Netcool/Impact
Version 6.1.1.5

Operator View Guide

SC27-4853-01

Netcool/Impact
Version 6.1.1.5

Operator View Guide

SC27-4853-01

Note
Before using this information and the product it supports, read the information in "Notices".

Edition notice

This edition applies to version 6.1.1.5 of IBM Tivoli Netcool/Impact and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Operator View Guide

The Netcool/Impact *Operator View Guide* contains instructions on creating operator views.

Intended audience

This publication is for users who are responsible for creating operator views.

Publications

This section lists publications in the Netcool/Impact library and related documents. The section also describes how to access Tivoli® publications online and how to order Tivoli publications.

Netcool/Impact library

- *Quick Start Guide*, CF39PML

  Provides concise information about installing and running Netcool/Impact for the first time.
- *Administration Guide*, SC14755901

  Provides information about installing, running and monitoring the product.
- *User Interface Guide*, SC27485101

  Provides instructions for using the Graphical User Interface (GUI).
- *Policy Reference Guide*, SC14756101

  Contains complete description and reference information for the Impact Policy Language (IPL).
- *DSA Reference Guide*, SC27485201

  Provides information about data source adaptors (DSAs).
- *Operator View Guide*, SC27485301

  Provides information about creating operator views.
- *Solutions Guide*, SC14756001

  Provides end-to-end information about using features of Netcool/Impact.
- *Integrations Guide*, SC27485401

  Contains instructions for integrating Netcool/Impact with other IBM® software and other vendor software.
- *Troubleshooting Guide*, GC27485501

  Provides information about troubleshooting the installation, customization, starting, and maintaining Netcool/Impact.

Accessing terminology online

The IBM Terminology Web site consolidates the terminology from IBM product libraries in one convenient location. You can access the Terminology Web site at the following Web address:

http://www.ibm.com/software/globalization/terminology

Accessing publications online

Publications are available from the following locations:

- The *Quick Start* DVD contains the Quick Start Guide. Refer to the readme file on the DVD for instructions on how to access the documentation.
- Tivoli Information Center web site at http://publib.boulder.ibm.com/infocenter/tivihelp/v8r1/topic/com.ibm.netcoolimpact.doc6.1.1/welcome.html. IBM posts publications for all Tivoli products, as they become available and whenever they are updated to the Tivoli Information Center Web site.

  Note: If you print PDF documents on paper other than letter-sized paper, set the option in the **File → Print** window that allows Adobe Reader to print letter-sized pages on your local paper.
- Tivoli Documentation Central at http://www.ibm.com/tivoli/documentation. You can access publications of the previous and current versions of Netcool/Impact from Tivoli Documentation Central.
- The Netcool/Impact wiki contains additional short documents and additional information and is available at https://www.ibm.com/developerworks/mydeveloperworks/wikis/home?lang=en#/wiki/Tivoli%20Netcool%20Impact.

Ordering publications

You can order many Tivoli publications online at http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss.

You can also order by telephone by calling one of these numbers:

- In the United States: 800-879-2755
- In Canada: 800-426-4968

In other countries, contact your software account representative to order Tivoli publications. To locate the telephone number of your local representative, perform the following steps:

1. Go to http://www.elink.ibmlink.ibm.com/publications/servlet/pbi.wss.
2. Select your country from the list and click **Go**.
3. Click **About this site** in the main panel to see an information page that includes the telephone number of your local representative.

Accessibility

Accessibility features help users with a physical disability, such as restricted mobility or limited vision, to use software products successfully. With this product, you can use assistive technologies to hear and navigate the interface. You can also use the keyboard instead of the mouse to operate all features of the graphical user interface.

For additional information, see Appendix A, “Accessibility,” on page 139.

Tivoli technical training

For Tivoli technical training information, refer to the following IBM Tivoli Education Web site at http://www.ibm.com/software/tivoli/education.

Support for problem solving

If you have a problem with your IBM software, you want to resolve it quickly. This section describes the following options for obtaining support for IBM software products:

- "Obtaining fixes"
- "Receiving weekly support updates"
- "Contacting IBM Software Support" on page viii

Obtaining fixes

A product fix might be available to resolve your problem. To determine which fixes are available for your Tivoli software product, follow these steps:

1. Go to the IBM Software Support Web site at http://www.ibm.com/software/support.
2. Navigate to the **Downloads** page.
3. Follow the instructions to locate the fix you want to download.
4. If there is no **Download** heading for your product, supply a search term, error code, or APAR number in the search field.

For more information about the types of fixes that are available, see the *IBM Software Support Handbook* at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html.

Receiving weekly support updates

To receive weekly e-mail notifications about fixes and other software support news, follow these steps:

1. Go to the IBM Software Support Web site at http://www.ibm.com/software/support.
2. Click the **My IBM** in the toobar. Click **My technical support**.
3. If you have already registered for **My technical support**, sign in and skip to the next step. If you have not registered, click **register now**. Complete the registration form using your e-mail address as your IBM ID and click **Submit**.
4. The **Edit profile** tab is displayed.
5. In the first list under **Products**, select **Software**. In the second list, select a product category (for example, **Systems and Asset Management**). In the third list, select a product sub-category (for example, **Application Performance & Availability** or **Systems Performance**). A list of applicable products is displayed.
6. Select the products for which you want to receive updates.
7. Click **Add products**.
8. After selecting all products that are of interest to you, click **Subscribe to email** on the **Edit profile** tab.
9. In the **Documents** list, select **Software**.
10. Select **Please send these documents by weekly email**.
11. Update your e-mail address as needed.
12. Select the types of documents you want to receive.
13. Click **Update**.

If you experience problems with the **My technical support** feature, you can obtain help in one of the following ways:

Online

Send an e-mail message to erchelp@u.ibm.com, describing your problem.

By phone

Call 1-800-IBM-4You (1-800-426-4409).

World Wide Registration Help desk

For word wide support information check the details in the following link: https://www.ibm.com/account/profile/us?page=reghelpdesk

Contacting IBM Software Support

Before contacting IBM Software Support, your company must have an active IBM software maintenance contract, and you must be authorized to submit problems to IBM. The type of software maintenance contract that you need depends on the type of product you have:

- For IBM distributed software products (including, but not limited to, Tivoli, Lotus®, and Rational® products, and DB2® and WebSphere® products that run on Windows or UNIX operating systems), enroll in Passport Advantage® in one of the following ways:

  Online

  Go to the Passport Advantage Web site at http://www-306.ibm.com/software/howtobuy/passportadvantage/pao_customers.htm .

  By phone

  For the phone number to call in your country, go to the IBM Worldwide IBM Registration Helpdesk Web site at https://www.ibm.com/account/profile/us?page=reghelpdesk.
- For customers with Subscription and Support (S & S) contracts, go to the Software Service Request Web site at https://techsupport.services.ibm.com/ssr/login.
- For customers with IBMLink, CATIA, Linux, OS/390®, iSeries, pSeries, zSeries, and other support agreements, go to the IBM Support Line Web site at http://www.ibm.com/services/us/index.wss/so/its/a1000030/dt006.
- For IBM eServer™ software products (including, but not limited to, DB2 and WebSphere products that run in zSeries, pSeries, and iSeries environments), you can purchase a software maintenance agreement by working directly with an IBM sales representative or an IBM Business Partner. For more information about support for eServer software products, go to the IBM Technical Support Advantage Web site at http://www.ibm.com/servers/eserver/techsupport.html.

If you are not sure what type of software maintenance contract you need, call 1-800-IBMSERV (1-800-426-7378) in the United States. From other countries, go to the contacts page of the *IBM Software Support Handbook* on the Web at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html and click the name of your geographic region for phone numbers of people who provide support for your location.

To contact IBM Software support, follow these steps:

1. “Determining the business impact” on page ix
2. “Describing problems and gathering information” on page ix
3. “Submitting problems” on page ix

Determining the business impact

When you report a problem to IBM, you are asked to supply a severity level. Use the following criteria to understand and assess the business impact of the problem that you are reporting:

Severity 1
: The problem has a *critical* business impact. You are unable to use the program, resulting in a critical impact on operations. This condition requires an immediate solution.

Severity 2
: The problem has a *significant* business impact. The program is usable, but it is severely limited.

Severity 3
: The problem has *some* business impact. The program is usable, but less significant features (not critical to operations) are unavailable.

Severity 4
: The problem has *minimal* business impact. The problem causes little impact on operations, or a reasonable circumvention to the problem was implemented.

Describing problems and gathering information

When describing a problem to IBM, be as specific as possible. Include all relevant background information so that IBM Software Support specialists can help you solve the problem efficiently. To save time, know the answers to these questions:

- Which software versions were you running when the problem occurred?
- Do you have logs, traces, and messages that are related to the problem symptoms? IBM Software Support is likely to ask for this information.
- Can you re-create the problem? If so, what steps were performed to re-create the problem?
- Did you make any changes to the system? For example, did you make changes to the hardware, operating system, networking software, and so on.
- Are you currently using a workaround for the problem? If so, be prepared to explain the workaround when you report the problem.

Submitting problems

You can submit your problem to IBM Software Support in one of two ways:

Online
: Click **Submit and track problems** on the IBM Software Support site at http://www.ibm.com/software/support/probsub.html. Type your information into the appropriate problem submission form.

By phone
: For the phone number to call in your country, go to the contacts page of the *IBM Software Support Handbook* at http://www14.software.ibm.com/webapp/set2/sas/f/handbook/home.html and click the name of your geographic region.

If the problem you submit is for a software defect or for missing or inaccurate documentation, IBM Software Support creates an Authorized Program Analysis Report (APAR). The APAR describes the problem in detail. Whenever possible, IBM Software Support provides a workaround that you can implement until the APAR is resolved and a fix is delivered. IBM publishes resolved APARs on the Software Support Web site daily, so that other users who experience the same problem can benefit from the same resolution.

Conventions used in this publication

This publication uses several conventions for special terms and actions, operating system-dependent commands and paths, and margin graphics.

Typeface conventions

This publication uses the following typeface conventions:

Bold

- Lowercase commands and mixed case commands that are otherwise difficult to distinguish from surrounding text
- Interface controls (check boxes, push buttons, radio buttons, spin buttons, fields, folders, icons, list boxes, items inside list boxes, multicolumn lists, containers, menu choices, menu names, tabs, property sheets), labels (such as **Tip:**, and **Operating system considerations:**)
- Keywords and parameters in text

Italic

- Citations (examples: titles of publications, diskettes, and CDs
- Words defined in text (example: a nonswitched line is called a *point-to-point line*)
- Emphasis of words and letters (words as words example: "Use the word *that* to introduce a restrictive clause."; letters as letters example: "The LUN address must start with the letter *L*.")
- New terms in text (except in a definition list): a *view* is a frame in a workspace that contains data.
- Variables and values you must provide: ... where *myname* represents....

`Monospace`

- Examples and code examples
- File names, programming keywords, and other elements that are difficult to distinguish from surrounding text
- Message text and prompts addressed to the user
- Text that the user must type
- Values for arguments or command options

Operating system-dependent variables and paths

This publication uses the UNIX convention for specifying environment variables and for directory notation.

When using the Windows command line, replace **$***variable* with **%***variable***%** for environment variables and replace each forward slash (**/**) with a backslash (\) in directory paths. The names of environment variables are not always the same in the Windows and UNIX environments. For example, %TEMP% in Windows environments is equivalent to $TMPDIR in UNIX environments.

Note: If you are using the bash shell on a Windows system, you can use the UNIX conventions.

Chapter 1. Introduction to operator views

An operator view is a custom Web-based tool that you use to view events and data in real time and to run policies that are based on that data.

Operator views

An operator view is a custom web-based tool that you use to view events and data in real time and to run policies that are based on that data.

The simplest operator views present a basic display of event and business data. More complex operator views can function as individual GUIs that you use to view and interact with event and business data in a wide variety of ways. You can use this kind of GUI to extensively customize an implementation of Netcool/Impact products and other Tivoli Monitoring applications.

Typically, you create operator views to:

- Accept incoming event data from Netcool®/OMNIbus or another application.
- Run a policy that correlates the event data with business data that is stored in your environment.
- Display the correlated business data to a user.
- Run one or more policies based on the event or business data.
- Start another operator view based on the event or business data.

One common way to use an operator view is to configure it to be started from within the Netcool/OMNIbus event list. Netcool/Impact operators can view related business data for an event by right-clicking the event in the event list and viewing the data as displayed in the view. The business data might include service, system, or device information related to the event, or contact information for administrators and customers that are affected by it.

Operator views are not limited to use as Netcool/OMNIbus tools. You can use the operator view feature to create a wide variety of tools that display event and business data to users.

Operator view types

Basic and advanced operator views are supported.

- Basic operator views that you use to display data in a preformatted web page. For more information about basic operator views, see "Basic operator views."
- Advanced operator views that you use to display data using any HTML formatting that you choose. For more information about advanced operator views, see "Advanced operator views" on page 2.

Basic operator views

You use basic operator views to view real-time data in a preformatted web page and to run policies based on that data.

A basic operator view has the following display elements:

Event panel
Displays incoming event information from Netcool/OMNIbus or information from another application that can be expressed in name/value pairs.

Actions panel
You use it to run one or more policies from within the operator view.

Information groups panel
Displays sets of data that you define when you create the view, or when you manually edit the operator view policy.

You create basic operator views using the GUI. The GUI automatically creates the corresponding display page and operator view policy.

If you need to customize the appearance of the view or the type of information displayed in the information group panel, you can manually edit the display page using a text editor. You can edit the operator view policy using the GUI.

Advanced operator views

You use advanced operator views to view real-time data in an HTML-formatted web page and to run policies based on that data.

Unlike basic operator views, which must use the provided preformatted page design, advanced operator views have no restrictions on the appearance of the resulting web page.

You can use any type of HTML formatting to specify how an advanced operator view is displayed and you can display data in an advanced view in any format that is viewable using a web browser. You can also further customize advanced operator views using cascading styles sheets (CSS) and browser scripting languages.

For detailed information about how to create and view advanced operator views, see the *Operator View Guide*.

Operator view components

An overview of the operator view components.

Display page
Text file that contains HTML content and special instructions called smart tags that determine what data to display and how to display it.

Operator view policy
Policy that contains the logic required to retrieve and manipulate the data displayed in the view.

Operator view policy: The operator view policy is a policy that contains the logic required to retrieve and manipulate the data that is displayed in the view. This policy is named `Opview_`*`viewname`*, where *`viewname`* is the name of the operator view. There is one such policy for each operator view. When a request to display an operator view is handled, a corresponding policy is run.

The content of the operator view policy differs, depending on whether it is associated with a basic view or an advanced view.

Basic operator view policies: Basic operator view policies consist of the following elements:

- Variable assignments that specify the position of the event panel and action panel.
- Variable assignments that specify which policies are displayed in the action panel.
- One GetByFilter or GetByKey statement for each information group.

When you create a basic operator view using the GUI, the GUI automatically creates a corresponding policy that contains all the required content. If necessary, you can manually edit the policy after it is created. For more information about editing operator view policies, see "Editing the operator view policy" on page 12.

The following example shows a policy that works with a basic operator view:

```
// This policy generated by Impact Operator View.
// Modify at your own risk!
// Once modified, this policy may no longer be configurable
// through the Impact Operator View GUI.

// LAYOUT PANEL
EventPos="top";
ActionPos="top";

// ACTION PANEL
ActionPanel0="Policy_03";
ActionPanel1="Policy_02";
ActionPanel2="Policy_01";

// INFO PANEL
InfoPanelAdmins=GetByFilter("ADMIN","1=1",null);
InfoPanelAdmins_style="table";
```

In this example, you use the action panel in the operator view to trigger three policies, named Policy_01, Policy_02 and Policy_03. It also contains an information group named InfoPanelAdmins that displays the data items that are returned from the ADMIN data type by a call to the GetByFilter function.

Advanced operator view policies: Advanced operator view policies consist of a set of statements that assign values to variables in the policy context. The data in these values is inserted into the operator view when the GUI Server filters the corresponding display page.

When you create an advanced operator view, you must manually create the operator view policy as a separate step using the GUI or a text editor. For more information about creating operator view policies, see "Creating the operator view policy" on page 14.

The following example shows a policy that works with an advanced operator view:

```
// Retrieve summary information about the node where an incoming event
// was reported, where @Node is an event field passed to the operator view
// using the URL syntax

DataType = "Host";
Filter = "Hostname = '" + @Node + "'";
CountOnly = False;

Hosts = GetByFilter(DataType, Filter, CountOnly);

// Retrieve geographical information about the node from the
// Contacts data type
```

```
DataType = "Admins";
Filter = "Facility = '" + Nodes[0].Facility + "'";
CountOnly = False;

Contacts = GetByFilter(DataType, Filter, CountOnly);
```

In this example, the policy retrieves information about the node from the `Host` data type and stores this information in a variable named `Hosts`. Then, it retrieves contact information for the administrator who is responsible for managing systems in the facility where the node is located and stores this information in the `Contacts` data item array. When the GUI Server filters the display page for the view, it can insert the value of any of these variables into the resulting operator view Web page.

Display pages:

A display page is a text file that contains HTML content and special instructions called smart tags. Smart tags determine what data to display in the operator view and how to display it.

Display pages are named *clustername-viewname*`.html`, where *clustername* is the name of the server cluster and *viewname* is the name of the operator view. These pages are located in the `$IMPACT_HOME/opview/displays` directory. There is one display page for each operator view.

If you create a new cluster and want the operator views to be available for that cluster, choose one of the following options:

- Copy the appropriate opview`.html` files in to the new cluster name
- Use the **`installAddOnOpview.xml`** command with the following format

  ```
  nc_ant -f installAddOnOpview.xml NewCluster -DCLUSTERNAME=<NewClusterName>
  ```

  Remember: This command overwrites any html files that are already there. Do not execute this command if you do not want to copy all the existing operator view from **`NCICLUSTER`** to the new cluster.

Display pages are similar to JSP pages or other types of HTML templates that are interpreted in real time by a Web server, to insert dynamic data obtained from a database or other data source. The HTML content in a display page is returned to the user's Web browser unaltered. Smart tags in the display page are filtered and evaluated by the GUI Server at runtime. For more information about smart tags, see Chapter 3, “Working with smart tags,” on page 27.

The content of a display page depends on whether it is associated with a basic operator view or an advanced operator view.

Basic display pages:

A basic display page contains sections of HTML markup that specify how the Web browser renders the event panel, action panel, and information groups panel that is displayed in the operator view. The page also contains embedded smart tags that specify the data that is displayed in the view and how it is displayed.

When you create a basic operator view using the GUI, the GUI automatically creates a basic display page.

You can manually modify the HTML tags and smart tags in a basic display page using a text editor after it has been created. After you modify the page, however, you cannot alter the operator view using the GUI. You must perform any additional configuration of the operator view using the text editor.

Advanced display pages:

An advanced display page contains HTML markup that freely specifies how the Web browser arranges the display elements in the operator view.

