege

<table>
<thead>
<tr>
<th>Element</th>
<th>Parent Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;privileges&gt;</code></td>
<td>NA</td>
<td>This is the root element for the <code>custom-privileges.xml</code> file.</td>
</tr>
<tr>
<td><code>&lt;your_privilege_name&gt;</code></td>
<td><code>&lt;privileges&gt;</code> or <code>&lt;your_group_name&gt;</code></td>
<td>This element defines the privilege. You create a new element for each privilege you define. The <code>type</code> attribute for this element defines the default role that you are assigning the privilege to. Possible values for the <code>type</code> attribute are &quot;read&quot; or &quot;write&quot;. If you are grouping privileges together, you will place all defined privileges for that group within the element that defines your group. See “Grouping Privileges” on page 80 for instructions.</td>
</tr>
<tr>
<td><code>&lt;label&gt;</code></td>
<td><code>&lt;your_privilege_name&gt;</code></td>
<td>The name of the privilege, which will be shown on the privilege list.</td>
</tr>
<tr>
<td><code>&lt;desc&gt;</code></td>
<td><code>&lt;your_privilege_name&gt;</code></td>
<td>The description of the privilege.</td>
</tr>
<tr>
<td><code>&lt;model-view-attr&gt;</code></td>
<td><code>&lt;privileges&gt;</code></td>
<td>See “Restricting Access to Attribute Values in Model Subviews” on page 79.</td>
</tr>
<tr>
<td><code>&lt;model-write-attr&gt;</code></td>
<td><code>&lt;privileges&gt;</code></td>
<td>See “Restricting Access to Attribute Values in Model Subviews” on page 79.</td>
</tr>
<tr>
<td><code>&lt;group&gt;</code></td>
<td><code>&lt;your_privilege_name&gt;</code></td>
<td>If you would like your privilege to appear in one of the existing groups, use the <code>&lt;group&gt;</code> element. The <code>scope</code> attribute defines the scope of the group. Below is a list of existing groups and their scope values: <code>&lt;group scope=&quot;alarm&quot;&gt; alarm-manager &lt;/group&gt;</code> <code>&lt;group scope=&quot;topo&quot;&gt; tools &lt;/group&gt;</code> <code>&lt;group scope=&quot;topo&quot;&gt; model-tab &lt;/group&gt;</code> <code>&lt;group scope=&quot;topo&quot;&gt; model-view-group &lt;/group&gt;</code> <code>&lt;group scope=&quot;topo&quot;&gt; model-write-group &lt;/group&gt;</code> To create a new group, see “Grouping Privileges” on page 80.</td>
</tr>
</tbody>
</table>
Defining a Custom Privilege

Figure 6-1: OneClick Privilege List

Restricting Access to Attribute Values in Model Subviews

You can restrict the users access to certain attributes using the `<model-view-attr>` and `<model-write-attr>` elements, where `attr` is equal to the attribute ID of the attribute you wish to restrict. These elements are used in the custom-privileges.xml file and will regulate the attributes that show up in the OneClick Privilege list’s Model Management>View Attributes folder and the Model Management>Model Write folder.

The `<model-view-attr>` element enables you to create a privilege that determines whether or not a user can see an attribute. For example, if you added the following XML to the custom-privileges.xml file, you will create a privilege called Community Name. This privilege restricts view access to attribute 10024, community name. This privilege will appear in the Model Management > View Attributes folder as specified with the `<group>` element. If the user does not have this privilege in any access group, they will not be able to see the community name attribute.

```
<model-view-attr type="read">
    <label>Community Name</label>
    <group scope="topo">model-view-group</group>
</model-view-attr>
```