Unlike basic display pages, an advanced page can contain any type of HTML content that can be displayed by a Web browser. Like basic display pages, the page also contains embedded smart tags that specify which data is displayed in the view and how it is displayed. You must manually create advanced display pages using a text editor. For more information, see "Creating display pages manually" on page 7.

The following example shows a simple page that can display the data that is retrieved by the policy in "Advanced operator view policies" on page 3.

```
<html>
<head>
<link rel="stylesheet" type="text/css" href="my.css" />
<script src="/netcool/scripts/prototype.js" type="text/javascript"></script>
<script src="/netcool/scripts/scriptaculous.js" type="text/javascript"></script>
<script src="/netcool/scripts/script.js" type="text/javascript"></script>
<script src="/netcool/scripts/opview.js" type="text/javascript"></script>
<title>Operator View: <!--property:policy="EX_01" --></title>
<!-- <!--property:DefaultClusterName="NCICLUSTER" --> -->
</head>
<body>
<h1>MY INTRANET</h2>
<p>Example Operator View</p>
<h1>Event Summary</h1>
<p>Information passed to view from event source using URL syntax:</p>
<table>
<tr>
<td>Node</td>
<td>Severity</td>
<td>Summary</td>
<td>Count</td>
</tr>
<tr>
<td>
<!--showdata:
    var="Node"
    type="scalar"
    format="string"
    id="node-element"
    class="node-class"
    style="color: #3f3f3f"
    title="Hostname or IP Address of Node"
    default="Hostname or IP Address Unknown"
    cacheread="true"
    cachewrite="true"
    autourl="false"
-->
</td>
<td>
<!--showdata:
    var="Severity"
    type="scalar"
    format="string"
    id="severity-element"
```

```
    class="severity-class"
    style="color: #3f3f3f"
    title="Severity of Event"
    default="Severity Unknown"
    cacheread="true"
    cachewrite="true"
    autourl="false"
-->
</td>
<td>
<!--showdata:
    var="Summary"
    type="scalar"
    format="string"
    id="summary-element"
    style="color: #3f3f3f"
    title="Summary of Event"
    default="Summary Unknown"
    cacheread="true"
    cachewrite="true"
    autourl="false"
-->
</td>
<td>
<!--showdata:
    var="Count"
    type="scalar"
    format="string"
    id="count-element"
    class="count-class"
    style="color: #3f3f3f"
    title="Event Count"
    default="Count Unknown"
    cacheread="true"
    cachewrite="true"
    autourl="false"
-->
</td>
</tr>
</table>
<h1>Node Summary</h1>
<p>Information about the node where the event occurred.</p>
<p>Retrieved from Netcool/Impact data source using a GetByFilter
query in the operator view policy.</p>
<!--showdata:
    var="Nodes"
    type="orgnodes"
    format="customtable"
    headerclass="head"
    action_count="0"
-->
<p>In per item format.</p>
<!-- test_showdata:
    var="Nodes"
    type="orgnodes"
    format="peritem"
--->
<h1>Contact Summary</h1>
<p>Contact information for the administrator currently on-call.</p>
<p>Retrieved from an LDAP data source using a GetByFilter query
in the operator view policy.</p>
<!--showdata:
    var="Contacts"
    type="orgnodes"
    format="customtable"
    headerclass="head"
```

```
    action_count="0"
-->
</body>
</html>
```

Using AJAX in advanced display pages:

AJAX capabilities enhance operator view display pages. You use them to refresh individual elements on a display page on a tag by tag basis.

Procedure

To enable AJAX in the operator view, add new attributes to the smart tags in your HTML display page.
The following smart tag attributes are used to enable AJAX in the operator view, and they apply to the Scalar, List, and OrgNodes tags:

- update_interval
- update_option
- update_delay
- update_label
- update_policy
- update_tags (and *_override_tags)
- update_params
- update_precall
- update_postcall
- update_effect

Smart tags with these attributes refresh content automatically at specified intervals, or they can be refreshed manually.

What to do next

For more information about each new smart tag attribute, see the following sections:

- “Scalar tag” on page 39
- “List tag” on page 41
- “OrgNodes tag” on page 44

Creating display pages manually:

If you create an operator view policy outside of the Operator View editor, you must manually upload its display page to the Impact Server.

About this task

The name of the manually created policy should be prefixed with “Opview_”. Policy names are case-sensitive.

Procedure

1. On the Impact Server navigate to the `$IMPACT_HOME/opview/displays` directory. If the GUI Server is configured in a failover cluster environment, then you need to copy the `.html` files from one GUI Server server to another.
2. Upload the display page.

Make sure the display page file name follows the naming convention, as specified in "Display pages" on page 4.

3. Open or refresh the Operator View UI.

   Select the required operator view from the list and open it for editing. Verify that the operator view editor contains your operator view.

Setting up an operator view

To set up a basic operator view, you use the GUI to specify a page layout and select the data to display in the view. The GUI automatically creates the display page and the operator view policy.

To set up an advanced operator view, you use a text editor or the GUI to create the operator view policy, and then you use the text editor to create the display page that is associated with the view.

For more information about setting operator views, see "Working with basic operator views" on page 9 and "Working with advanced operator views" on page 14.

Managing an operator view

You can use the GUI to view, modify, and delete existing basic operator views and the policies associated with any view type. You use a text editor and other system utilities to view, modify, and delete the display page for an advanced operator view.

For more information about managing operator views, see "Working with basic operator views" on page 9 and "Working with advanced operator views" on page 14.

Operator view process

When a user opens an operator view, the Web browser sends an HTTP request to the GUI Server for the data that is located at the specified URL. The GUI Server then performs the following actions:

- Parses the incoming HTTP request to identify the operator view and to obtain any event information that is passed using the URL.
- Opens the display page associated with the operator view.
- Runs the policy that is associated with the operator view and passes the policy any incoming event information contained in the URL.
- Filters the display page, interprets any smart tags that it contains, and then replaces the tags with the data retrieved by the operator view policy
- Returns the resulting HTML output to the requesting browser.

The Web browser then displays the operator view.

Chapter 2. Working with operator views

Using a combination of the GUI, operating system utilities, and an external text editor you create new operator views, and view, modify, and delete existing operator views. After you create an operator view, you can customize it by manually editing the operator view policy and display page.

Working with basic operator views

You create basic operator views with the GUI.

Operator views have the following configuration properties:

- Operator view name
- Layout options
- Action panel policies
- Information groups

Each basic operator view has a corresponding operator view policy. This policy is named `Opview_`*`viewname`*, where *`viewname`* is the name of the operator view. By default, this policy is in the global repository but is not the member of any project, including the one currently selected in the GUI.

Operator view name

An operator view name is a unique name used to identify the view.

The GUI Server also uses this name as part of the operator view URL that you use when you open the view in a web browser.

Layout options

When you create a basic operator view using the GUI, you can use the layout options and the associated preview feature to specify how different parts of the tool are arranged on the resulting web page.

The following table shows the display panels in a basic operator view:

Table 1. Operator view display panels

Display Panel	Description
Event panel	Displays information, if any, passed from Netcool/OMNIbus or another application to the operator view. This information can be fields in a Netcool/OMNIbus event, or any other information that can be expressed as a set of name/value pairs. You can configure the layout so that the event panel is displayed on the top or the bottom of the operator view, or not at all.
Action panel	Contains a list of policies associated with this view. You can configure the layout so that the action panel is displayed on the top, the bottom, the left or the right of the display, or not at all.
Information group panel	Displays sets of information retrieved from data types. This data is often business data that is related to event information passed to the view from Netcool/OMNIbus or another application.

Action panel policies

You can use the action panel editor in the GUI to specify one or more policies that are displayed in the action panel of a basic operator view.

The action panel presents a list of policies that the user can start from within the view. This is an optional part of the operator view function. You use the action panel to start policies only, you cannot use it to display data that is returned by a policy. An advanced operator view, however, does provide the capability to display this data.

Information groups

An information group is a set of dynamic data that is displayed when you open the view.

This is often business data that is related to event information that is passed to the view from Netcool/OMNIbus or another application. The data that is displayed in an information group is obtained by a query to a data source either by filter or by key.

When you create a basic operator view using the GUI, you can specify one or more information groups that are to be displayed by the view.

The following table shows the properties that you specify when you create an information group:

Table 2. Information group configuration properties

Property	Description
Group name	Unique name that identifies the information group.
Type	Type of query to a data source. Available options are: • By Key: Key expression that specifies which data to retrieve from the data type. • The filter syntax is similar to the contents of the `WHERE` clause in an SQL select statement. • By Filter: SQL filter string that specifies which data to retrieve from the data type.
Data type	Data type that contains the data that you want to display.
Value	Add a value.
Style	Layout style for data items in the resulting information group. Options are `Tabbed` and `Table`.

You can customize the information that is displayed in the information groups by editing the operator view policy.

Creating a basic operator view

Complete the following steps to create basic operator views.

About this task

Procedure

1. Log on to the GUI.

2. In the navigation tree, expand **System Configuration** > **Event Automation** click **Operator Views** to open the **Operator Views** tab.
3. From the **Cluster** list, select the cluster you want to use.
4. From the **Project** list, select the project you want to use.
5. Click the **New Operator View** icon to open the **New Operator View**.
6. In the **Operator View Name** field, enter a unique name for the operator view. You cannot edit the name once the operator view is saved.
7. In the **Layout Options** area, specify the position of the event panel and action panel in the operator view. You can preview the appearance of the operator view using the images available in the **Preview** area.
8. Click the **Action Panel** link, select one or more action policies that the user can open from within the operator view.
9. Click the **Information Groups** link. Use the following steps to create one or more information groups:
   a. Click the **New Information Group** icon to insert a new row into the information groups table.
   b. In the **Group Name** field, type a unique name for the group.
   c. From the **Type** list, select **By Filter** or **By Key** to specify whether the information group retrieves data from a data type by filter or by key.
   d. From the **Data Type** list, select the data type that contains the information you want to display.
   e. In the **Value** field, enter a filter or key expression. If the **Type** is **By Filter** adding a value is optional. If the **Type** is **By Key** then the value is mandatory.
   f. In the **Style** list, select **Tabbed** or **Table** to specify how the operator view shows the resulting data.
   g. Press `Enter` on your keyboard to confirm the value you are adding to the information group (or press `Escape` on your keyboard to cancel the edit).
   h. Repeat these steps to create multiple information groups for any operator view.
   i. To edit an information group, click the item you want to edit and change the value.
   j. To delete one or more information groups, multiselect the rows groups using the Ctrl and shift keys on the keyboard, then click **Delete**.
      - To sort rows up or down, select a row or multiple rows to activate the **Move Up** and **Move Down** arrows on the toolbar. Click the required icon to move the rows up or down by one row.
10. Click the **Save** icon on the main editor toolbar to implement the changes.

Manually editing basic operator view components

After you use the GUI to create a basic operator view, you customize the view by manually editing the operator view policy and display page.

For more information about editing the operator view policy and display page, see “Editing the operator view policy” on page 12 and “Editing the operator view display page” on page 12.

Editing the operator view policy
About this task

Basic operator view policies are named `Opview_viewname`, where *viewname* is the name of the operator view. By default, these policies are not part of any project.

Procedure

1. In the navigation tree, expand **System Configuration** > **Event Automation**. click **Policies** to open the **Policies** tab.
2. From the **Projects** list, select **Global**, browse the policy list.
3. To edit an operator view policy, right click on the policy and click **Edit**, or double click on the policy.
4. Change the filter or key expression associated with an information group so that it contains event variables.

   The most common change to a basic operator view policy is to change the filter or key expression associated with an information group so that it contains event variables. You can select data that appears in the group based on event information passed to the view from Netcool/OMNIbus or another application.

   You reference these event variables in the filter using `EventContainer.field` , where *field* is the name of the variable.

Example

The `GetByFilter` statement contains a filter that retrieves data items from a `Node` data type where the value of the `Hostname` field matches the value of the `Node` event field passed to the operator view:

```
InfoPanelNodeGroup=GetByFilter
("Node", "Hostname='" + EventContainer.Node + "'", false);
```

The `GetByKey` statement contains a key expression that retrieves data items from a Node data type where the value of the key field matches the value of the `Node` event field passed to the operator view:

```
InfoPanelNodeGroup=GetByKey("Node", EventContainer.Node, Null);
```

Editing the operator view display page

Basic operator view display pages are named *clustername-viewname*`.html`, where *clustername* is the name of the Tivoli Netcool/Impact server cluster and *viewname* is the name of the operator view. These pages are located in the `$IMPACT_HOME/opview/displays` directory.

You can make relatively unrestricted changes to the HTML content and smart tags in the display page for a basic operator view. However, you must make sure that changes you make to smart tags in the display page are also reflected in the operator view policy. In addition, after you modify the display page, you can no longer use the GUI to edit the configuration properties for the view.

For information about operator view smart tags, see Chapter 3, “Working with smart tags,” on page 27.

Viewing operator views

To view the basic and advanced operator views that are currently defined in IBMTivoliNetcool/Impact, log on to the GUI.

Procedure

1. Log on to the GUI
2. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Operator Views** to open the **Operator Views** tab.
3. From the **Cluster** list, select the cluster that you want to use.
4. From the **Project** list, select the project that you want to use.
5. Double-click the operator view to see the details or right click the operator view and click **Edit**.

Editing operator views

You can edit operator view properties.

About this task

You can use either the GUI or an external text editor to modify the policy that is associated with an advanced operator view. If you modify the policy using an external text editor, you must import the policy manually after you make your changes. You are not required to stop and restart the Impact Server or GUI Server after modifying an existing operator view policy. Any changes that you make to the policy are immediately recognized by the system.

You can only use an external text editor to modify the display page that is associated with an advanced operator view only. Do not attempt to use the tools provided in the GUI for use with basic operator views to modify the display page. If you use the GUI to modify the display page, the changes that you make override the HTML content and smart tags in the existing HTML file.

Procedure

1. To modify a basic operator view, log on to the GUI.
2. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Operator Views** to open the **Operator Views** tab.
3. From the **Cluster** list, select the cluster that you want to use.
4. From the **Project** list, select the project that you want to use.
5. Double-click the operator view you want to modify, or click the **Edit Operator View** icon.
6. Modify the operator view configuration properties as required. You cannot modify the **Operator View Name**.
7. Click the **Save** to implement the changes.

Deleting operator views

To delete a basic or advanced operator view log on to the GUI.

Procedure

1. Log on to the GUI.
2. In the navigation tree, expand **System Configuration** > **Event Automation**, click **Operator Views** to open the **Operator Views** tab.
3. From the **Cluster** list, select the cluster that you want to use.
4. From the **Project** list, select the project that you want to use.
5. Select the operator view that you want to delete and click the **Delete** icon on the toolbar, or right click the operator view and click **Delete**.

6. A confirmation message displays.
7. Click **OK** to delete the operator view.

   The operator view is removed from the GUI display and the operator view display file and policy are removed from the system.

Working with advanced operator views

To set up an advanced operator view, you do the following processes:

- Create an operator view policy
- Create a display page

For more information about creating operator view policies and creating display pages, see "Creating the operator view policy" and "Creating the display page" on page 15.

Creating the operator view policy

The operator view policy is a policy that contains the logic that is required to retrieve and manipulate the data that is displayed in the view.

This policy must be named `Opview_`*`viewname`*, where *`viewname`* is the name of the operator view. There is one policy for each operator view. You can create an operator view policy in one of the following ways:

- You can use the GUI to create the operator view policy.
- You can use an external text editor to create the policy. After you create it, you import the policy.

Tip: You can also create the basic operator view, and then modify the content of the operator view policy. Delete the content from the operator view `.html`files. The content between the `<body> </body>` can be deleted and updated with your own content.

An operator view policy performs the following tasks:

- Handles incoming events.
- Queries data sources for data.
- Manipulates and normalizes the data as necessary so that the format is suitable for display in the view.
- Overrides smart tag attributes specified in the display page.

Handling incoming events

The operator view provides a special syntax that you use to pass event data to the view using query string values in its URL.

When the GUI Server receives an HTTP request for an operator view, it passes any event values that are contained in the URL to the operator view policy for processing.

Each event field that is passed to the policy is stored as a variable in the policy context before processing begins. The variable names for the event fields are exactly as specified in the URL. For example, if you pass a summary value in the URL using the `Summary=Node+not+responding+to+ping` string, you can access this value in the policy through the EventContainer using `EventContainer.Summary` or

`@Summary` for short. If you set a summary variable within the policy to a value, for example, `Summary="Node not responding to ping"`, it can be accessed through a smart tag, for example:

```
showdata: var="Summary"
             type="scalar"
             format="string"
```

If you want to display an event field value exactly as it was passed to the operator view, you do not need to perform any operations on the value in the policy.

Querying data sources

An operator view typically queries one or more data sources for information to display.

Often, this information is correlated with incoming event field data that is passed to the view using the URL syntax.

You can query the data sources using the `GetByFilter`, `GetByKey` or `DirectSQL` functions, or using any other function that retrieves data as a scalar variable or array of data items.

If you are retrieving data using a function that returns an array of data items, you must explicitly set the return variable for the function. You reference this return variable when you create the display page.

Manipulating or normalizing data

You must perform any manipulation or normalization of data within the policy before it is displayed in the operator view.

For example, if you need to extract strings from data, trim white space from strings or perform calculations on numeric values, you must perform these operations in the operator view policy before they are displayed in the view.

Overriding smart tag attributes

You can also optionally override the value of smart tag attributes in a policy.

These attributes are specified as part of the smart tag definitions that you insert into a display page. Overriding smart tag attributes in a policy you dynamically control some aspects of how an operator view displays data. For more information about overriding attributes, see “Overriding attributes” on page 29.

Creating the display page

A display page is a text file that contains HTML content and special instructions called smart tags that are used to determine which data to display in the operator view and how to display it.

Display pages are named *clustername*`-`*viewname*`.html`, where *clustername* is the name of the IBM Tivoli Netcool/Impact server cluster and *viewname* is the name of the operator view. These pages are located in the `$IMPACT_HOME/opview/displays` directory. There is one display page for each operator view.

The HTML content in a display page is identical to the HTML code that is in any other Web page. The HTML content specifies the static content and formatting for the resulting operator view, and any additional metadata that is required to present the content. You can use HTML syntax to arrange and format the content in the display page in the same way you create or design any other Web page.

For advanced operator views, you create the display page using an external text editor. One approach for creating operator views is to design the Web page and enter the HTML content first using mock data that is similar to the data that you expect to display in the operator view. Then, you can insert the smart tags that present the dynamic data that is obtained by the operator view policy.

The smart tags in the display page do the following things:

- Identify the cluster where the operator view is running.
- Identify the operator view policy.
- Specify which data to display.
- Specify how the data is displayed.

Identifying the server cluster

Every operator view display page must contain one instance of the property smart tag that specifies the name of the server cluster where the view is running.

Typically, this property tag is located in the `<head>` element of the HTML page, inside a double set of HTML comment tags that ensure that the tag is not displayed by the Web browser and can be read when you view the page source.

The following example shows how you identify the server cluster in a display page:

```
<!-- <!--property:DefaultClusterName="NCICLUSTER" --> -->
```

Identifying the operator view policy

Every operator view display page must also contain one instance of the property smart tag that specifies the name of the policy that is associated with the operator view.

As with the previous property tag, this tag must come before any tag that inserts data in the page. This property tag is also typically located in the `<head>` element of the HTML page, inside a double set of HTML comment tags.

The following example shows how you identify the operator view policy in a display page:

```
<!-- <!--property:policy="EX_01" --> -->
```

For more information about the property tag, see "Property tag" on page 35.

Specifying which data to display

Advanced display pages can use any or all special smart tags that specify which data obtained by the operator view policy to display in the view.

You insert these tags in the `<body>` element of the display page at the location in the HTML content where you want the data to be displayed on the resulting Web page.

The following smart tags display data:

Scalar tag
: You use the scalar tag to display single values, such as a string or a numeric value (in string format only).

List tag
: You use the list tag to display a list of scalar values.

OrgNodes tag

You use the orgnodes tag to display an array of data items that are retrieved from a data source using a function like `GetByFilter` or `GetByKey`.

For more information about the scalar tag, list tag, and orgnodes tag, see "Scalar tag" on page 39, "List tag" on page 41, and "OrgNodes tag" on page 44.

The following example shows how you display an array of data items that are stored by the operator view policy in a variable named `Admins`:

```
<p>Display the administrators who are on call in a table:</p>
<!--showdata:
    var="Admins"
    type="orgnodes"
    format="customtable"
-->
```

Specifying how data is displayed

The scalar, list, and orgnodes tags are used to specify how the data obtained by the operator view policy is displayed in the view.

You use these tags to format the data as plain text, links, buttons, or actions (which cause the view to open another operator view). Attributes that are supported by the tags are used to control the appearance of data that is displayed in table format. They are also used to associate CSS styles with the data and to associate the data with HTML DOM IDs and classes. You can then use these elements to format the operator view Web page using DHTML or CSS.

The following example shows how you use the attributes of an orgnode tag to control how data is displayed in a table:

```
<p>Display the administrators who are on call in a table:</p>
<!--showdata:
    var="Admins"
    type="orgnodes"
    format="customtable"
    id="admin-table"
    class="formatted-table"
    headerclass="head"
    cellclass="formatted-table-class"
    cellstyle="background-color: #3f3f3f;"
    action_count="0"
-->
```

Customizing operator view displays index page

The operator view displays index page is the page where you can view and access all your defined views.

You access the operator view displays index page in a web browser. For more information about accessing the operator view displays index page, see "Creating the custom operator view page" on page 22.

You can customize the appearance and behavior of the operator view displays index page in one of the following ways:

- .css definitions
- .meta files
- index URL (cluster, stylesheet)

For more information about customizing the operator view displays index page, see “Customizing the index page using CSS definitions,” “Customizing the index page using .meta files,” and “Customizing the index page using index URL” on page 20

Customizing the index page using CSS definitions

Every section and subsection of the index page is wrapped either in a `div` or `span` tag and you can customize it through the style definitions in the `$IMPACT_HOME/opview/assets/installed/opview_index.css` stylesheet.

About this task

To customize the index page through the style definitions:

Procedure

Open the `opview_index.css` stylesheet and modify any of the following CSS elements:

- `.logout_link`
- `#login_label`
- `#login_user`
- `.superheader`
- `.header`
- `#tab_list`
- `.tab_entry`
- `.display_list`
- `#list_[CLUSTERNAME]`
- `#entry_[CLUSTERNAME]-[DISPLAYNAME]`
- `.display_entry`
- `.display_icon`
- `.display_title`
- `.display_description`
- `.display_params`
- `.parameter`
- `.parameter_[PARAMETER_NAME]`
- `.display_launch .display_lastupdate_section`
- `.display_lastupdate_label`
- `.display_lastupdate_value`

Customizing the index page using .meta files

You can customize the properties of the displays listed on the operator view index page, using additional `.meta` files in the `$IMPACT_HOME/opview/displays` directory. Each display is individually customized with its own `.meta` file.

About this task

To customize the index page through a `.meta` file:

Procedure

1. Create a new text file and fill it in with the parameters that you want to customize in your display. For more information about the parameters that you can customize in a `.meta` file, see "Properties used in .meta files."
2. Save the file to the `$IMPACT_HOME/opview/displays` directory with the same name as the display file but postfixed with a `.meta` extension. For example, for a `NCICLUSTER-ReprocessFailedEvent.html` display page you have to create a meta file with the name `NCICLUSTER-ReprocessFailedEvent.html.meta` file.

Example

Example of a meta file with customized parameters:

```
title= Fancy display
description= Really nice!

parameters=cost,profit
target-window=_new

last-update=In the year 3000
graphic=/opview/assets/installed/pretty_picture.gif
hide-fields=
authorized-roles=OPVIEW_USER
```

Properties used in .meta files

The following table provides details about the `.meta` file properties that you can use to change the look and behavior of a display in the operator view displays index page.

Table 3. List of properties used in .meta files

Property	Description
`title`	Use this property to specify an alternate name for the operator view display.
`description`	Use this property to provide descriptive details for the operator view.
`last-update`	By default, the last modified timestamp of the operator view display filename is listed as the **Last Update** on the index page. Use this property to override that value.

`graphic`	With this property you specify the URL to a logo or an application icon that you would like to display alongside your operator view entry. The path must be either a full URL, for example
	`http://www.google.com/images?q=tbn:I1KyULEyeN1Z6M: : i192.photobucket.com/albums/z48/adtracker/noid.gif`
	or, if the file is local to your file system, you must put it in the `$IMPACT_HOME/guiserver/webapps/opview/assets/installed` directory and provide a path that is relative to the `/opview` directory. For example:
	`graphic=/opview/assets/installed/my_picture.gif`

`hide-fields`	If there is any information you want to hide on the index page for a specific entry, then you can assign a comma-delimited list of the fields to hide in this property. For example, if you want to hide the description and last-update information for a specific entry set the property to: `hide-fields=description,last-update`

Table 3. List of properties used in .meta files (continued)

Property	Description
`parameters`	The index page provides event context. The parameters property configures the entry on the index page with parameter inputs that you can provide before running the operator view display.
`target-window`	Use this parameter to specify an alternate window to open the operator view display into. Specifying `target-window=_new` would run the operator view display into the window specified. If you omit out this parameter in the `.meta` file your display will open in the same window as the operator view display index.
`authorized-role`	You can prevent certain operator view displays from being listed on the index page. By assigning a role or roles to the authorized-roles property, you are effectively requiring the currently logged-in user to have the proper credentials to view the display. If they are not authorized the entry will not show up on the index page. If more than one role is used as the value of the parameter separate them with a comma (,). For example: `authorized-roles=IMPACT_USER,OPVIEW_USER` For more information about the roles that can be assigned to users, see the Administration Guide, *Configuring the GUI server*.

Customizing the index page using index URL

You can customize the index page through the following parameters that can be passed in the URL query string:

- `cluster` - you can load up the index page with just the available displays for a single cluster or you can specify an ordered list of clusters for the index page.
- `stylesheet` - an alternate stylesheet can be swapped in so the appearance of the operator view can be customized and displayed to various users or user types.

Passing a cluster with the index page

Pass the cluster parameter in the operator view display URL to load up the index page with just the available displays for a single cluster or to specify an ordered list of clusters for the index page.

Procedure

- Displaying a single cluster.

  To load up the index page with just the available displays for a single cluster append the cluster name to the index page using the `?cluster=<cluster_name>` syntax. For example:

  `URL: http://<hostname>:16310/opview/displays/index?cluster=NCICLUSTER`

  Note: If there is only one cluster in the configuration, the cluster tabs are not displayed.
- Reordering cluster tabs.

  To specify an ordered list of clusters for the index page, append a comma delimited list of clusters to the index file using the `?cluster=<cluster1_name>,<cluster2_name>` syntax. For example:

  `URL: http://<hostname>:16310/opview/displays/index?cluster=NCICLUSTER,NCI1CLUSTER`

  Note: If there is only one cluster in the configuration, the cluster tabs are not displayed.

Passing an alternate stylesheet with the index page

You can swap in an alternate stylesheet to customize the appearance of the operator view to suit various users or user types.

Procedure

To swap in an alternate stylesheet, append the new stylesheet name to the index page using the following syntax:

```
http://<hostname>:<port>/opview/displays/index?stylesheet=<aternate_stylesheet>
```

Example

An example of using an alternate stylesheet in the operator view index page URL:

```
http://<hostname>:16310/opview/displays/index?stylesheet=fancy
```

In this example the default stylesheet, `opview_index.css`, is replaced by the alternate stylesheet, `fancy.css`. Place the alternate stylesheet under the assets path, `$IMPACT_HOME/opview/assets/installed`.

Viewing an operator view page in the Tivoli Integrated Portal

You can use the portlet and page features in the Tivoli Integrated Portal to view an operator view URL.

Complete the following steps to set up a custom operator view page in the Tivoli Integrated Portal.

- Select the operator view browser URL you want to display.
- Create a custom operator view portlet and add the URL to the portlet.
- Create the custom operator view page and add the portlet page to view the operator view URL.

Selecting the operator view URL

Select the operator view that you want to display in the Tivoli Integrated Portal and copy the URL from the browser URL field.

Before you begin

Make sure your Tivoli Integrated Portal user rights give you permissions to create pages.

Procedure

1. Log on to the Tivoli Integrated Portal.
2. In the Tivoli Integrated Portal navigation pane, click the **System Configuration -> Event Automation -> Operator Views** node.
3. Select the operator view, then right-click and select **View** to open the operator view in a new window.
4. In the new window, copy the URL from the browser URL field.

Creating a custom operator view portlet

Create a custom operator view portlet in the Tivoli Integrated Portal and add the operator view URL to the portlet.

Before you begin

Make sure that you copy the operator view URL from the URL field in the browser.

Procedure

1. In the Tivoli Integrated Portal navigation pane, click **Settings** > **Portlets** node.
2. Click the **New** icon in the toolbar to open the Portlet creation wizard.
3. Click **Next** to start the wizard.
4. For the **Base Portlet**, select **Web Widget**. Click **Next** .
5. Complete the portlet **Name** and **Description** fields. Click **Next**.
6. You can assign the level of privileges for any of the available roles by selecting the role and adding it to the relevant list, **User, Privileged User, Editor, Manager**.
7. Click **Next**.
8. Complete the following fields:

   Widget Title:
   Add the name you want to use.

   Home Page:
   Paste in the URL of the operator view, for example, `https://<hostname>:<port>/opview/displays/NCICLUSTER-myoperatorview.html`

   Help Page:
   You can paste in the URL of the help page if one is available. Otherwise, leave the field blank.

   HTML iframe name:
   Leave this field blank.

   Show a browser control toolbar
   Clear this check box.
9. Review the summary, then click **Finish** to save the portlet.

Creating the custom operator view page

Create a custom operator view page to display the operator view portlet that shows the operator view URL page.

Before you begin

Make sure your Tivoli Integrated Portal user rights give you permissions to create pages.

Procedure

1. In the Tivoli Integrated Portal navigation pane, click the **Settings** > **Pages** node.
2. Click **New Page** in the **Pages** window.
3. In the **Page Settings** window, provide the following information:

   Page name
   The name of the new page.

   Page location
   The position of the new page in the navigation pane.

Page layout
Choose the “Classic”, or the “Freeform” layout for the new page. Optional

Click **OK** to continue.

4. In the **Choose a Portlet** window, select the portlet you created for the operator view and add it to the page.
5. Click **OK**.
6. Click **Save** in the upper right corner of the page. The operator view URL is displayed.

Sending data from a charting table to an operator view

You can send data from a Tivoli Integrated Portal charting table to an operator view in Netcool/Impact.

About this task

After you complete this task, you can use the node click events feature of the Tivoli Integrated Portal to send data from a charting table to an operator view in Netcool/Impact. The information is passed through the URL to the receiving operator view. The information is made available through the `EventContainer` in the operator view policy.

Procedure

1. Create a Tivoli Integrated Portal page.

   To create a Tivoli Integrated Portal page, click the **Settings** > **Pages** node in the Tivoli Integrated Portal navigation pane. Click **New Page** in the **Pages** window.
2. Create two portlets. The first portlet represents the operator view and acts as a container. The second portlet uses the Tivoli Integrated Portal charting table as a base portlet and it retrieves information from the specified data type in Netcool/Impact.

   To create a portlet, click **Settings** > **Portlets** in the Tivoli Integrated Portal navigation pane. Click the **New** icon in the toolbar to open the Portlet creation wizard and click **Next** to start the wizard.
3. Edit the portlet preferences. To add the receiving operator view to the portlet that represents the operator view, edit the portlet preferences. Enter the name of the operator view that you want to send information to in the **Operator View Name** field.
4. To send data from the chart to the operator view, right click on a row in a table in Tivoli Integrated Portal and click **Emit Context**. The results are returned in the operator view policy.

Results

You can now use the emit context feature to send data from the portlet in Tivoli Integrated Portal to the operator view in Netcool/Impact.

The results are returned in the operator view policy in the `Event Container`. To display the results in the JavaScript Object Notation (JSON) format, you must decode the results that are contained in the `Event Container`. For example:

```
eventVar = @Event
eventVar = Replace(eventVar, "&quot;", '"', 100);
```

Example

The following example demonstrates how to send data from a charting table in Tivoli Integrated Portal to an operator view that is based on a flat file policy.

This example uses the following flat file policy that contains a resource name and ID:

```
resourcename, id
resource1, 1
resource2, 2
```

1. Create a flat file data source that uses the data from an existing file.
2. Create a flat file data type. You must ensure that the **Access the data through UI data provider** check box is selected and that the key field is selected. For example, you can select ID as the key field for the example policy that is defined in the previous step.
3. Create a portlet that uses the Tivoli Integrated Portal charting portlet as the base portlet. To create a portlet, in the Tivoli Integrated Portal navigation pane, click **Settings** > **Portlets**. Click the **New** icon in the toolbar to open the Portlet creation wizard.
   a. Click **Next** to start the wizard.
   b. Select Charting base portlet and click **Next**.
   c. Select a Provider a name and description and click **Next**.
   d. Choose your security and click **Next**.
   e. Select Create a chart and click **Next**.
   f. Select the flat file data source that you created as the data set.
   g. Select the flat file data type that you created from the data set.
   h. Set the visualization settings as a Tivoli Integrated Portal table that selects all columns.
4. Create an operator view if required. You can use the default operator view that is created by the editor. This step is optional because you can also use an existing operator view.
5. Add the following statements to the operator view that you are using. This example assumes that you are using the Impact Policy Language (IPL) for your operator view policy:

   ```
   log(CurrentContext());
   eventVar = @Event;
   eventVar = Replace(eventVar, "&quot;", '"', 100);
   log("Passed JSON values: "+eventVar);
   ```

6. Create a Tivoli Integrated Portal page that contains two portlets. The first portlet is the charting portlet that you created previously. The second represents the operator view. To create a Tivoli Integrated Portal page, click the **Settings** > **Pages** node. Click **New Page** in the **Pages** window in the Tivoli Integrated Portal navigation pane.
7. Add the operator view policy from Netcool/Impact to the operator view portlet. To add the policy, edit the preferences for the operator view portlet and enter the name of the operator view in the **Operator View Name** field. Save your changes. For example, if the operator view policy is called **Opview_test**, you enter `test` in the **Operator View Name** field.
8. After both portlets load, right click a row in the table and click **Emit context**. The selected row of data is copied to the event area of the operator view.
9. Open the log file for the operator view policy **Opview_test**. The details of the passed data are displayed as follows:

```
Parser log: Passed JSON values: {"resourcename":"resource1","id":"1"}
```

Chapter 3. Working with smart tags

Smart tags are text elements in a display page that contain special instructions that are used to identify the Impact Server cluster, specify the policy associated with the operator view and determine which data to display in the view and how to display it.

Smart tags overview

You enclose smart tags in HTML comments and embed them inside the HTML content that makes up the display page.

The following example shows simple smart tags as they are displayed in a display page.

```
<html>
<head>
<title>My Operator View</title>
<!-- <!--property:policy="MyView" --> -->
<!-- <!--property:DefaultClusterName="NCICLUSTER_02" --> -->
</head>
<body>
<h1>OrgNodes Smart Tag Example</h1>
<!--showdata:
    var="Nodes"
    type="orgnodes"
    format="customtable"
-->
</body>
</html>
```

The smart tags in this example specify that the name of the server cluster is `NCICLUSTER_02` and that the name of the policy that is associated with the operator view is `Opview_MyView`. The tags also display the contents of a variable named `Nodes` that contains a set of data items that are retrieved by the operator view policy.

Every display page must contain at least two property tags: one tag that specifies the name of the server cluster and another tag that specifies the operator view policy. For more information about the property tag, see “Property tag” on page 35.

Smart tag syntax

Smart tags us a specific syntax.

Smart tags have the following syntax:

```
<!--tagtype:
    attribute=value
    attribute=value
    attribute=value
    .
    .
    .
-->
```

Where *tagtype* is either `property` or `showdata`, and *attribute* and *value* are name/value pairs that specify the parameters for the tag. You enclose smart tags inside HTML comments in the display page. You can place attributes in the smart tag in any order.

Attribute values in a smart tag must be specified as text strings enclosed with double quotation marks ("). For example, the following attribute assignments are valid:

```
<!--showdata:
    var="Nodes"
    type="orgnodes"
    format="customtable"
-->
```

The following attribute assignments are invalid:

```
<!--showdata:
    var=Nodes
    type=orgnode
    format=customtable
-->
```

White space

White space is permitted in a smart tag only as a separator between the *tagtype* values and between the attribute assignments.

You cannot use white space between the HTML comment characters and the *tagtype*, or to separate attribute names from the equal sign (=) and the attribute value.

For those attribute assignments that contain a comma-separated list of values, you cannot use white space between the assigned values. White space in the list is interpreted as part of the value of the list element where it occurs. The following example shows a valid attribute assignment that specifies a comma-separated list of values:

```
headerclass="class1,class2,class3"
```

The white space in the following assignment is interpreted as part of the values of elements in the list:

```
headerstyle="class1, class2, class3"
```

This means that the second and third elements in the list have a leading white space in their string value.

Escape characters

The smart tag syntax supports escape characters for the double quotation mark ("), backslash (\), and comma (,) characters only.

For example, to use the double quotation mark in an attribute value, you specify it as `\"`. Other escape characters, such as `\n` or `\t` are not supported. The following example shows how to use the escape characters to specify the double quotation mark as part of a value that is assigned to an attribute:

```
default="My default is \"Default\"."
```

If there is a comma inside an attribute assignment that contains a list of values, you must double-escape the character. This is written in the assignment as three

backslashes followed by the comma (\\\,). You must double-escape the comma character in this instance because the contents of such a list are parsed twice during processing.