The `<model-write-attr>` element enables you to create a privilege that determines whether or not a user can edit an attribute. For example, if you added the following XML to the custom-privileges.xml file, you will create a privilege called Community Name. This privilege restricts write access to attribute 10024, community name. This privilege will appear in the Model Management > Model Write folder as specified with the `<group>` element. If the user does not have this privilege in any access group, they will not be able to edit the community name attribute.

```
<model-write-attr type="write">
    <label>Community Name</label>
    <group scope="topo">model-view-group</group>
</model-write-attr>
```
Chapter 6: Creating Custom Privileges

Grouping Privileges

If you would like to group privileges together, you can create a group in the custom-privileges.xml file. To specify a group, nest the element that defines the privilege (<your_privilege_name>) between the element that defines the group (<your_group_name>). The group’s <label> element defines the name that represents the group in the privileges tree (see Figure 6-2 on page 81).

In the following example, the <my-tools> element creates a group in which privileges can be nested. The value defined for the group’s <label> is “My-Tools Folder”. This will create a “My-Tools Folder” group in the privileges list as shown in Figure 6-2 on page 81. The <launch-app> and <launch-web> privileges will appear in this group.

```
<privilege>
  <my-tools>
    <launch-app type="read">
      <label>Launch Apps</label>
      <desc>Ability to launch Applications.</desc>
    </launch-app>
    <launch-web type="read">
      <label>Launch Web</label>
      <desc>Ability to launch Web URLS.</desc>
    </launch-web>
  </my-tools>
</privilege>
```

**Note:** In order for changes made to the custom-privileges.xml file to be available in OneClick. You must restart the Tomcat Web Server software and then restart OneClick.
Referencing the Privilege when Defining a Menu Item, Column, or Subview

When you create a menu item, column, or subview, you can use the `<privilege>` element to reference a custom privilege you have defined in the `custom-privileges.xml` file. For example, if you have defined the “launch-app” privilege in the `custom-privileges.xml` file, you can use the following XML when defining a menu item, column, or subview:

```
<privilege>
  <name>launch-app</name>
</privilege>
```

This will associate the “launch-app” privilege with the menu item, column, or subview. The user must have an associated role that grants the “launch-app” privilege in order for the menu item, column, or subview to be displayed. If granted, the menu item is always enabled. See the OneClick Administration Guide (5166) for information about privileges and roles.
Chapter 7: XML Usage Common to All Customization Files

This chapter explains common XML elements and strategies that can be used across customization files.

In this chapter

• “Using Parameters”
• “Using a Select Case” on page 84
• “Manipulating the Output of Attributes with Renderers” on page 84
• “Using Expressions” on page 87
• “Referencing XML Files” on page 89
• “Referencing Images” on page 90

Using Parameters

You can use the <param> element in a number of different instances to reference parameter values within a OneClick XML file. Here are several common cases where you will likely use the <param> element.

• If you need to pass a parameter to a web page, use the <param> element as a child element of the <url> element. See “Launching a Browser” on page 21 for an example.

• If you need to pass a parameter to an application, use the <param> element as a child element of the <launch-application> element. See “Launching an Application” on page 22 for an example.

• If you need to pass a parameter to a command, use the <param> element as a child element of the <command> element. See “Launching a Browser” on page 21, “Launching an Application” on page 22, and “Launching a Web Server Script” on page 23.

• If you need to format a series of values, use the <param> element in conjunction with standard HTML formatting elements. See “Defining Model Icon Tooltips” on page 57 for an example.
• If you need to manipulate the value of an attribute, you may need to use the `<param>` element when accessing one of the renderers. See “Manipulating the Output of Attributes with Renderers” on page 84 for a complete explanation.

Using a Select Case

If you want to conditionally display something in the OneClick interface, you may use the `<select>` and the `<case>` elements to create a decision structure similar to those used in many programming languages. The `<select>` and `<case>` elements are used as follows:

```xml
<select>
  <case>
    <expression>the expression to evaluate</expression>
    <yield>what to yield if the expression is true</yield>
  </case>
  <case>
    <expression>the expression to evaluate</expression>
    <yield>what to yield if the expression is true</yield>
  </case>
  ...
  <default>what to yield if no matches are found</default>
</select>
```

The “Sample Image Definition File” on page 49 shows an example of the `<select>` and `<case>` elements used to select the image to be displayed on a OneClick device model icon depending on the model’s condition.