Common attributes

Advanced smart tags share a set of common attributes that you must set for every instance of the tag.

These attributes are `var`, `type` and `format`. Together, the common attributes are known as VTF attributes. Table 4 shows the common attributes.

Table 4. Common attributes

Attribute	Description
`var`	Specifies the name of a variable in the operator view policy. This variable stores the value that is displayed by the smart tag. For scalar tags, this variable stores a numeric, Boolean, or string value. For list tags, this variable stores a character-delimited list of values. For orgnode tags, this variable stores an array of data items.
`type`	Specifies the type of data to display. Options are `scalar`, `list` and `orgnode`.
`format`	Specifies how to display the value or values in the operator view. Options are plain, string, url, and action. The `plain` format inserts the data into the operator view as plain text. The `string` format inserts the data inside an HTML span element. The `url` format inserts the data as a link. The `action` format inserts the data as a link or button that opens another operator view.

Overriding attributes

The operator view policy can override values of attributes in a smart tag, except for the common VTF attributes `var`, `type` and `format`.

You dynamically change the attribute values in real time in response to conditions specified in a policy. One typical use of this feature is to dynamically control the CSS style that is used by HTML elements that contain operator view data.

The following basic syntax is for overriding a smart tag attribute from within a policy:

```
variable_attribute=value;
```

Where *`variable`* is the name of the variable that is passed from the policy to the smart tag and *`attribute`* is the name of the attribute to override.

The following example shows how to override the `style` attribute of a smart tag that inserts the value of the `latency` variable into an operator view. The `style` attribute is overwritten if the value of latency is greater than `1000`.

```
threshold = 1000;
If (latency > threshold) {
    latency_style="font-weight: bold; font-size: 10pt; color: red";
}
```

Indexed attributes

Listable tag attributes can be assigned a list of values that is values from the range [*value0*],[*value1*],...,[*valueN*].

Many listable tag attributes can be further modified or augmented individually. You can do that by assigning a value to an original attribute postfixed with an index. We later call such an attribute an "indexed attribute".

An indexed attribute name has the following syntax:

```
attribute_index
```

Where *attribute* is the name of the original attribute and *index* is an arbitrary number or string, depending on the type of an indexed attribute.

You can override an indexed attribute by policy, which means that you can apply the same overriding rules to indexed attributes as you can to original attributes. For more information about overriding attributes, see "Overriding attributes" on page 29.

There are fives types of indexed attributes:

Augmentation
In the augmentation type indexed attributes, the indexed attribute adds to the list, rather than replace an existing attribute. For more information about augmentation type indexed attributes, see "Augmentation type indexed attributes."

Default replacement
In default replacement type indexed attributes, the base attribute holds a single value, considered the default value, and the indexed attribute holds any exceptional values. For more information about default replacement type indexed attributes, see "Default replacement type indexed attributes" on page 31.

Index replacement
Index replacement type attributes take a list from the base attribute, and replace a specific index from within that list. For more information about index replacement type indexed attributes, see "Index replacement type indexed attributes" on page 32.

Field replacement
With field replacement type attributes, you can also index a specific element by field name rather than (or in addition to) an integer index. For more information about field replacement type indexed attributes, see "Field replacement type indexed attributes" on page 32.

Index field replacement
If a specific change is required to a single data item at a specific field in a specific row, then index field replacement is required. For more information about index field replacement type indexed attributes, see "Index field replacement type indexed attributes" on page 33.

Augmentation type indexed attributes

In the augmentation type indexed attributes, the indexed attribute adds to the list, rather than replace an existing attribute.

The following attributes are examples of augmentation type indexed attributes:

- params
- action_fieldparams

For more information about augmentation type indexed attributes, see "params attribute" on page 103 and "action_fieldparams attribute" on page 53.

Syntax of augmentation type attributes

Augmentation type attributes have the following syntax:

```
[attribute]=[value0],[value1],...,[valueN]
[attribute]_[index]=[addval0],[addval1],...,[addvalN]
```

Example of augmentation type attribute usage

The `params` attribute is used in tags of the List tag type with the following syntax:

```
params=[var0],[var1],...,[varN]
params_[index]=[var0],[var1],...,[varN]
```

The `params` list is a set of `var=value` pairs sent with an action. In a List type, you can create a list of actions that are available for the user to click. The unindexed `params` attribute provides the base list of parameters that is sent with every action. However, if you want to send an additional parameter with a specific action, you must specify its position in the list in the indexed attribute.

Assume that the list is two actions long, and you want to send the name/value pairs of userid and `lastname` with each action, but you also want to send `firstname` and `age` with the second action:

```
params="userid,lastname"
params_1="firstname,age"
```

The first action (at index 0) has the following params list:

```
userid,lastname
```

The second action (at index 1) has the following params list:

```
userid,lastname,firstname,age
```

Default replacement type indexed attributes

In default replacement type indexed attributes, the base attribute holds a single value, considered the default value, and the indexed attribute holds any exceptional values.

The following attributes are examples of default replacement type indexed attributes:

- target
- isbutton

For more information about default replacement type indexed attributes, see "target attribute" on page 123 and "action_isbutton attribute" on page 56.

Syntax of default replacement type attributes

Default replacement type attributes have the following syntax:

```
[attribute]=[defaultval]
[attribute]_[index]=[exceptionval]
```

Example of default replacement type attribute usage

In the List tag type, for url format, you can specify a default target to which you are taken after you click a URL. It can be `_self`, `_new`, `_parent`, `_top`, and others.

Assume that you want to click any of the listed URLs to show the resulting display in the same window.

```
target="_self"
```

But, assume that the list has three URLs, and the middle action (index 1) leads to an external help page so you want only that URL to open a new browser window:

```
target_1="_new"
```

Then the three url targets would be as follows:

```
first url: target="_self"
second url: target="_new"
third url: target="_self"
```

The default attribute value is replaced by the indexed attribute value.

Index replacement type indexed attributes

Index replacement type attributes take a list from the base attribute, and replace a specific index from within that list.

The following attributes are examples of index replacement indexed attributes:

- action_class
- action_style
- cellclass
- cellstyle

For more information about index replacement indexed attributes, see "action_class attribute" on page 49, "action_style attribute" on page 60, "cellclass attribute" on page 70, and "cellstyle attribute used in list tag" on page 75.

Syntax of index replacement type attributes

Index replacement type attributes have the following syntax:

```
[attribute]=[value0],[value1],...,[valueN]
[attribute]_[index]=[newvalx]
```

Example of index replacement type attribute usage

The `headerclass` attribute is of the orgnodes type. It holds a list of `.css` files that can be applied to a specific header in an orgnodes table.

Assume that the orgnodes table has three fields, in the following order:

```
userid, lastname, firstname
```

And for each header, assume that you assign a different `.css` file:

```
headerclass="keyhead.css,ltbluehead.css,whitehead.css"
```

Now assume that duplicates were detected in the userid field during runtime, so it could not possibly be a primary key. You want to change the style applied to this header to one of the other `.css` files. Thus, your policy includes this line:

```
[var]_headerclass_0="whitehead.css"
```

Field replacement type indexed attributes

With field replacement type attributes, you can also index a specific element by field name rather than (or in addition to) an integer index.

The following attributes are examples of field replacement type indexed attributes:

- cellclass
- cellstyle

For more information about replacement type indexed attributes, see “cellclass attribute” on page 70 and “cellstyle attribute used in list tag” on page 75.

Syntax of field replacement type attributes

Augmentation type attributes have the following syntax:

```
[attribute]=[value0],[value1],...,[valueN]
[attribute]_[field]=[newvalx]
```

Example of field replacement type attribute usage

The `headerclass` attribute is of the OrgNodes type. It holds a list of `.css` files that can be applied to a specific header in an OrgNodes table.

Assume that the OrgNodes table has three fields, in the following order:

```
userid, lastname, firstname
```

And for each header you assign a different `.css` file:

```
headerclass="keyhead.css,ltbluehead.css,whitehead.css"
```

Assume that duplicates were detected in the userid field during runtime, so it could not possibly be a primary key. You want to change the style applied to this header to one of the other `.css` files. You can change it by field name:

```
[var]_headerclass_userid="whitehead.css"
```

Important:

If aliasing (OrgNodes aliases attribute) is in effect, do not use the alias name for the field. Always use the original field name.

Index field replacement type indexed attributes

If a specific change is required to a single data item at a specific field in a specific row, then index field replacement is required.

The following attributes are examples of index field replacement type indexed attributes:

- action_class
- action_style
- rowcelltext

For more information about index field replacement type indexed attributes, see “action_class attribute” on page 49, “action_style attribute” on page 60, and “rowcelltext attribute” on page 112.

Syntax of index field replacement type attributes

Index field replacement type attributes have the following syntax:

```
[attribute]_[idx]_[field]=[value]
```

In this case, there might not be an individual base attribute.

Example of index field replacement type attribute usage

Assume that the OrgNodes set is populated with three entries.

userid	lastname	firstname	birthdate
12345	Doe	Jane	1973-09-23
24680	Smith	John	5/12/1957
36925	Jones	Bryan	1977-03-21

You use the `rowcelltext_[row]_[field]` to change a specific value in a table. For example:

```
rowcelltext_1_birthdate="1957-05-12"
```

This piece of code results in the following set of values:

userid	lastname	firstname	birthdate
12345	Doe	Jane	1973-09-23
24680	Smith	John	1957-05-12
36925	Jones	Bryan	1977-03-21

Chapter 4. Working with basic smart tags

Basic tags are a type of smart tags. You use basic tags to:

- Specify the name of the server cluster where the operator view is running (required for all display pages).
- Specify the name of the operator view policy (required for all display pages).
- Display the event panel (basic operator views only).
- Display the action panel (basic operator views only).
- Display the information groups panel (basic operator views only).

For more information about basic tags, see "Property tag," "Event panel tag" on page 36, "Action panel tag" on page 36, and "Information groups panel tag" on page 37.

Property tag

You use the property tag to specify the name of the server cluster where the operator view is running and the name of the operator view policy

You must use the property tag in every display page to specify the name of the server cluster and the name of the operator view policy. You use the other basic smart tags in basic operator view display pages only.

The property tag has the following syntaxes:

```
<!--property:DefaultClusterName=clustername-->
<!--property:policy=policyname-->
```

Where *clustername* is the name of the server cluster and *policyname* is the name of the operator view policy, without the `Opview_` prefix. For example, if the name of the policy is `Opview_EX_01`, you must specify the value of the policy attribute as `EX_01`.

Every operator view display page must contain both types of property tags. If you are creating a basic operator view, these tags are automatically inserted when you create the view in the GUI. If you are creating an advanced operator view, you must manually add them to the corresponding display page.

The following example shows how to use the property tag to specify the name of the policy and server cluster in a display page.

```
<html>
<head>
<title>My Operator View</title>
<!-- <!--property:policy="MyView" --> -->
<!-- <!--property:DefaultClusterName="NCICLUSTER_02" --> -->
</head>
.
.
.
</html>
```

In this example, the name of the policy is `MyView` and the name of the cluster is `NCICLUSTER_02`. The property tags are wrapped in an extra set of HTML comment characters to prevent the web browser from displaying the property value.

Event panel tag

You use the event panel tag to display the event panel in an operator view.

You can use this tag in basic display pages such as those created with the GUI. In advanced operator views, you can format and display incoming event values using the scalar and list tags. For more information about event panel tags, see“Scalar tag” on page 39 and “List tag” on page 41.

The event panel has the following syntax:

```
<!--showdata:type=panel-event-->
```

The following example shows how to use the event panel tag in a display page.

```
<html>
<head>
<title>My Operator View</title>
<!-- <!--property:policy="MyView" --> -->
<!-- <!--property:DefaultClusterName="NCICLUSTER_02" --> -->
</head>
<body>
<!--showdata:type="panel-event"-->
</body>
</html>
```

Action panel tag

You use the action panel tag to display the action panel in an operator view.

You can use this tag in basic display pages such as those created with the GUI. In advanced operator views, you can use the scalar, list, and orgnodes tags to freely format and display links that run other policies. For more information about action panel tags, see“Scalar tag” on page 39, “List tag” on page 41 and “OrgNodes tag” on page 44.

The action panel tag has the following syntax:

```
<!--showdata:
    type=panel-action
    format=format
-->
```

Table 5 describes the syntax attributes for this tag:

Table 5. Action panel tag attributes

Attribute	Description
`type`	Specifies the type of data to display. Must be `panel-action`.
`format`	Specifies the format to use in displaying policies in the panel. Can be `horiz` or `vert`. Default is `vert`.

The following example shows how to use the action panel tag in a display page.

```
<html>
<head>
<title>My Operator View</title>
<!-- <!--property:policy=MyView --> -->
<!-- <!--property:DefaultClusterName=NCICLUSTER_02 --> -->
</head>
<body>
<!--showdata:
```

```
    type="panel-action"
    format="horiz"
-->
</body>
</html>
```

Information groups panel tag

You use the information groups panel tag to display the information groups panel in an operator view.

You can use this tag in basic display pages such as those created with the GUI. For advanced operator views, you can freely format and display data using the orgnodes tag. For more information about orgnodes tag, see "OrgNodes tag" on page 44.

The information groups panel tag has the following syntax:

```
<!--showdata:
    type=orgnodes
    format=format
    var=groupname
-->
```

Table 6 describes the syntax attributes for this tag:

Table 6. Information groups panel tag attributes

Attribute	Description
`type`	Specifies the type of data to display. It must be `orgnodes`.
`format`	Specifies the format to use in displaying data in the panel. It can be `tabbed` or `table`. Default is `table`.
`var`	Specifies the name of the information group to display.

The following example shows how to use the information groups panel tag in a display page.

```
<html>
<head>
<title>My Operator View</title>
<!-- <!--property:policy="MyView" --> -->
<!-- <!--property:DefaultClusterName="NCICLUSTER_02" --> -->
</head>
<body>
<!--showdata:
    type="orgnodes"
    format="tabbed"
    var="InfoGroupAdmins"
-->
</body>
</html>
```

In this example, the information groups panel is displayed in the tab delimited format and the name of the group when created in the GUI is `Admins`. The GUI adds the `InfoGroup` prefix when it creates the operator view policy.

Chapter 5. Working with advanced smart tags

Advanced tags are a type of smart tag that you use to format and display data event data and Impact data in the operator view.

You use advanced tags to display data that is stored in variables in the context of the operator view policy. This data can be a scalar value, a character-delimited list of values or a set of data items returned by a function such as `GetByFilter` or `GetByKey`. All the advanced tags are of *`tagtype`* `showdata`.

For more information about advanced smart tags, see "Scalar tag," "List tag" on page 41, and "OrgNodes tag" on page 44.

Scalar tag

You use the scalar tag to display the value of a scalar variable (for example, a number, Boolean, or string) that is set by an operator view policy. This value is set in the policy using the policy language assignment syntax.

Before you insert a scalar tag into the display page, you must make sure that the value of the corresponding variable is set in the policy in string format. This ensures that the value is displayed correctly in the operator view. You can convert any integer, float, or Boolean value to string format within the policy using the `String` function.

You use the scalar tag to specify a format for the string of `plain`, `string`, `URL` or `action`.

The `plain` format displays the scalar value in the operator view as plain text.

The `string` format displays the scalar value in the operator view inside an HTML `span` element. You can set the `id`, `class`, `style` and `title` of the `span` using attributes in the smart tag. You can also specify that the scalar value is a URL that must be displayed in the operator view as a link using the `autourl` attribute.

The `url` format displays the scalar value as a link inside an HTML `span` element. You can set the `href` and `target` attributes of the link using attributes in the smart tag. You can also set the `id`, `class`, `style` and `title` of the `span`.

The `action` format displays the scalar value as a link or button that opens another operator view. You specify the name of the operator view using the `policy` attribute in the smart tag and specify runtime parameters for the view using the `params` attribute.

The scalar tag has the following syntax:

```
<!--showdata:
    var=variable
    type=scalar
    format=plain|string|url|action

    // Core Attributes
    id=id
    class=classname
    style=styletext
```

```
    title=tooltip
    default=msg
    cacheread=true|false
    cachewrite=true|false

    // format=string only
    autourl=true|false

    // format=url only
    url=url
    target=target

    // format=action only
    policy=policyname
    target=targetname
    params=var0,var1...
    isbutton=true|false

 // AJAX-specific attributes (all formats except format="plain")
 update_interval=seconds
 update_option="link|button|none"
 update_delay=seconds
 update_policy=policyname
 update_tag=tagname1, tagname2, ...
 update_params=paramname1, paramname2, ...
 update_precall=functionname
 update_postcall=functionname
 update_effect=effectname
-->
```

Attributes used in scalar tag

This following attributes can be used in scalar tags:

Table 7. List of attributes that can be used in scalar tag

Attribute	Short description
autourl	Boolean type
cacheread	Boolean type
cachewrite	Boolean type
class	String type
default	String type
id	String. This attribute is required if the value of `format` attribute in the smart tag is `string`, `url` or `action`. Otherwise, optional.
isbutton	Boolean type
params	String type
policy	String type
style	String type
target	String type
title	String type
update_interval	Integer type
update_option	String type (either "link," "button," or "none")
update_delay	Integer type
update_label	String type
update_policy	String type

Table 7. List of attributes that can be used in scalar tag (continued)

Attribute	Short description
update_tags and *_override_tags	Comma delimited list of strings that refer to web page element IDs to update through AJAX calls
update_params	Comma delimited list of Strings that refer to web page element IDs
update_precall and update_postcall	String type. This attribute is the name of an available JavaScript function
update_effect	String type. This attribute refers to one of the available effect types listed below
url	String type
var, type, and format	The `var`, `type` and `format` attributes are common attributes that are shared by all the advanced smart tags. These attributes are always required. For information about `var`, `type` and `format`, see "Common attributes" on page 29.

For more information about the attributes that are used in scalar tags, see "Attributes used in advanced smart tags" on page 47.

List tag

You use the list tag to display a list of values that are set by an operator view policy. The operator view displays the list as a formatted table. The list is specified in the policy using the policy language assignment syntax.

The syntax for a valid list of values is as follows:

```
item1,item2,item3 ... itemn
```

Where item is a string value. You must observe the rules for using white space and escape characters as described in "White space" on page 28 and "Escape characters" on page 28.

Before you insert a list tag into the display page, you must make sure that the value of the corresponding variable is set in the policy in string format. This ensures that the value is displayed correctly in the operator view. Using the `String` function, you can convert any integer, float, or Boolean value to string format within the policy.

The follow example shows how to assign a list of values to a variable in the operator view policy. In this example, the list contains four items. White space is not used to separate items in the list.

```
MyList = "one,two,three,four";
```

By default, the operator view displays the items in the list as a formatted table, where each item is a cell in a table row and there is one cell per row. You can change the orientation of the cells in the table using the `orientation` attribute in the list tag.

You use the list tag to specify a format of `string`, `URL` or `action` for the values in the table cells.

The `string` format displays each value in the list inside an HTML `td` element. You can set the `id`, `class`, `style` and `title` of the `td` using attributes in the list tag. You

can also specify that each value is a URL that must be displayed in the operator view as a link using the `autourl` attribute.

The `url` format displays each value in the list as a link inside an HTML `id` element. You can set the `href` and `target` attributes of the link using attributes in the smart tag. You can also set the `id`, `class`, `style` and `title` of the `span`.

The `action` format displays each value in the list as a link or button that opens another operator view. You specify the name of the operator view using the `policy` or `policy_index` attribute in the smart tag, and specify runtime parameters for the view using the `params` or `params_index` attribute.

The list tag has the following syntax:

```
<!--showdata:
    var=variable
    type=list
    format=string|url|action

    // Core Attributes
    id=id
    class=classname
    style=styletext
    title=tooltip

    // common
    default=msg
    delimiter=delimiter
    cacheread=true|false
    cachewrite=true|false
    orientation=horiz|vert
    cellclass=classname0,classname1...
    cellclass_index=classname
    cellstyle=styletext0,styletext1...
    cellstyle_index=classname

    // string only
    autourl=true|false

    // format=url only
    url=url
    url_index=url
    target=target
    target_index=target

    // format=action only
    isbutton=true|false
    isbutton_index=true|false
    policy=policyname
    policy_index=policyname // 1 or the other - policy/url
    url=url
    url_index=url
    target=target
    target_index=target_window
    params=var0,var1...
    params_index=var0,var1...

 // AJAX-specific attributes
 update_interval=seconds
 update_option="link|button|none"
 update_delay=seconds
 update_policy=policyname
 update_tag=tagname1, tagname2, ...
 update_params=paramname1, paramname2, ...
```

```
 update_precall=functionname
 update_postcall=functionname
 update_effect=effectname
-->
```

Attributes used in list tag

This following attributes can be used in list tags:

Table 8. List of attributes that can be used in list tags

Attribute	Short description
autourl	String type
cacheread	Boolean type
cachewrite	Boolean type
cellclass	String or list type. Indexable, index replacement
cellstyle	String or list type. Indexable, index replacement
class	String type
default	String type
delimiter	String type. The default is the comma (,) character
id	String type
isbutton	Boolean type. Indexable, default replacement.
orientation	String type
params	String type. Indexable, augmentation
policy	String type. Indexable, default replacement
style	String type
target	String type. Indexable, default replacement
title	String type
update_interval	Integer type
update_option	String type (either "link," "button," or "none")
update_delay	Integer type
update_label	String type
update_policy	String type
update_tags and *_override_tags	Comma delimited list of strings that refer to Web page element IDs to update through AJAX calls.
update_params	Comma delimited list of Strings that refer to Web page element IDs.
update_precall and update_postcall	String type. The name of an available JavaScript function.
update_effect	String type. This refers to one of the available effect types listed below.
url	String type
var, type, and format	The `var`, `type` and `format` attributes are common attributes that are shared by all the advanced smart tags. These attributes are always required. For information about `var`, `type` and `format`, see “Common attributes” on page 29.

For more information about the attributes used in list tags, see “Attributes used in advanced smart tags” on page 47.

OrgNodes tag

You use the orgnodes tag to display a set of data items retrieved from a data source by the operator view policy.

The operator view displays the data items as a custom table or in per item format. The data items are retrieved in the policy using the `GetByFilter`, `GetByKey` or `DirectSQL` functions, or using another function that returns a set of data items.

This is an example of a statement in a policy that retrieves a set of data items:

```
MyContacts = GetByFilter("Contacts", "Location='New York'", False);
```

When you insert an orgnodes tag into an operator view display page, you specify the name of the variable that stores the data items (in this example, `MyContacts`), as the value of the `var` attribute.

By default, the operator view displays the items in the list as a custom table, where each data item occupies a row in the table and each data item field occupies a cell. In addition, you can display the items in per item format, where each data item occupies a separate table. You also use the orgnodes smart tag to change many parameters that affect how the data items are displayed.

Table 9 shows the formats you can use to display the field values in the data items.

Table 9. OrgNodes tag formats

Format	Description
Custom Table	When you display the field values as a custom table, they are displayed in the resulting operator view in a horizontal grid form, where the column headers are the names of the fields and each row represents a data item. You can optionally append an action column to the right of the custom table. This column can be used to start one or more policies that are related to the data item.
Per Item	When you display the field values in per item format, they are displayed in the resulting operator view in a vertical grid form, where each field value in a data item appears in a separate row. You can optionally append an action row after the last field of each data item. This row can be used to start one or more policies that are related to the data item.

The orgnodes tag has the following syntax:

```
<!--showdata: var=variable
    type=orgnodes
    format=customtable|peritem

    // core html tag attributes
    id=id
    class=classname
    style=styletext
    title=tooltip

    // general
    default=msg
    headerclass=classname0,classname1...
    headerclass_col=classname
    headerclass_field=classname
```

```
   headerstyle=styletext0,styletext1...
   headerstyle_col=styletext
   headerstyle_field=styletext
   rowclass=classname0,classname1...
   rowclass_row=classname
   rowstyle=styletext0,styletext1...
   cellclass=classname0,classname1...
   cellclass_col=classname
   cellclass_field=classname
   cellstyle=styletext0,styletext1...
   cellstyle_col=styletext
   cellstyle_field=styletext
   rowcellclass_row_field=classname
   rowcellstyle_row_field=styletext
   rowcelltext_row_field=text
   showheader=true|false
   autourl=true|false
   includes=field0,field1,...
   excludes=field0,field1,...
   aliases=field0,alias0,field1,alias1,...,fieldN,aliasN
   action_align=left|right (top|bottom also for peritem)\
   action_count=# (def:0)
   action_label=[String]
   action_label_[actionidx]=[String]
   action_hiderow=true|false
   action_hiderow_[row]=true|false
   action_hide=true|false
   action_hide_[actionidx]=true|false
   action_disabled=true|false
   action_disabled_[actionidx]=true|false
   action_isbutton=true|false
   action_isbutton_[actionidx]=true|false
   action_policy=[policyname]
   action_policy_[actionidx]=[policyname]
   action_url=[url]
   action_url_[actionidx]=[url]
   action_target=[target]
   action_target_[actionidx]=[target]
   action_fieldparams=[field0],[field1],...,[fieldN]
   action_fieldparams_[actionidx]=[field0],[field1],...,[fieldN]
   action_varparams=[var0],[var1],...,[varN]
   action_varparams_[actionidx]=[var0],[var1],...,[varN]
   action_class=[classname0],[classname1],...,[classnameN]
   action_class_[actionidx]=[classname]
   action_class_[actionidx]_[row]=[classname]
   action_style=[styletext0],[styletext1],...,[styletextN]
   action_style_[actionidx]=[styletext]
   action_style_[actionidx]_[row]=[styletext]

   // peritem only attributes
   spacewidth=[width]
   spaceheight=[height]
   grouping=[1-N]
   reversepair=true|false
   orientation=horiz|vert
   label_text=[label_text]
   label_text_[row]=[label_text]
   label_show=true|false
   label_show_[row]=true|false
   label_class=[classname]
   label_class_[row]=[classname]
   label_style=[styletext]
   label_style_[row]=[styletext]
   label_align=top|bottom|left|right
   label_align_[row]=top|bottom|left|right

// AJAX-specific attributes
```

```
 update_interval=seconds
 update_option="link|button|none"
 update_delay=seconds
 update_policy=policyname
 update_tag=tagname1, tagname2, ...
 update_params=paramname1, paramname2, ...
 update_precall=functionname
 update_postcall=functionname
 update_effect=effectname
-->
```

Attributes used in orgnodes tag

This following attributes can be used in orgnodes tags:

Table 10. List of attributes that can be used in orgnodes tags

Attribute	Short description
action_align	String type
action_count	Integer type
action_label	String type. Indexable
action_hiderow	String type. Indexable
action_hide	Boolean type. Indexable
action_disabled	Boolean type. Indexable
action_isbutton	Boolean type. Indexable
action_policy	String type. Indexable
action_url	String type. Indexable
action_target	String type. The supported values are _self, _top, _parent and _new. Indexable
action_fieldparams	String type. Indexable
action_varparams	String type. Indexable, augmentation
action_class	String type. Indexable
action_style	String type. Indexable
aliases	String type
autourl	Boolean type
cellclass	String or list
cellstyle	String or list
class	String type
default	String type
excludes	String type
grouping	Integer type
headerclass	String or list. Indexable, index replacement, field replacement
headerstyle	String or list. Indexable, index replacement, field replacement
id	String type
includes	String type
label_text	String type. Indexable
label_show	Boolean type

Table 10. List of attributes that can be used in orgnodes tags (continued)

Attribute	Short description
label_class	String type. Indexable
label_style	String type. Indexable
label_align	String type. Indexable
orientation	String type
reversepair	Boolean type
rowcellclass	String type. Indexable, index field replacement
rowcellstyle	String type. Indexable, index field replacement
rowcelltext	String type. Indexable, index field replacement
rowclass	String or list. Indexable, index replacement
rowstyle	String or list. Indexable, index replacement
showheader	Boolean type
spacewidth	Integer type
spaceheight	Integer type
style	String type
title	String type
update_interval	Integer type
update_option	String type (either "link," "button," or "none")
update_delay	Integer type
update_label	String type
update_policy	String type
update_tags and *_override_tags	Comma delimited list of strings that refer to Web page element IDs to update through AJAX calls.
update_params	Comma delimited list of Strings that refer to Web page element IDs.
update_precall and update_postcall	String type. This is the name of an available JavaScript function.
update_effect	String type. This refers to one of the available effect types listed below.
var, type, and format	The `var`, `type` and `format` attributes are common attributes that are shared by all the advanced smart tags. These attributes are always required. For information about `var`, `type` and `format`, see "Common attributes" on page 29.

For more information about the attributes used in list tags, see "Attributes used in advanced smart tags."

Attributes used in advanced smart tags

This section contains attributes that are used in advanced smart tags.

action_align attribute

This attribute specifies where the row of available actions for each data item is displayed.

For custom tables, possible values are `left` and `right`. For per item tables, possible values are `top`, `bottom`, `left` and `right`.

Table 11 shows the properties of the `action_align` attribute.

Table 11. action_align attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	For custom tables, the default is `right`. For per item tables, the default is `bottom`.
Overridable	Yes
Indexable	No

The following example shows how to specify the location of the row of available actions.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count=1
    action_policy="MyOperatorViewPolicy"
    action_align="right"
    action_label="Click here"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
```

```
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_class attribute

This attribute specifies the value of the `class` attribute in the HTML `td` element that contains the action.

The `class` attribute identifies the `td` as one of a class of elements in the HTML DOM. You can use this attribute to format the `td` with CSS or to manipulate it with DHTML and JavaScript code.

To specify this value for all actions, you can assign a list of `class` names to the `action_class` attribute. You can also specify the value for specific actions, by appending an index value starting with zero that identifies the action to the attribute name (for example, `action_class_0`, `action_class_1`, and so on). To specify the value by action and by row, append the action index and then the row index values to the attribute name (for example, `action_class_0_0`, `action_class_0_1`, and so on.).

Table 12 shows the properties of the `action_class` attribute.

Table 12. action_class attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_class` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_class="action"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td class="action"><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td class="action"><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td class="action"><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
```

```
</td>
</tr></table></td>
</tr>
</table>
```

action_count attribute

This attribute specifies the number of actions that are displayed with each data item in the HTML table.

You must specify a value for this attribute in order for actions to be displayed.

Table 13 shows the properties of the `action_count` attribute.

Table 13. action_count attribute properties

Property	Description
Type	Integer
Applies To	Orgnodes tag
Required	Required in order to display any actions
Default	`0`
Overridable	Yes
Indexable	No

The following example shows how to set the `action_count` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_label="Click here"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form><a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here
</a>
</td>
</tr></table></td>
```

```
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_disabled attribute

This attribute specifies that an action associated with all rows in a table is displayed, but disabled.

You specify the action to disable by appending an index value starting with zero that identifies it ot the attribute name (for example, `action_disabled_0`, `action_disabled_1`, and so on).

Table 14 shows the properties of the `action_disabled` attribute.

Table 14. action_disabled attribute properties

Property	Description
Type	Boolean
Applies To	Orgnodes tag
Required	Optional
Default	`false`
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_disabled` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_disabled_0="true"
-->
```

action_fieldparams attribute

This attribute specifies a list of fields in the HTML table whose values are sent to the action policy or URL as a set of name/value pairs when a user clicks an action.

The values are sent as form variables using the HTTP method `POST`. You can handle an incoming form variable in the action policy by referencing its name with the `@` symbol prefixed to it in the same manner that you handle fields in incoming events.

To specify a list of fields for all actions, you assign the list to the `action_fieldparams` attribute. To specify a list for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_fieldparams_0`, `action_fieldparams_1`, and so on).

Table 15 shows the properties of the `action_fieldparams` attribute.

Table 15. action_fieldparams attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_fieldparams` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_fieldparams="Last"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
```

```
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Last" value="Abduallah">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Last" value="Du">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Last" value="Oalaleye">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

The following policy segment shows how to handle the incoming form parameter and how to print its value to the policy log.

```
// Field name was "Last," so policy variable name is "@Last"

Log("Incoming last name: " + @Last);
```

action_hide attribute

This attribute specifies whether to hide specific actions associated with all rows in the HTML table.

To hide all actions, you assign a value of `true` to the `action_hide` attribute. To hide a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_hide_0`, `action_hide_1`, and so on).

Table 16. action_hide attribute properties

Property	Description
Type	Boolean
Applies To	OrgNodes tag
Required	Optional
Default	`false`
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_hide` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_hide="true"
-->
```

action_hiderow attribute

This attribute specifies whether to hide the actions associated with all rows or a specified row in the HTML table.

It is useful in operator view policies where you want to hide actions based on the contents of the associated rows using conditions determined during policy runtime. To hide actions for all rows, you assign a value of true to the `action_hiderow` attribute. To hide actions for a specific row, append an index value starting with zero that identifies it to the attribute name (for example, `action_hiderow_0`, `action_hiderow_1`, and so on).

Table 17 shows the properties of the `action_hiderow` attribute.

Table 17. action_hiderow attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	`false`
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_hiderow` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
```

```
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_hiderow="true"
-->
```

action_isbutton attribute

This attribute specifies that all actions or a specific action associated with rows in the HTML table appears as buttons rather than text links.

To display buttons for all actions, you assign a value of true to the `action_isbutton` attribute. To display a button for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_isbutton_0`, `action_isbutton_1`, and so on).

Table 18 shows the properties of the `action_isbutton` attribute.

Table 18. action_isbutton attribute properties

Property	Description
Type	Boolean
Applies To	Orgodes tag
Required	Optional
Default	`false`
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_isbutton` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_isbutton="true"
-->
```

When this tag is parsed this tag, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="submit" value="Click here"></form>
```

```
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="submit" value="Click here"></form>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="submit" value="Click here"></form>
</td>
</tr></table></td>
</tr>
</table>
```

action_label attribute

This attribute specifies the string text that appears in the HTML link or button that allows a user to do an action that is associated with data items that appear in the HTML table.

If the table contains more than one action, you can specify a different label for each action by appending an index value starting with zero that identifies it to the attribute name (for example, `action_label_0`, `action_label_1`, and so on).

Table 19 shows the properties of the `action_label` attribute.

Table 19. action_label attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	`Null`
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_label` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
```

```
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_label="Click here"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. The specified action label appears in the link text of each action.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_policy attribute

This attribute specifies which operator view you want to open as a result of the action, where the operator view is identified using a truncated name for the associated policy.

You must name operator view policies using the convention `Opview_`*`viewname`*, where *`viewname`* is the name of the operator view. When you specify an operator view using the `action_policy` attribute, you use only the *`viewname`* portion of the policy name.

To specify an operator view to display for all actions, you assign the name to the `action_policy` attribute. To specify a policy for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_policy_0`, `action_policy_1`, and so on).

Table 20 shows the properties of the `action_policy` attribute.

Table 20. action_policy attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	Policy associated with the operator view currently displayed
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_policy` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_label="Click here"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
```

```
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form><a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_style attribute

This attribute specifies the value of the `style` attribute in the HTML `td` element that contains the action.

The `style` attribute contains CSS information that applies to the `td`. You can use this attribute to format the `td` with CSS.

To specify this value for all actions, you can assign a list of `style` values to the `action_style` attribute. You can also specify the value for specific actions, by appending an index value starting with zero that identifies the action to the attribute name (for example, `action_style_0`, `action_style_1`, and so on). To specify the value by action and by row, append the action index and then the row index values to the attribute name (for example, `action_style_0_0`, `action_style_0_1`, and so on).

Table 21 shows the properties of the `action_style` attribute.

Table 21. action_style attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag

Table 21. action_style attribute properties (continued)

Property	Description
Required	Optional
Default	Value of the `var` attribute in the smart tag
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_style` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_style="font-weight:bold"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td style="font-weight:bold"><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td style="font-weight:bold"><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
```

```
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td style="font-weight:bold"><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_target attribute

This attribute specifies the target window where the operator view or URL associated with an action is displayed.

To specify a target for all actions, you assign the window name to the `action_target` attribute. To specify a target for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_target_0`, `action_target_1`, and so on).

Table 22 shows the properties of the `action_target` attribute.

Table 22. action_target attribute properties

Property	Description
Type	String. Values of `_self`, `_top`, `_parent` and `_new` are supported.
Applies To	Orgnodes tag
Required	Optional.
Default	The default is the current window.
Overridable	Yes
Indexable	Yes

The following example shows how to set the `action_target` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_label="Click here"
    action_target="_new"
-->
```

action_url attribute

This attribute specifies which URL you want to open as a result of the action.

To specify a URL to display for all actions, you assign the name to the `action_url` attribute. To specify a URL for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_url_0`, `action_url_1`, and so on).

Table 23 shows the properties of the action_url attribute.

Table 23. action_url attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes

The following example shows how to set the action_url attribute in the HTML table element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    action_count="1"
    action_url="http://www.example.com"
    action_label="Click here"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the MyContacts variable is an array of three data items and each data item contains fields named First, Last, Email and Phone.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="http://www.example.com"></form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="http://www.example.com">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
```

```
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="http://www.example.com">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
</table>
```

action_varparams attribute

This attribute specifies a list of policy variables whose values are sent to the action policy or URL as a set of name/value pairs when a user clicks an action.

The policy variables are set by the operator view policy at policy runtime. The values are sent as form variables using the HTTP method `POST`. You can handle an incoming form variable in the action policy by referencing its name with the `@` symbol prefixed to it in the same manner that you handle fields in incoming events.