Manipulating the Output of Attributes with Renderers

There are several built in attribute renderers that you can use to manipulate how the attributes you have specified in a OneClick table are displayed. You use the `<renderer>` element to access one of these renderers. The text of the element must be a fully-qualified Java class name, each allowable Java class name is explained below.

**Note:** You will need some background in programming to fully understand the renderer concepts presented below.

You can pass parameters to a renderer using the `<param>` element. The text of the `<param>` element is the parameter value. A `<param>` element must have a name attribute that specifies the name of the parameter. Parameter names follow the Java bean paradigm – there must be corresponding “get” and “set” methods in the renderer class.

**Example**

```xml
<renderer>
  <param name="trueTag">Enabled</param>
  <param name="falseTag">Disabled</param>
  com.aprisma.spectrum.app.util.render.BooleanRenderer
</renderer>
```

This example specifies the BooleanRenderer with parameter `trueTag` set to No and parameter `falseTag` set to Yes. The BooleanRenderer renderer has corresponding methods named...
getTrueTag, setTrueTag, getFalseTag and setFalseTag. Only the "set" methods are actually used – the "get" methods are necessary to satisfy Java’s bean introspector.

**Boolean Renderer**

The class name for the boolean renderer is com.aprisma.spectrum.app.util.render.BooleanRenderer. This renderer outputs an enumerated String for an input Boolean value. By default, "Yes" is rendered for TRUE and "No" is rendered for FALSE but other elements or text may be specified via the following parameters:

- **trueTag** – the tag or text to render for TRUE
- **falseTag** – the tag or text to render for FALSE

The following example reverses the TRUE/FALSE output:

```html
<renderer>
  <param name="trueTag">No</param>
  <param name="falseTag">Yes</param>
  com.aprisma.spectrum.app.util.render.BooleanRenderer
</renderer>
```

If the input value is TRUE, "No" is rendered and if FALSE, "Yes" is rendered.

**Commented Text Renderer**

The class name for the commented text renderer is com.aprisma.spectrum.app.util.render.CommentedTextRenderer. This renderer strips off the HTML-commented prefix that is added by some renderers (e.g. DateRenderer). It searches for the first occurrence of the ending character sequence of an HTML comment, e.g. `<!---comment text --`, and returns the rest of the string.

**Date Renderer**

The class name for the date renderer is com.aprisma.spectrum.app.util.render.DateRenderer. This renderer outputs a date/time using Java’s DateFormat. If the input to the renderer is a long integer (i.e. of type java.lang.Long), it is assumed to represent the date/time in milliseconds. If the input is any other numeric type, it is assumed to be an integer representing the date/time in seconds. Otherwise the only other valid input type is java.util.Date. The output string is prefixed by the numeric date/time value enclosed in HTML comments (`<!---comment text --->`). An example output would be:

```html
<!---1089808869000-->Jul 14, 2004 8:41:09 AM EDT
```

You use the numeric value prefix in the comments tag for sorting. Without the prefix, SPECTRUM would sorting on the formatted date/time string, and this would not work correctly. Therefore, you should only use the DateRenderer in the `<content>` element section of a column. To strip off the prefix for display, use the CommentedTextRenderer in the `<swing-cell-template>` section. For example:

```html
<content>
  <attribute>0x11620</attribute>
  <!–an attribute that contains an integer date/time in seconds >
  <renderer>
    com.aprisma.spectrum.app.util.render.DateRenderer
  </renderer>
</content>
```
Enumerated Attribute Renderer

The classname for the enumerated attribute renderer is `com.aprisma.spectrum.app.util.render.EnumeratedAttrRenderer`. This renderer outputs an enumerated String for an attribute value. The renderer obtains the enumerations from the SPECTRUM database. You must specify the attribute ID via the “attrID” parameter. This renderer is most commonly preceded by an `<attribute>` element with the same attribute ID as the “attrID” parameter. The following sample XML renders the enumerated value for the Model_Class attribute (ID 0x11ee8):

```
<attribute>0x11ee8</attribute>
<renderer>
  <param name="attrID">0x11ee8</param>
  com.aprisma.spectrum.app.util.render.EnumeratedAttrRenderer
</renderer>
```

Lists Renderer

The classname for the list renderer is `com.aprisma.spectrum.app.util.render.ListRenderer`. This renderer outputs the components of a Java Collection or an array of any type as a comma separated string.

Null Renderer

The classname for the null renderer is `com.aprisma.spectrum.app.util.render.NullRenderer`. This renderer outputs a null input value as an empty string.

Object ID Renderer

The classname for this renderer is `com.aprisma.spectrum.app.util.render.ObjectIDRenderer`. This renderer outputs an object identifier (OID). The input value is expected to be of type CsObjectId. The following parameters are supported:

- `term` - an integer value that specifies the index of a particular term of the OID to render.
- `startTerm` - an integer value that specifies the index of the first term of the OID to render.
- `endTerm` - an integer value that specifies the index of the last term of the OID to render.

The term indices start at 1. If you specify the `startTerm` without the `endTerm`, then the portion of the OID from the `startTerm` to the last term of the OID is rendered. If you specify the `endTerm` without the `startTerm`, then the portion of the OID from the first term to the `endTerm` is rendered. The ObjectIDRenderer is most commonly used to render the row instance of a MIB table. You obtain the row instance via the `getRowId()` method in an `<expression>` element. You can then pass the result to the ObjectIDRenderer. For example, the following column renders the first term of the row instance:

```
<column>
  <name>com.aprisma.spectrum.app.topo.client.ifIndex</name>
  <content>
    <expression>getRowId()</expression>
  </content>
</column>
```
Using Expressions

The following example renders terms 5 thru 8:

```
<column>
  <name>com.aprisma.spectrum.app.topo.client.NetworkAddr</name>
  <content>
    <expression>getRowId()</expression>
    <renderer>
      <param name="startTerm">5</param>
      <param name="endTerm">8</param>
      com.aprisma.spectrum.app.util.render.ObjectIDRenderer
    </renderer>
  </content>
</column>
```

The following example combines an expression with the ObjectID renderer to enable you to display the last term in of OID value in a table:

```
<column>
  <name>com.aprisma.spectrum.app.topo.client.ifIndex</name>
  <content>
    <expression>
      ((com.aprisma.spectrum.global.CsObjectID)value()).get_sub_oid((com.aprisma.spectrum.global.CsObjectID)value()).get_term_count(),
      ((com.aprisma.spectrum.global.CsObjectID)value()).get_term_count()
    </expression>
    <renderer>
      <param name="term">1</param>
      com.aprisma.spectrum.app.util.render.ObjectIDRenderer
    </renderer>
  </content>
</column>
```

The Round Number Renderer

The classname for this renderer is com.aprisma.spectrum.app.util.render.RoundNumberRenderer. This renderer outputs a number rounded to the nearest 100th (or 2 decimal places).

System Up Time Renderer

The classname for this renderer is com.aprisma.spectrum.app.util.render.SysUpTimeRenderer. This renderer outputs a numeric time value represented in one hundredths of a second. This time representation is typically used in MIB objects, for example sysUpTime. The output is in terms of days, hours, and minutes, i.e. 30 days 1 hr 55 min.