To specify a list of policy variables for all actions, you assign the list to the `action_varparams` attribute. To specify a list for a specific action, append an index value starting with zero that identifies the action to the attribute name (for example, `action_varparams_0`, `action_varparams_1`, and so on).

Table 24 shows the properties of the `action_varparams` attribute.

Table 24. action_varparams attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Index type	Augmentation

The following example shows how to set the `action_varparams` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
```

```
    action_count="1"
    action_policy="MyOperatorViewPolicy"
    action_varparams="Location"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. The value of the `Location` variable set in the policy is `New York`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_0" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Location" value="New York">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_1" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Location" value="New York">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
</td>
</tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row -->
<td><table><tr>
<form id="MyContacts_form_0_2" name="MyContacts_form_0_0"
method="post"
action="/opview/displays/NCICLUSTER-MyOperatorViewPolicy.html">
<input type="hidden" name="Location" value="New York">
</form>
<a href="javascript:document.forms.MyContacts_form_0_0.submit()">
Click here</a>
```

```
</td>
</tr></table></td>
</tr>
</table>
```

The following policy segment shows how to handle the incoming form parameter and how to print its value to the policy log.

```
// Field name was "Location," so policy variable name is "@Location"

Log("Incoming last name: " + @Location);
```

aliases attribute

This attribute allows you to specify alternate field names for fields in the data items displayed in the HTML `table`.

The syntax of this attribute is as follows:

```
field0,alias0,[field1],[alias1]...
```

Table 25 shows the properties of the `aliases` attribute.

Table 25. aliases attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

The following example shows how to specify a list of alternative field names.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    aliases="Email,E-mail,Phone,Telephone"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. The names of the `Email` and `Phone` fields are replaced by their aliases in the heading row of the table.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>E-mail</th>
<th>Telephone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
```

```
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

autourl attribute

This attribute specifies whether to automatically format URL text as a link using the HTML `a` tag. Possible values are `true` and `false`.

If the value of the attribute is set to true, the tag value is read to check if it is in valid URL format. If the format is valid, an `a` element is returned in the HTML output, where the value of the `href` attribute is the URL. This attribute is recognized only if the value of the `format` attribute is `string`.

Note: For different tags different values are read to check if they are in valid URL format:

- scalar tag - the scalar value must be a string in valid URL format.
- list tag - each value in the list must be a string in valid URL format.
- orgnodes tag - the text must be in valid URL format.

Table 26 shows the properties of the `autourl` attribute.

Table 26. autourl attribute properties

Property	Description
Type	Boolean (scalar tag, orgnodes tag), String (list tag)
Applies To	Scalar tag, list tag, orgnodes tag
Required	Optional
Default	None (scalar tag, list tag), `true` (orgnodes tag)
Overridable	Yes
Indexable	No

Example of using autourl attribute in scalar tag

The following example shows how to format a URL string as a link in the resulting HTML output.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    autourl="true"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyString` variable is `http://www.example.com`.

```
<span id="MyString" name="MyString">
<a href="http://www.example.com">http://www.example.com</a>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using autourl attribute in list tag

The following example shows how to format URL strings as links in the resulting HTML output.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    autourl="true"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `http://www.example.com,http://www.ibm.com`.

```
<table>
<tr><td>
<a href="http://www.example.com">http://www.example.com</a>
</td><tr>
<tr><td>
<a href="http://www.ibm.com">http://www.ibm.com</a>
</td><tr>
</table>
```

Example of using autourl attribute in orgnodes tag

The following example shows how to format URL strings as links in the resulting HTML output.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="orgnodes"
    autourl="true"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `URL`. The `URL` field contains a formatted URL string.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>URL</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td><a href="http://www.example.com/~pabduallah">
http://www.example.com/~pabduallah
</a>/td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
```

```
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td><a href="http://www.example.com/~mdu">
http://www.example.com/~mdu
</a>/td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td><a href="http://www.example.com/~joalaleye">
http://www.example.com/~joalaleye
</a>/td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

cacheread attribute

This attribute specifies whether to read the value (list of values, in case of the list tag) from the session cache.

Possible values are `true` and `false`. If no value (list of values, in case of the list tag) for the corresponding variable is set in the operator view policy and no default value is set in the smart tag, the session cache is checked and the cached value (list of values, in case of the list tag) is returned if it was previously stored.

Table 27 shows the properties of the `cacheread` attribute.

Table 27. cacheread attribute properties

Property	Description
Type	Boolean
Applies To	Scalar tag, list tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

Example of using cacheread attribute

The following example shows how to read the scalar value from the session cache.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    cacheread="true"
-->
```

You can use the same code to read the list of values from the session cache. You need to change the value of the `type` property, which for a list tag is `type="list"`.

cachewrite attribute

This attribute specifies whether to store the scalar value (or the list of values, in case of the list tag) in the session cache.

Possible values are `true` and `false`. If no value (no list of values, in case of the list tag) for the corresponding variable is set in the operator view policy and no default value is set in the smart tag, the session cache is checked and the cached value (list of values, in case of the list tag) is returned if it was previously stored.

Table 28 shows the properties of the `cachewrite` attribute.

Table 28. cachewrite attribute properties

Property	Description
Type	Boolean
Applies To	Scalar tag, list tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

Example of using cachewrite attribute

The following example shows how to store the scalar value in the session cache.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    cachewrite="true"
-->
```

You can use the same code to store the list of values in the session cache. You need to change the value of the `type` property, which for a list tag is `type="list"`.

cellclass attribute

This attribute specifies the value of the `class` attribute in the HTML `td` elements in the table that contain the list of values (data item field values, in case of orgnodes tag).

This excludes any `td` elements that contain action links or buttons. The `class` attribute identifies the `td` as one of a class of element in the HTML DOM. You can use this attribute to format the `td` with CSS or to manipulate it with DHTML and JavaScript code.

This attribute has the following syntax:

```
cellclass=classname
cellclass=classname0,classname1,classname2 ...
```

Where *`classname`* is the name of a DOM class.

The first supported syntax allows you to specify a single class for every `td` element in the table. The second syntax allows you to specify a list of classes, where each item in the list is associated with an individual `td` element (`th` element, in case of orgnodes tag) in the order it appears in the table (in a row, in case of orgnodes tag).

Note: If the number of `td` elements in the table exceeds the number of specified classes, the list wraps back to the beginning.

This attribute is recognized for all display formats.

Table 29 shows the properties of the `cellclass` attribute.

Table 29. cellclass attribute properties

Property	Description
Type	String or list
Applies To	List tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Indexing Type	Index Replacement (list tag), Default replacement (orgnodes tag)

Example of using cellclass attribute in list tag

The following example shows how to set the same `class` attribute for all the HTML `td` elements that contain the list values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellclass="cell-class"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td class="cell-class">one</td><tr>
<tr><td class="cell-class">two</td></tr>
<tr><td class="cell-class">three</td></tr>
<tr><td class="cell-class">four</td></tr>
</table>
```

The following example shows how to set different `class` attributes for all the HTML `td` elements that contain each list value.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellclass="first,second,third,fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td class="first">one</td><tr>
<tr><td class="second">two</td></tr>
<tr><td class="third">three</td></tr>
<tr><td class="fourth">four</td></tr>
</table>
```

Example of using cellclass attribute in orgnodes tag

The first supported syntax allows you to specify a single class for every td element in the table. The second syntax allows you to specify a list of classes, where each item in the list is associated with an individual th element in the order it appears in a row. If the number of td elements in the table exceeds the number of specified classes, the list wraps back to the beginning.

The following example shows how to set the same class attribute for all the HTML td elements in the table that contains the data item field values.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellclass="cell-class"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the MyContacts variable is an array of three data items and each data item contains fields named First, Last, Email and Phone.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td class="cell-class">Peter</td>
<td class="cell-class">Abduallah</td>
<td class="cell-class">pabduallah@example.com</td>
<td class="cell-class">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="cell-class">Mary</td>
<td class="cell-class">Du</td>
<td class="cell-class">mdu@example.com</td>
<td class="cell-class">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="cell-class">John</td>
<td class="cell-class">Oalaleye</td>
<td class="cell-class">joalaleye@example.com</td>
<td class="cell-class">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the id and name attributes in the table element contain the name of the var attribute in the smart tag as a default.

The following example shows how to set different class attributes for the HTML td elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellclass="first,second,third,fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td class="first">Peter</td>
<td class="second">Abduallah</td>
<td class="third">pabduallah@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="first">Mary</td>
<td class="second">Du</td>
<td class="third">mdu@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="first">John</td>
<td class="second">Oalaleye</td>
<td class="third">joalaleye@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Indexed cellclass attribute in list tag

The `cellclass` attribute with an index used in a list tag allows you to specify or override the `class` attribute individually for each item in the list.

The syntax of this attribute is as follows:

`cellclass_`*`index=class`*

Where *`index`* is an integer that identifies the item in the list and *`class`* is the name of the DOM class. Index values for this attribute are zero-based.

This attribute is recognized for all display formats.

Example of using indexed cellclass attribute in list tag

The following example shows how to set the `class` attribute in the HTML `td` elements that contain each list value.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellclass_0="row-1"
    cellclass_1="row-2"
    cellclass_2="row-3"
    cellclass_3="row-4"
-->
```

When this tag is parsed this tag, it returns the following HTML output to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td class="row-1">one</td><tr>
<tr><td class="row-2">two</td></tr>
<tr><td class="row-3">three</td></tr>
<tr><td class="row-4">four</td></tr>
</table>
```

Indexed cellclass attribute in orgnodes tag

The `cellclass` attribute postfixed with an index used in a orgnodes tag allows you to specify or override the `class` attribute for `td` elements in the table by column or by field name.

The syntax of this attribute is as follows:

`cellclass_`*`col`*`=`*`class`*

or

`cellclass_`*`field`*`=`*`class`*

Where *`col`* is an integer that identifies the column that contains the `td` elements **,** *`field`* is the name of the data type field, and *`class`* is the name of the DOM class. Index values for this attribute are zero-based.

Example of overriding the `class` attribute by column

The following example shows how to set the `class` attribute by column for the HTML `td` elements in the table that contain the data item fields.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellclass_0="first"
    cellclass_1="second"
    cellclass_2="third"
    cellclass_3="fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td class="first">Peter</td>
<td class="second">Abduallah</td>
<td class="third">pabduallah@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="first">Mary</td>
<td class="second">Du</td>
<td class="third">mdu@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
```

```
<td class="first">John</td>
<td class="second">Oalaleye</td>
<td class="third">joalaleye@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Example of overriding the `class` attribute by field name

The following example shows how to set the `class` attribute by field name for the HTML td elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellclass_First="first"
    cellclass_Last="second"
    cellclass_Email="third"
    cellclass_Phone="fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td class="first">Peter</td>
<td class="second">Abduallah</td>
<td class="third">pabduallah@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="first">Mary</td>
<td class="second">Du</td>
<td class="third">mdu@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td class="first">John</td>
<td class="second">Oalaleye</td>
<td class="third">joalaleye@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

cellstyle attribute used in list tag

This attribute specifies the value of the `style` attribute in the HTML `td` elements that contain the list of values.

The `style` attribute contains CSS information that applies to the `td`. You can use this attribute to format the `td` with CSS.

The syntax of this attribute is as follows:

```
cellstyle=style
cellstyle=style0,style1,style2 ...
```

Where *style* is a valid CSS style statement.

The first supported syntax allows you to specify a single style for every `td` element in the table. The second syntax allows you to specify a list of styles, where each item in the list is associated with an individual `td` element in the order it appears in the table.

This attribute is recognized for all display formats.

Table 30 shows the properties of the `cellstyle` attribute.

Table 30. cellstyle attribute properties

Property	Description
Type	String or list
Applies To	List tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Index type	Index Replacement

Example of using cellstyle attribute in list tag

The following example shows how to set the same `style` attribute for all the HTML `td` elements that contain the list values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellstyle="font-family: Verdana; color: red"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td style="font-family: Verdana; color: red">one</td><tr>
<tr><td style="font-family: Verdana; color: red">two</td></tr>
<tr><td style="font-family: Verdana; color: red">three</td></tr>
<tr><td style="font-family: Verdana; color: red">four</td></tr>
</table>
```

The following example shows how to set different `style` attributes for all the HTML `td` elements that contain each list value.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellstyle="color: red,color: green,color: blue,color: black"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td style="color: red">one</td><tr>
<tr><td style="color: green">two</td></tr>
<tr><td style="color: blue">three</td></tr>
<tr><td style="color: black">four</td></tr>
</table>
```

Indexed cellstyle attribute used in list tag

The `cellstyle` attribute postfixed with an index allows you to specify or override the `style` attribute individually for each item in the list.

The syntax of this attribute is as follows:

```
cellstyle_index=style
```

Where *index* is an integer that identifies the item in the list and *style* is a valid CSS style statement. Index values for this attribute are zero-based.

This attribute is recognized for all display formats.

Example of using indexed cellstyle attribute in list tag

The following example shows how to set the `style` attribute in the HTML `td` elements that contain each list value.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    cellclass_0="color: red"
    cellclass_1="color: green"
    cellclass_2="color: blue"
    cellclass_3="color: black"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td style="color: red">one</td><tr>
<tr><td style="color: green">two</td></tr>
<tr><td style="color: blue">three</td></tr>
<tr><td style="color: black">four</td></tr>
</table>
```

cellstyle attribute used in orgnodes tag

This attribute specifies the value of the `style` attribute in the HTML `td` elements in the table that contain data item field values.

This excludes any `td` elements that contain action links or buttons. The `style` attribute contains CSS information that applies to the `td`. You can use this attribute to format the `td` with CSS.

The syntax of this attribute is as follows:

```
cellstyle=style
cellstyle=style0,style1,style2 ...
```

Where *style* is a valid CSS style statement.

The first supported syntax allows you to specify a single style for every td element in the table. The second syntax allows you to specify a list of styles, where each item in the list is associated with an individual td element in the order it appears in a row. If the number of td elements in a row exceeds the number of specified styles, the list wraps back to the beginning.

Table 31 shows the properties of the cellstyle attribute.

Table 31. cellstyle attribute properties

Property	Description
Type	String or list.
Applies To	OrgNodes tag.
Required	Optional.
Default	None.
Overridable	Yes.
Indexable	Yes.
Indexing Type	Index Replacement, field replacement.

Example of using cellstyle attribute in orgnodes tag

The following example shows how to set the same style attribute for all the HTML td elements in the table that contains the data item field values.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellstyle="font-family: Verdana; font-weight: bold"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the MyContacts variable is an array of three data items and each data item contains fields named First, Last, Email and Phone.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td style="font-family: Verdana; font-weight: bold">Peter</td>
<td style="font-family: Verdana; font-weight: bold">Abduallah</td>
<td style="font-family: Verdana; font-weight: bold">pabduallah@example.com</td>
<td style="font-family: Verdana; font-weight: bold">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="font-family: Verdana; font-weight: bold">Mary</td>
<td style="font-family: Verdana; font-weight: bold">Du</td>
<td style="font-family: Verdana; font-weight: bold">mdu@example.com</td>
<td style="font-family: Verdana; font-weight: bold">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="font-family: Verdana; font-weight: bold">John</td>
<td style="font-family: Verdana; font-weight: bold">Oalaleye</td>
<td style="font-family: Verdana; font-weight: bold">joalaleye@example.com</td>
```

```
<td style="font-family: Verdana; font-weight: bold">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes in the `table` element contain the name of the `var` attribute in the smart tag as a default.

The following example shows how to set different `style` attributes for the HTML `td` elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellstyle="color: red,color: blue,color: green,color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td style="color: red">Peter</td>
<td style="color: blue">Abduallah</td>
<td style="color: green">pabduallah@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">Mary</td>
<td style="color: blue">Du</td>
<td style="color: green">mdu@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">John</td>
<td style="color: blue">Oalaleye</td>
<td style="color: green">joalaleye@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Indexed cellstyle attribute used in orgnodes tag

The `cellstyle` attribute with an index allows you to specify or override the `style` attribute for `td` elements in the table by column or by field name.

The syntax of this attribute is as follows:

`cellstyle_`*`col=style`*

or

`cellstyle_`*`field=style`*

Where *col* is an integer that identifies the column that contains the td elements, *field* is the name of the data type field and *style* is a valid CSS style statement.

Example of overriding the cellstyle attribute by column

The following example shows how to set the style attribute by column for the HTML td elements in the table that contain the data item fields.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellstyle_0="color: red"
    cellstyle_1="color: blue"
    cellstyle_2="color: green"
    cellstyle_3="color: black"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the MyContacts variable is an array of three data items and each data item contains fields named First, Last, Email and Phone.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td style="color: red">Peter</td>
<td style="color: blue">Abduallah</td>
<td style="color: green">pabduallah@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">Mary</td>
<td style="color: green">Du</td>
<td style="color: blue">mdu@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">John</td>
<td style="color: green">Oalaleye</td>
<td style="color: blue">joalaleye@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Example of overriding the cellstyle attribute by field

The following example shows how to set the style attribute by field name for the HTML td elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    cellstyle_First="color: red"
    cellstyle_Last="color: blue"
    cellstyle_Email="color: green"
    cellstyle_Phone="color: black"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td style="color: red">Peter</td>
<td style="color: blue">Abduallah</td>
<td style="color: green">pabduallah@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">Mary</td>
<td style="color: blue">Du</td>
<td style="color: green">mdu@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td style="color: red">John</td>
<td style="color: blue">Oalaleye</td>
<td style="color: green">joalaleye@example.com</td>
<td style="color: black">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

class attribute

This attribute specifies the value of the `class` attribute in the HTML `table` element (`span` element, in case of scalar tag) that contains the data items (the scalar value, in case of the scalar tag, and the list of values in case of the list tag).

The `class` attribute identifies the `table` (`span`, in case of scalar tag) as one of a class of elements in the HTML DOM. You can use this attribute to format the operator view with CSS or to manipulate the DOM with DHTML and JavaScript code.

Note: In case of a scalar tag, this attribute is only recognized if the value of the `format` attribute is `string`, `url` or `action`. In case of a list tag, this attribute is recognized for all display formats.

To specify the class value for individual cells in the table, see “cellclass attribute” on page 70.

Table 32 shows the properties of the `class` attribute.

Table 32. class attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes

Table 32. class attribute properties (continued)

Property	Description
Indexable	No

Example of using class attribute in scalar tag

The following example shows how to set the `class` attribute in the HTML `span` element that contains the scalar value.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    class="string-element"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Testing`.