Using Expressions

When you are customizing the OneClick interface, there are several places where you may want to use an expression to display a calculated value. For example, you may want to display a calculated value in a table or subview. The section below explains how to use expressions to manipulate attribute information.
Note: Expressions are created using standard Java expressions. You must be familiar with Java code in order to implement the following instructions creating expressions in the OneClick XML files. If you are not familiar with Java code, you should refer to a Java reference before attempting to create expressions when customizing OneClick files.

Manipulating Attribute Information

The most common use of an expression is to manipulate attribute information. The attribute information available is dependent upon the OneClick context in which you are using the expression.

You can use the following methods in the context of an expression to retrieve attribute information:

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>attr(attrID)</td>
<td>Retrieves the attribute value</td>
</tr>
<tr>
<td>attrBoolean(attrID)</td>
<td>Retrieves the boolean attribute</td>
</tr>
<tr>
<td>attrByte(attrID)</td>
<td>Retrieves the byte attribute</td>
</tr>
<tr>
<td>attrChar(attrID)</td>
<td>Retrieves the character attribute</td>
</tr>
<tr>
<td>attrDouble(attrID)</td>
<td>Retrieves the double attribute</td>
</tr>
<tr>
<td>attrFloat(attrID)</td>
<td>Retrieves the float attribute</td>
</tr>
<tr>
<td>attrInt(attrID)</td>
<td>Retrieves the integer attribute</td>
</tr>
<tr>
<td>attrLong(attrID)</td>
<td>Retrieves the long attribute</td>
</tr>
<tr>
<td>attrShort(attrID)</td>
<td>Retrieves the short attribute</td>
</tr>
</tbody>
</table>

The following example shows a column configuration that displays the contact person for a device. In this example, an expression displays the attribute 0x23000c (AttributeID.CONTACT_PERSON) if the attribute 0x10b5a (AttributeID.SYS_CONTACT) is null or has no value.

```xml
<column id="column-contact-config"
	xmlns ="http://www.aprisma.com"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.aprisma.com
	../../common/schema/column-config.xsd">
  <name>Contact Person</name>
  <content>
    <expression>
      ( attr(AttributeID.SYS_CONTACT) == null ||
        ((String)attr(AttributeID.SYS_CONTACT)).length() == 0) ?
        attr(AttributeID.CONTACT_PERSON) : value()
    </expression>
  </content>
</column>
```

An alternative to the above example is to use the attribute renderer to retrieve the value of the SYS_CONTACT attribute. You can then access the value returned via an expression that uses the value() method.
Referencing XML Files

As you are customizing OneClick XML files, you may find it necessary to split a single XML file into two or more XML files. There are two major reasons you may want to do this:

- Some XML files may become so complex that they become unreadable. In this case breaking the XML file down into two or more files assists you in keeping your code organized making it easily editable in the future.
- You may want to reuse certain sections of XML code. If you put this XML code in a separate XML file you can reference it from multiple file.

When you split up an XML file for either readability or for code reuse, you will use the standard XML id and idref attributes to label and reference the split up code. (For information on XML standards include id and idref, see www.w3.org).

Example

The following example shows a portion of an XML file used to define a table. Rather than defining each column in the same file that defines the entire table, the example uses separate files to define the first two columns in the table. The example uses the idref attribute with each <column> element to link to the file that defines the column.

Referencing XML Files

As you are customizing OneClick XML files, you may find it necessary to split a single XML file into two or more XML files. There are two major reasons you may want to do this:

- Some XML files may become so complex that they become unreadable. In this case breaking the XML file down into two or more files assists you in keeping your code organized making it easily editable in the future.
- You may want to reuse certain sections of XML code. If you put this XML code in a separate XML file you can reference it from multiple file.

When you split up an XML file for either readability or for code reuse, you will use the standard XML id and idref attributes to label and reference the split up code. (For information on XML standards include id and idref, see www.w3.org).