```
<span id="MyString" name="MyString" class="string-element">Testing</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using class attribute in list tag

The following example shows how to set the `class` attribute in the HTML `table` element that contains the list of values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    class="table-class"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table class="table-class">
<tr><td>one</td><tr>
<tr><td>two</td></tr>
<tr><td>three</td></tr>
<tr><td>four</td></tr>
</table>
```

Example of using class attribute in orgnodes tag

The following example shows how to set the `class` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    class="table-class"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element" class="table-class">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td><
td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

default attribute

This attribute specifies a default value that is displayed in the HTML output if no value for the corresponding variable is set in the operator view policy.

For OrgNode tags, the default value appears as plain text in the resulting HTML output. The operator view does not return a complete OrgNodes table when the default value is displayed.

Table 33 shows the properties of the `default` attribute.

Table 33. default attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag, orgnode tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

Example of using default attribute in scalar tag

The following example shows how to specify a default value for the scalar tag.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    default="Default string goes here"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is not assigned in the policy.

```
<span id="MyString" name="MyString" title="Some tooltip help here">
Default string goes here
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using default attribute in list tag

The following example shows how to specify a list of default value for the list tag.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    default="four,three,two,one"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is not assigned in the policy.

```
<table>
<tr><td>four</td><tr>
<tr><td>three</td></tr>
<tr><td>two</td></tr>
<tr><td>one</td></tr>
</table>
```

Example of using default attribute in orgnodes tag

The following example shows how to specify a default value for the orgnodes tag.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    title="No data available."
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is not assigned in the policy.

```
No data available.
```

delimiter attribute

This attribute specifies the character used to separate items in the list.

Table 34 shows the properties of the `delimiter` attribute.

Table 34. delimiter attribute properties

Property	Description
Type	String

Table 34. delimiter attribute properties (continued)

Property	Description
Applies To	List tag
Required	Optional
Default	The default is the comma character.
Overridable	Yes
Indexable	No

Example of using delimiter attribute

The following example shows how to specify a delimiter character for the list of values that is displayed by the tag.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    delimiter="|"
    default="four|three|two|one"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyList` variable is not assigned in the policy.

```
<table>
<tr><td>four</td><tr>
<tr><td>three</td></tr>
<tr><td>two</td></tr>
<tr><td>one</td></tr>
</table>
```

excludes attribute

This attribute specifies which fields to exclude from the HTML table that contains the data items.

You specify the fields as a comma-separated list of field names.

Table 35 shows the properties of the `excludes` attribute.

Table 35. excludes attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

The following example shows how to set the `excludes` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    excludes="Email"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. The `Email` field specified by the `excludes` attribute is not displayed.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

grouping attribute

For tables of format `peritem`, this attribute specifies the number of name/value pairs displayed per row.

Table 36 shows the properties of the `grouping` attribute.

Table 36. grouping attribute properties

Property	Description
Type	Integer
Applies To	Orgnodes tag
Required	Optional
Default	1
Overridable	Yes
Indexable	No

The following example shows how to set the `grouping` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
```

```
    format="peritem"
    orientation="horiz"
    grouping="2"
-->
```

headerclass attribute

This attribute specifies the value of the `class` attribute in the HTML `th` elements that contain the list of field names in the table.

The `class` attribute identifies the `th` as one of a class of elements in the HTML DOM. You can use this attribute to format the `th` with CSS or to manipulate it with DHTML and JavaScript code.

The syntax of this attribute is as follows:

```
headerclass=classname
headerclass=classname0,classname1,classname2 ...
```

Where *`classname`* is the name of a DOM class.

The first supported syntax allows you to specify a single class for every `th` element in the table. The second syntax allows you to specify a list of classes, where each item in the list is associated with an individual `th` element in the order it appears. If the number of `th` elements in the table exceeds the number of specified classes, the list wraps back to the beginning.

Table 37 shows the properties of the `headerclass` attribute.

Table 37. headerclass attribute properties

Property	Description
Type	String or list
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Indexing Type	Index replacement, field replacement

The following example shows how to set the same `class` attribute for all the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerclass="header-class"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th class="header-class">First</th>
<th class="header-class">Last</th>
<th class="header-class">Email</th>
```

```
<th class="header-class">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes in the `table` element contain the name of the `var` attribute in the smart tag as a default.

The following example shows how to set different `class` attributes for all the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerclass="first,second,third,fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th class="first">First</th>
<th class="second">Last</th>
<th class="third">Email</th>
<th class="fourth">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
```

```
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Indexed headerclass attribute

The `headerclass` attribute postfixed with an index allows you to specify or override the `class` attribute for `th` elements in the table by column or by field name.

The syntax of this attribute is as follows:

`headerclass_`*`col=class`*

or

`headerclass_`*`field=class`*

Where *`col`* is an integer that identifies the column that contains the `th` elements, *`field`* is the name of the data type field, and *`class`* is the name of the DOM class. Index values for this attribute are zero-based.

Example of overriding the `class` attribute by column

The following example shows how to set the `class` attribute by column for the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerclass_0="first"
    headerclass_1="second"
    headerclass_2="third"
    headerclass_3="fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th class="first">First</th>
<th class="second">Last</th>
<th class="third">Email</th>
<th class="fourth">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
```

```
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Example of overriding the `class` attribute by field name

The following example shows how to set the `class` attribute by field name for the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerclass_First="first"
    headerclass_Last="second"
    headerclass_Email="third"
    headerclass_Phone="fourth"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th class="first">First</th>
<th class="second">Last</th>
<th class="third">Email</th>
<th class="fourth">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

headerstyle attribute

This attribute specifies the value of the `style` attribute in the HTML `th` elements that contain the list of field names in the table.

The `style` attribute contains CSS information that applies to the `td`. You can use this attribute to format the `td` with CSS.

The syntax of this attribute is as follows:

```
headerstyle=style
headerstyle=style0,style1,style2 ...
```

Where *style* is a valid CSS style statement.

The first supported syntax allows you to specify a single style for every th element in the table. The second syntax allows you to specify a list of styles, where each item in the list is associated with an individual th element in the order it appears. If the number of th elements in the table exceeds the number of specified styles, the list wraps back to the beginning.

Table 38 shows the properties of the headerstyle attribute.

Table 38. headerstyle attribute properties

Property	Description
Type	String or list.
Applies To	OrgNodes tag.
Required	Optional.
Default	None.
Overridable	Yes.
Indexable	Yes.
Indexing Type	Index replacement, field replacement.

The following example shows how to set the same style attribute for all the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerstyle="font-family: Verdana; font-weight: bold"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the MyContacts variable is an array of three data items and each data item contains fields named First, Last, Email and Phone.

```
<table id="table-element" name="table-element">
<tr>
<th style="font-family: Verdana; font-weight: bold">First</th>
<th style="font-family: Verdana; font-weight: bold">Last</th>
<th style="font-family: Verdana; font-weight: bold">Email</th>
<th style="font-family: Verdana; font-weight: bold">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
```

```
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes in the `table` element contain the name of the `var` attribute in the smart tag as a default.

The following example shows how to set different `style` attributes for the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerstyle="color: red,color: green,color: blue,color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th style="color: red">First</th>
<th style="color: blue">Last</th>
<th style="color: green">Email</th>
<th style="color: black">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Indexed headerstyle attribute

The `headerstyle` attribute postfixed with an index allows you to specify or override the `style` attribute for `th` elements in the table by column or by field name.

The syntax of this attribute is as follows:

`headerstyle_`*`col=class`*

or

`headerstyle_field=class`

Where *col* is an integer that identifies the column that contains the `th` elements, *field* is the name of the data type field, and *style* is a valid CSS style statement. Index values for this attribute are zero-based.

Example of overriding the `style` attribute by column

The following example shows how to set the `style` attribute by column for the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerstyle_0="color: red"
    headerstyle_1="color: blue"
    headerstyle_2="color: green"
    headerstyle_3="color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th style="color: red">First</th>
<th style="color: blue">Last</th>
<th style="color: green">Email</th>
<th style="color: black">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Example of overriding the `style` attribute by field name

The following example shows how to set the `style` attribute by field name for the HTML th elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    headerstyle_First="color: red"
```

```
    headerstyle_Last="color: blue"
    headerstyle_Email="color: green"
    headerstyle_Phone="color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th style="color: red">First</th>
<th style="color: blue">Last</th>
<th style="color: green">Email</th>
<th style="color: black">Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

id attribute

This attribute specifies the value of the `id` and `name` attributes in the HTML `span` element (`table` element, in case of list tag and orgnodes tag) that contains the scalar value (the list of values, in case of list tag, and the data items, in case of orgnodes tag).

The `id` attribute uniquely identifies the `span` (`table`, in case of list tag and orgnodes tag) in the HTML document object model (DOM). You can use this attribute to format the operator view with CSS or to manipulate the DOM with DHTML and JavaScript code.

Note: In case of the scalar tag, this attribute is recognized only if the value of the `format` attribute is `string`, `url` or `action`. In case of the list tag, this attribute is recognized for all display formats.

Table 39 shows the properties of the `id` attribute.

Table 39. id attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag, orgnodes tag

Table 39. id attribute properties (continued)

Property	Description
Required	Required if the value of `format` attribute in the smart tag is `string`, `url` or `action` (in case of the scalar tag). Otherwise, optional.
Default	Value of the `var` attribute in the smart tag.
Overridable	Yes
Indexable	No

Example of using id attribute in scalar tag

The following example shows how to set the `id` attribute in the HTML `span` element that contains the scalar value.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    id="string-element"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Testing`.

```
<span id="string-element" name="string-element">Testing</span>
```

Example of using id attribute in list tag

The following example shows how to set the `id` attribute in the HTML `table` element that contains the list of values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    id="table-element"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table id="table-element" name="table-element">
<tr><td>one</td><tr>
<tr><td>two</td></tr>
<tr><td>three</td></tr>
<tr><td>four</td></tr>
</table>
```

Example of using id attribute in orgnodes tag

The following example shows how to set the `id` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    id="table-element"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

includes attribute

This attribute specifies which fields to include in the HTML table that contains the data items.

You specify the fields as a comma-separated list of field names. This attribute takes precedence over the `excludes` attribute.

Table 40 shows the properties of the `includes` attribute.

Table 40. includes attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

The following example shows how to specify which fields are displayed in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    includes="First,Last"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. Only the `First` and `Last` fields specified by the `includes` attribute are displayed.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

isbutton attribute

This attribute specifies whether to format the scalar value (list value, in case of the list tag) inserted by the tag as a button instead of a link.

Possible values are `true` and `false`.

This attribute is only recognized if the value of the `format` attribute is `action`.

Table 41 shows the properties of the `isbutton` attribute.

Table 41. isbutton attribute properties

Property	Description
Type	Boolean
Applies To	Scalar tag, list tag
Required	Required if the value of the `format` attribute in the smart tag is `action`. Otherwise, this is not recognized.
Default	The default is `false`
Overridable	Yes
Indexable	Yes, in case of the list tag
Index type	Default replacement

Example of using isbutton attribute in scalar tag

The following example shows how to format a value inserted by a scalar tag as a button.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="action"
    policy="MY_POLICY_01"
    isbutton="true"
-->
```

When this tag is parsed, it returns HTML output similar to the following to the Web browser, where the value of the `MyString` variable is `Click to launch view`.

```
<span id="MyString" name="MyString" class="my-class">
<form  id="MyString_form_0_0" name="MyString_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_02.html">
<input type="submit" value="Click to launch view">
</form>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using isbutton attribute in list tag

The following example shows how to format a value inserted by a list tag as a button.

```
<!--showdata:
    var="MyList"
    type="list"
    format="action"
    policy="MY_POLICY_01"
    isbutton="true"
-->
```

When this tag is parsed, it returns HTML output similar to the following example to the Web browser, where the value of the `MyList` variable is `First View,Second View`.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html"></form>
<input type="submit" value="First View">
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_02.html">
</form>
<input type="submit" value="Second View">
</td></tr>
</table>
```

Indexed isbutton attribute

The `isbutton` attribute posfixed with an index can be used in a list tag.

It allows you to specify or override the button setting for each item in the list. Any value that you specify using this attribute overrides the `isbutton` attribute as it applies to the item.

The syntax of this attribute is as follows:

```
isbutton_index=true|false
```

Where *index* is an integer that identifies the item in the list. Index values for this attribute are zero-based.

This attribute is only recognized if the value of the `format` attribute is `action`.

The following example shows how to specify the button setting for a list of action links.

```
<!--showdata:
    var="MyList"
    type="list"
    format="action"
    policy_0="MY_POLICY_01"
    policy_1="MY_POLICY_02"
    isbutton_0="true"
    isbutton_1="false"
-->
```

When this tag is parsed this tag, it returns HTML output similar to this example to the Web browser, where the value of the `MyList` variable is `First View,Second View`.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
</form>
<input type="submit" value="First View">
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_02.html">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
Second View
</a>
</td></tr>
</table>
```

label_align attribute

For tables of format peritem, this attribute specifies the position of the label with respect to the information in the data item fields.

Possible values are `top`, `bottom`, `left` and `right`.

Table 42 shows the properties of the `label_align` attribute.

Table 42. label_align attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	`top`
Overridable	Yes
Indexable	Yes

The following example shows how to set the label_align attribute in the HTML table element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    label_align="top"
-->
```

label_class attribute

For tables of peritem format, this attribute specifies the value of the class attribute in the HTML td element that contains the label.

The class attribute identifies the td as one of a class of elements in the HTML DOM. You can use this attribute to format the td with CSS or to manipulate it with DHTML and JavaScript code.

Table 43 shows the properties of the label_class attribute.

Table 43. label_class attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes

The following example shows how to set the label_class attribute in the HTML table element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    label_class="label"
-->
```

label_show attribute

For tables of peritem format, this attribute specifies whether to display the label for each data item.

By default, the label is the value of the data item key field. You can use this attribute to suppress display of the label.

Table 44 shows the properties of the label_show attribute.

Table 44. label_show attribute properties

Property	Description
Type	Boolean
Applies To	Orgnodes tag
Required	Optional
Default	true

Table 44. label_show attribute properties (continued)

Property	Description
Overridable	Yes
Indexable	No

The following example shows how to set the `label_show` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    label_show="false"
-->
```

label_style attribute

For tables of format `peritem`, this attribute specifies the value of the `style` attribute in the HTML `td` element that contains the label.

The `style` attribute contains CSS information that applies to the `td`. You can use this attribute to format the `td` with CSS.

Table 45 shows the properties of the `label_style` attribute.

Table 45. label_style attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes

The following example shows how to set the `label_style` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    label_style="font-weight: bold"
-->
```

label_text attribute

For tables of `peritem` format, this attribute specifies the contents of the label that is displayed with each data item.

By default, the label is the value of the data item key field. You can use this attribute to override the default value.

Table 46 on page 102 shows the properties of the `label_text` attribute.

Table 46. label_text attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	Value of the data item key field
Overridable	Yes
Indexable	Yes

The following example shows how to set the `label_text` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    label_text="Contact:"
-->
```

orientation attribute used in list tag

This attribute specifies whether the HTML `table` element that contains the list of values is arranged in horizontal or vertical format.

Possible values are `horiz` and `vert`. The default value is vert. When the table is arranged in horizontal format, each of the list values occupies a cell in a single table row. When the table is arranged in vertical format, each of the list values occupies a cell in its own row.

Table 47 shows the properties of the `orientation` attribute.

Table 47. orientation attribute properties

Property	Description
Type	String
Applies To	List tag
Required	Optional
Default	The default is `vert`.
Overridable	Yes
Indexable	No

Example of using orientation attribute in list tag

The following example shows how to specify a horizontal orientation for the HTML table element that displays the list of values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    orientation="horiz"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td>one</td><td>two</td><td>three</td><td>four</td></tr>
</table>
```

orientation attribute used in orgnodes tag

For tables of format `peritem`, this attribute specifies whether name/value pairs in each data item are displayed horizontally, where the name of the field and the value are in the same row, or vertically, where the name of the field appears as a separate row.

Possible values are `horiz` and `vert`.

Table 48 shows the properties of the `orientation` attribute.

Table 48. orientation attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	`vert`
Overridable	Yes
Indexable	No

Example of using orientation attribute in orgnodes tag

The following example shows how to set the `orientation` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    orientation="vert"
-->
```

params attribute

This attribute specifies a list of variables whose values are to be sent to another operator view as runtime parameters.

This attribute works with the `policy` attribute. Values for the runtime parameters are set in the operator view policy.

Note: In case of the list tag, you can only use this attribute to specify the same parameters for every operator view in the list. If you want to augment the parameters for each item with additional parameters, you can use the `params_index` attribute as described below.

This attribute is only recognized if the value of the `format` attribute is `action`.

Table 49 shows the properties of the `params` attribute.

Table 49. params attribute properties

Property	Description
Type	String

Table 49. params attribute properties (continued)

Property	Description
Applies To	Scalar tag, list tag
Required	Required if the value of the format attribute in the smart tag is `action`. Otherwise, this is not recognized.
Default	None
Overrideable	Yes
Indexable	Yes, in case of the list tag
Index type	Augmentation

Example of using params attribute in scalar tag

The following example shows how to specify runtime parameters in an operator view policy.

```
First = "Sanjay";
Last = "Johnson";
Location = "Chicago";
Email = "555-5555";
Phone = "sjohnson@example.com";
```

The following example shows how to specify these same runtime parameters in the scalar tag in an operator view display page.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="action"
    policy="MY_POLICY_01"
    params="First,Last,Location,Email,Phone"
-->
```

When this tag is parsed, it returns HTML output similar to the following example to the Web browser, where the value of the `MyString` variable is `Click to launch view`. The runtime parameters are inserted into the HTML output as hidden `input` elements, where the name of the element is the parameter name and the value is the value assigned to them in the operator view policy.

```
<span id="MyString" name="MyString" class="my-class">
<form id="MyString_form_0_0" name="MyString_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
<input type="hidden" name="phone" value="555-5555">
<input type="hidden" name="email" value="sjohnson@example.com">
<input type="hidden" name="last" value="Johnson">
<input type="hidden" name="location" value="Chicago">
<input type="hidden" name="first" value="Sanjay">
</form>
<a href="javascript:document.forms.MyString_form_0_0.submit()">
Click to launch view
</a>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using params attribute in list tag

The following example shows how to specify runtime parameters in an operator view policy.

```
First = "Sanjay";
Last = "Johnson";
Location = "Chicago";
Email = "555-5555";
Phone = "sjohnson@example.com";
```

The following example shows how to specify these same runtime parameters in the list tag.

```
<!--showdata:
    var="MyList"
    type="list"
    format="action"
    policy="MY_POLICY_01"
    params="First,Last,Location,Email,Phone"
-->
```

When this tag is parsed, it returns HTML output similar to the following example to the Web browser, where the value of the `MyList` variable is `First View,Second View`. The runtime parameters are inserted into the HTML output as hidden `input` elements, where the name of the element is the parameter name and the value is the value assigned to them in the operator view policy.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0" method="post"
action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
<input type="hidden" name="phone" value="555-5555">
<input type="hidden" name="email" value="sjohnson@example.com">
<input type="hidden" name="last" value="Johnson">
<input type="hidden" name="location" value="Chicago">
<input type="hidden" name="first" value="Sanjay">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
First View
</a>
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0" method="post"
action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
<input type="hidden" name="phone" value="555-5555">
<input type="hidden" name="email" value="sjohnson@example.com">
<input type="hidden" name="last" value="Johnson">
<input type="hidden" name="location" value="Chicago">
<input type="hidden" name="first" value="Sanjay">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
Second View
</a>
</td></tr>
</table>
```

Indexed params attribute

The `params` attribute postfixed with an index can be used in a list tag.

It allows you to augment the list of parameters passed to an operator view for each item in the list. Any value that you specify using this attribute adds to the parameters specified by the `params` attribute and does not override them.

The syntax of this attribute is as follows:

`params_`*`index`*`=`*`parameters`*

Where *`index`* is an integer that identifies the item in the list and *`parameters`* is the URL. Index values for this attribute are zero-based.

This attribute is only recognized if the value of the format attribute is action.

The following example shows how to specify runtime parameters in an operator view policy. This example specifies two sets of parameters. The first is a set of basic parameters that are passed to every operator view in the list displayed by the list tag. The second is a set of additional parameters that are passed only to the second operator view in the list.