Example

The following example shows a portion of an XML file used to define a table. Rather than defining each column in the same file that defines the entire table, the example uses separate files to define the first two columns in the table. The example uses the idref attribute with each <column> element to link to the file that defines the column.
Chapter 7: XML Usage Common to All Customization Files

```xml
xmlns="http://www.aprisma.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.aprisma.com
      ../../../common/schema/table-config.xsd">
  <swing-row-template>
    <enumerated-color idref="alternatingrow-color-config"/>
  </swing-row-template>
  <swing-table-template>
    <show-vertical-lines>true</show-vertical-lines>
    <show-horizontal-lines>false</show-horizontal-lines>
  </swing-table-template>
  <swing-header-row-template>
    <static-color idref="row-header-color-config"/>
  </swing-header-row-template>
  <column-list>
    <column idref="column-servicestate-config"/>
    <column idref="column-modelname-config">
      <default-width>300</default-width>
    </column>
  </column-list>
</table>
```

The first column is defined in the `column-servicestate-config.xml` file. The beginning portion of this file is shown below. Note the `id` attribute used with the `<column>` element to define this file as “column-servicestate-config”.

```xml
<column id="column-servicestate-config"
  xmlns="http://www.aprisma.com"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.aprisma.com
      ../../../common/schema/table-config.xsd">
```

**Referencing Images**

In some instances you may need to reference image files from within your XML. You must place all image files that you would like to reference in the `<Install>/WebApps/tomcat/webapps/spectrum/images` directory. When you reference an image placed in this directory you must express the path starting from the images directory, i.e. `images/myimage.png`. See “Sample Icon Configuration File #1” on page 42 for an example.
Index

A
accelerator 15
action 15
alarm attributes 20
app-config 36
application-subview 65

cellicon-config 39
column 27
column-list 27
command 17
custom-app-config.xml 10, 35
custom-menu-config.xml 10, 11
default-sort 28
directory structure 10
Directory, alarm/config 10
Directory, common/config 10
Directory, Custom 10
Directory, topo/config 9
display-if 74
dynamic-cellicon 39

element - accelerator 16
element - action 16
element - context 17
element - filter 18
element - item 13
element - launch-web-server-script 23
element - menu 12
element - toolbar-image 15
element - toolbar-image-disabled 15
element - toolbar-image-rollover 15
Ellipse 44
Expressions 87

F
field-subview 65
filter 17

H
hot-key 15

I
icon configuration 53
Icon Configuration file 42, 45
icon-config 40
icon-reg-id 37, 39
ifIndex 33
Image components 47
image types 15
Images 90
information-config 37
Interfaces Table 32, 33
is-derived-from 37
Index

item 15

K
keyboard accelerators 16

L
launching applications 21
launching browsers 20
launching scripts 23
Line 45
line-color 27

M
menu 15
Model-class 37
Model-type 37
model-view-attr 79
model-write-attr 79

N
numerated-color 27

O
off-page reference 39
on-page reference 39

P
parameter 58
pipe location 46
pipe-connection 41
platform 17
platforms 23
Polygon 44
preferred-height 27
preferred-width 27
preserve customizations during an upgrade 10
privilege 15, 74

R
Rectangle 43
registry 36
related-model-subview 65

related-model-table-subview 65
renderer 27, 59, 84
Rounded Rectangle 43

S
select case 84
selection components 55
separator 15
shape 40
show-horizontal-lines 27
show-icon 65
show-labels 65
show-tree-lines 27
show-vertical-lines 27
sort 31
sort-column 28
Specifying a Username 21
static-cellicon 39
static-color 27
stroke 41
subview-group 65
subviews 64, 74
swing-cell-template 28
swing-header-row-template 27
swing-row-template 27
swing-table-template 27

T
table columns 25
table-subview 65
text components 51
Theme 39
theme configuration 37
theme-config 36, 39
title 69, 74
toolbar-image 15
toolbar-image-disabled 15
toolbar-image-rollover 15
tooltip 57
tooltip-config 36, 37, 59
topo-app-config.xml 10

U
USER_PARAMETER_NAME 21
Username 21