```
// Basic runtime parameters to pass to every operator view in the list

First = "Sanjay";
Last = "Johnson";
Location = "Chicago";

// Additional runtime parameters to pass to the second operator view
// in the list only

Email = "555-5555";
Phone = "sjohnson@example.com";
```

The following example shows how to specify the First, Last and Location variables as default runtime parameters in the list tag. The example also shows how to augment the parameters passed to the second operator view on the list with the Email and Phone variables.

```
<!--showdata:
    var="MyList"
    type="list"
    format="action"
    policy_0="MY_POLICY_01"
    policy_1="MY_POLICY_02"
    params="First,Last,Location"
    params_1="Email,Phone"
-->
```

When this tag is parsed, it returns HTML output similar to the following to the Web browser, where the value of the MyList variable is First View,Second View. The runtime parameters are inserted into the HTML output as hidden input elements, where the name of the element is the parameter name and the value is the value assigned to them in the operator view policy.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0" method="post"
action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
<input type="hidden" name="last" value="Johnson">
<input type="hidden" name="location" value="Chicago">
<input type="hidden" name="first" value="Sanjay">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
First View
</a>
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0" method="post"
action="/opview/displays/NCICLUSTER-MY_POLICY_02.html">
<input type="hidden" name="phone" value="555-5555">
<input type="hidden" name="email" value="sjohnson@example.com">
<input type="hidden" name="last" value="Johnson">
<input type="hidden" name="location" value="Chicago">
<input type="hidden" name="first" value="Sanjay">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
```

```
Second View
</a>
</td></tr>
</table>
```

policy attribute

This attribute specifies the name of another operator view. The specified operator view must reside on the same server cluster as the first operator view.

If the value of the `format` attribute in the tag is `action`, the current operator view opens this second view specified with this attribute when you click the link (one of the links, in case of the list tag) that contains the tag value (list of values, in case of the list tag).

Note: In case of the list tag, the list items are returned in table format, where each item in the list is a cell in the table and each item is a link or button. You can only use this attribute to specify the same operator view for every item in the list. If you want to specify or override different target windows for each item, you must use the `policy_index` attribute as described below.

The value of this attribute must be the name of the policy associated with the operator view, without the `Opview_` prefix. For example, if the name of the operator view policy is `Opview_MY_POLICY_01`, you must assign the value `MY_POLICY_01` to the attribute.

You can specify runtime parameters for the policy using the `params` attribute. For more information about the params attribute, see "params attribute" on page 103.

This attribute is only recognized if the value of the `format` attribute in the tag is `action`.

Table 50 shows the properties of the `policy` attribute.

Table 50. policy attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag
Required	Required if the value of the `format` attribute in the smart tag is `action`. Otherwise, this is not recognized.
Default	None
Overridable	Yes
Indexable	Yes, in case of the list tag
Index type	Default replacement

Example of using policy attribute in scalar tag

The following example shows how to specify the operator view that is run when you click a scalar value that is formatted as an action.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="action"
    policy="MY_POLICY_01"
-->
```

When this tag is parsed, HTML output similar to the one below is returned to the Web browser, where the value of the `MyString` variable is `Click to launch view`.

```
<span id="MyString" name="MyString">
<form id="MyString_form_0_0" name="MyString_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html">
</form>
<a href="javascript:document.forms.MyString_form_0_0.submit()">
Click to launch view
</a>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using policy attribute in list tag

The following example shows how to specify the operator view that is run when you click a list value that is formatted as an action.

```
<!--showdata:
    var="MyList"
    type="scalar"
    format="action"
    policy="MY_POLICY_01"
-->
```

When this tag is parsed, HTML output similar to the one below is returned to the Web browser, where the value of the `MyList` variable is `Click to launch view,Click to launch view`.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html"></form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
Click to launch view
</a>
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html"></form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
Click to launch view
</a>
</td></tr>
</table>
```

Indexed policy attribute

The `policy` attribute postfixed with an index can be used in a list tag.

It allows you to specify a different operator view to open for each item in the list. Any value that you specify using this attribute overrides the `policy` attribute as it applies to the item.

The syntax of this attribute is as follows:

`policy_`*`index`*`=`*`opview`*

Where *`index`* is an integer that identifies the item in the list and *`opview`* is the name of the operator view. Index values for this attribute are zero-based.

This attribute is only recognized if the value of the `format` attribute is `action`.

Example of using indexed policy attribute

The following example shows how to specify the operator views that is run when you click a list value that is formatted as an action.

```
<!--showdata:
    var="MyList"
    type="scalar"
    format="action"
    policy_0="MY_POLICY_01"
    policy_1="MY_POLICY_02"
-->
```

When this tag is parsed, HTML output similar to the one below is returned to the Web browser, where the value of the `MyList` variable is `First View`, `Second View`.

```
<table>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_01.html"></form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">First View</a>
</td></tr>
<tr><td>
<form id="MyList_form_0_0" name="MyList_form_0_0"
method="post" action="/opview/displays/NCICLUSTER-MY_POLICY_02.html">
</form>
<a href="javascript:document.forms.MyList_form_0_0.submit()">
Second View
</a>
</td></tr>
</table>
```

reversepair attribute

For tables of format `peritem`, this attribute specifies the order of the `th` and `td` elements in the HTML table that contain the name and value for each field in the data item.

By default, `th` elements come before the `td` elements in the table. If you set this attribute to `true`, the `th` elements are displayed after the `td` elements.

Table 51 shows the properties of the `reversepair` attribute.

Table 51. reversepair attribute properties

Property	Description
Type	Boolean
Applies To	Orgnodes tag
Required	Optional
Default	`false`
Overridable	Yes
Indexable	No

The following example shows how to set the `reversepair` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    reversepair="true"
-->
```

rowcellclass attribute

This attribute performs the same function as the `cellclass` attribute described in the cellclass attribute, except that it allows you to specify or override the `class` attribute for a specific `td` element in the table by a combination of row and field name.

The syntax of this attribute is as follows:

`rowcellclass_`*`row_field=class`*

Where *`row`* is an integer that identifies the row that contains the `td` elements, *`field`* is the name of the data type field, and *`class`* is the name of the DOM class. Index values for this attribute are zero-based.

Table 52 shows the properties of the `rowcellclass_row_field` attribute.

Table 52. rowcellclass_row_field attribute properties

Property	Description
Type	String
Applies To	OrgNodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Indexing type	Index field replacement

Example of using rowcellclass attribute

The following example shows how to set the `class` attribute by row and field name for the HTML `td` elements in the table that contain the data item fields.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowcellclass_0_First="first"
    rowcellclass_0_Second="second"
    rowcellclass_0_Third="third"
    rowcellclass_0_Fourth="fourth"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
```

```
</tr>
<tr>
<td class="first">Peter</td>
<td class="second">Abduallah</td>
<td class="third">pabduallah@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

rowcellstyle attribute

This attribute performs the same function as the `cellclass` attribute described in the cellstyle attribute used in orgnodes tag, except that it allows you to specify or override the `style` attribute for a specific `td` element in the table by a combination of row and field name.

The syntax of this attribute is as follows:

`rowcellstyle_`*`row_field`*`=`*`style`*

Where *row* is an integer that identifies the row that contains the `td` elements, *field* is the name of the data type field, and *style* is a valid CSS style statement. Index values for this attribute are zero-based.

Table 53 shows the properties of the `rowcellstyle_row_field` attribute.

Table 53. rowcellstyle_row_field attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Indexing type	Index field replacement

The following example shows how to set the `style` attribute by row and field name for the HTML `td` elements in the table that contain the data item fields.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowcellstyle_0_First="color: red"
```

```
    rowcellstyle_0_Second="color: blue"
    rowcellstyle_0_Third="color: green"
    rowcellstyle_0_Fourth="color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td class="first">Peter</td>
<td class="second">Abduallah</td>
<td class="third">pabduallah@example.com</td>
<td class="fourth">123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

rowcelltext attribute

This attribute allows you to specify or override the text value that appears in a `td` element in the table.

The syntax of this attribute is as follows:

`rowcelltext_`*`row_field=text`*

Where *`row`* is an integer that identifies the row that contains the `td` elements, *`field`* is the name of the data type field, and *`text`* is any text string. Index values for this attribute are zero-based.

Table 54 shows the properties of the `rowcelltext_row_field` attribute.

Table 54. rowcelltext_row_field attribute properties

Property	Description
Type	String
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes

Table 54. rowcelltext_row_field attribute properties (continued)

Property	Description
Indexable	Yes
Indexing type	Index field replacement

The following example shows how to set text value by row and field name for HTML `td` elements in the table.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowcelltext_0_First="Anne"
    rowcelltext_0_Second="Rodriguez"
    rowcelltext_0_Third="arodriguez@example.com"
    rowcelltext_0_Fourth="567-123"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Ann</td>
<td>Rodriguez</td>
<td>arodriguez@example.com</td>
<td>567-123</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

rowclass attribute

This attribute specifies the value of the `class` attribute in the HTML `tr` elements that contain the data items in the table.

This excludes any `tr` elements that contain the table header cells or actions. The `class` attribute identifies the `tr` as one of a class of elements in the HTML DOM. You can use this attribute to format the `tr` with CSS or to manipulate it with DHTML and JavaScript code.

The syntax of this attribute is as follows:

```
rowclass=classname
rowclass=classname0,classname1,classname2 ...
```

Where *classname* is the name of a DOM class.

The first supported syntax allows you to specify a single class for every `tr` element in the table. The second syntax allows you to specify a list of classes, where each item in the list is associated with an individual `tr` element in the order it appears. If the number of `tr` elements in the table exceeds the number of specified classes, the list wraps back to the beginning.

Table 55 shows the properties of the `rowclass` attribute.

Table 55. rowclass attribute properties

Property	**Description**
Type	String or list
Applies To	Orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	Yes
Indexing Type	Index replacement

The following example shows how to set the same `class` attribute for all the HTML tr elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowclass="row-class"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr class="row-class">
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr class="row-class">
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
```

```
<tr class="row-class">
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes in the `table` element contain the name of the `var` attribute in the smart tag as a default.

The following example shows how to set different `class` attributes for all the HTML `tr` elements in the table that contain the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowclass="row-a,row-b,row-c"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr class="row-a">
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr class="row-b">
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr class="row-c">
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Indexed rowclass attribute

The `rowclass` attribute postfixed with an index allows you to specify or override the `class` attribute for `tr` elements in the table by row.

The syntax of this attribute is as follows:

`rowclass_`*`row`*`=`*`class`*

Where *row* is an integer that identifies the row that contains the `th` elements and *class* is the name of the DOM class. Index values for this attribute are zero-based.

Example of overriding the `class` attribute by row

The following example shows how to set the `class` attribute by row for the HTML `tr` elements in the table that contain the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowclass_0="row-a"
    rowclass_1="row-b"
    rowclass_2="row-c"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table>
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr class="row-a">
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr class="row-b">
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr class="row-c">
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

rowstyle attribute

This attribute specifies the value of the `style` attribute in the HTML `tr` elements that contain the data items in the table.

This excludes any `tr` elements that contain the table header cells or actions. The `style` attribute contains CSS information that applies to the `tr`. You can use this attribute to format the `tr` with CSS.

The syntax of this attribute is as follows:

```
rowstyle=style
rowstyle=style0,style1,style2 ...
```

Where *style* is a valid CSS style statement.

The first supported syntax allows you to specify a single style for every `tr` element in the table. The second syntax allows you to specify a list of styles, where each item in the list is associated with an individual `tr` element in the order it appears. If the number of `tr` elements in the table exceeds the number of specified styles, the list wraps back to the beginning.

Table 56 shows the properties of the `rowstyle` attribute.

Table 56. rowstyle attribute properties

Property	Description
Type	String or list.
Applies To	OrgNodes tag.
Required	Optional.
Default	None.
Overridable	Yes.
Indexable	Yes.
Indexing Type	Index replacement.

The following example shows how to set the same `style` attribute for all the HTML `tr` elements in the table that contain the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowstyle="font-family: Verdana; font-weight: bold"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr style="font-family: Verdana; font-weight: bold">
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr style="font-family: Verdana; font-weight: bold">
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr style="font-family: Verdana; font-weight: bold">
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
```

```
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes in the `table` element contain the name of the `var` attribute in the smart tag as a default.

The following example shows how to set different `style` attributes for the HTML `tr` elements in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    rowstyle="color: red,color: green,color: blue,color: black"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr style="color: red">
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr style="color: blue">
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr style="color: green">
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

showheader attribute

This attribute specifies whether to display the header row in the HTML `table` element that contains the data items.

Possible values are `true` and `false`.

Table 57 shows the properties of the `showheader` attribute.

Table 57. showheader attribute properties

Property	Description
Type	Boolean

Table 57. showheader attribute properties (continued)

Property	Description
Applies To	Orgnodes tag
Required	Optional
Default	The default is `true`.
Overridable	Yes
Indexable	No

The following example shows how to hide the header row in the table that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    showheader="false"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`. The names of the data item fields are not displayed in a header row in the table.

```
<table>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

spaceheight attribute

This attribute specifies the amount of space (for example, pixels or points) between name/value pairs in a group where the table format is `peritem`, the orientation is `vert`, and the number of groups is greater than one.

The amount of space is specified in CSS-supported units (for example, pixels or points).

Table 58 on page 120 shows the properties of the `spaceheight` attribute.

Table 58. spaceheight attribute properties

Property	Description
Type	Integer
Applies To	Orgnodes tag
Required	Optional
Default	10px
Overridable	Yes
Indexable	No

The following example shows how to set the `spaceheight` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    orientation="vert"
    grouping="2"
    spaceheight="92px"
-->
```

spacewidth attribute

This attribute specifies the amount of space in CSS-supported units between name/value pairs in a group where the table format is `peritem`, the orientation is `horiz`, and the number of groups is greater than one.

The amount of space if specified in CSS-supported units (for example, pixels or points)

Table 59 shows the properties of the `spacewidth` attribute.

Table 59. spacewidth attribute properties

Property	Description
Type	Integer
Applies To	Orgnodes tag
Required	Optional
Default	10px
Overridable	Yes
Indexable	No

The following example shows how to set the `spacewidth` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="peritem"
    grouping="2"
    spacewidth="92px"
-->
```

style attribute

This attribute specifies the value of the `style` attribute in the HTML `table` element (`span` element, in case of the scalar tag and list tag) that contains the data items.

The `style` attribute contains CSS information that applies to the `table` (`span`, in case of the scalar tag and list tag). You can use this attribute to format the `span` with CSS.

Note: In case of the scalar tag, this attribute is only recognized if the value of the `format` attribute is `string`, `url` or `action`. In case of the list tag, this attribute is recognized for all display formats.

To specify the `style` value for individual cells in the table, see "cellstyle attribute used in list tag" on page 75.

Table 60 shows the properties of the `style` attribute.

Table 60. style attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

Example of using style attribute in scalar tag

The following example shows how to set the `style` attribute in the HTML `span` element that contains the scalar value.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    style="font: Verdana; size: 48pt; color: #7f7f7f"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Testing`.

```
<span id="MyString" name="MyString" style="font: Verdana;
size: 48pt; color: #7f7f7f">Testing</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using style attribute in list tag

The following example shows how to set the `class` attribute in the HTML `table` element that contains the list of values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    style="font-family: Verdana; font-size 12pt; color: red"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table style="font-family: Verdana; font-size 12pt; color: red">
<tr><td>one</td><tr>
<tr><td>two</td></tr>
<tr><td>three</td></tr>
<tr><td>four</td></tr>
</table>
```

Example of using style attribute in orgnodes tag

The following example shows how to set the `style` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    style="background-color: gray; border: 2px solid black"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element"
style="background-color: gray; border: 2px solid black">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

target attribute

This attribute specifies a target browser window.

If the value of the `format` attribute in the tag is `url`, the scalar value (value in the list, in case of the list tag) is formatted as a link using the HTML `a` tag. The value of the `target` attribute in the tag is the specified target window. Possible values include `_self`, `_top`, `_parent`, `_new` or any other valid name for a target window.

Note: In case of the list tag, you can only use this attribute to specify the same target window for every item in the list. If you want to specify or override different target windows for each item, you must use the `target_index` attribute as described below.

This attribute is only recognized if the value of the `format` attribute is `action` or `url`.

Table 61 shows the properties of the `target` attribute.

Table 61. target attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag
Required	Required if the value of the format attribute in the smart tag is `action` or `url`. Otherwise, this is not recognized.
Default	None
Overridable	Yes
Indexable	Yes, in case of the list tag
Index type	Default replacement

Example of using target attribute in scalar tag

The following example shows how to specify a target browser window for a scalar value that is formatted as a link.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="url"
    url="http://www.example.com"
    target="_new"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyString` variable is `Example`.

```
<span id="MyString" name="MyString">
<a href="http://www.example.com" target="_new">Example</a>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using target attribute in list tag

The following example shows how to specify a target browser window for a list of values that are formatted as links.

```
<!--showdata:
    var="MyList"
    type="list"
    format="url"
    url="http://www.example.com"
    target="_new"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `Example 1,Example 2`.

```
<table>
<tr><td>
<a href="http://www.example.com" target="_new">Example 1</a>
</td><tr>
<tr><td>
<a href="http://www.example.com" target="_new">Example 2</a>
</td><tr>
</table>
```

Indexed target attribute

The `target` attribute with an index can be used in the list tag.

It allows you to specify or override a different target window for each item in the list. Any value that you specify using this attribute overrides the `target` attribute as it applies to the item.

The syntax of this attribute is as follows:

`target_`*`index`*`=`*`window`*

Where *`index`* is an integer that identifies the item in the list and *`window`* is the name of the target window. Index values for this attribute are zero-based.

This attribute is only recognized if the value of the `format` attribute is `action` or `url`.

The following example shows how to specify a target browser window for a list of values that are formatted as links.

```
<!--showdata:
    var="MyList"
    type="list"
    format="url"
    url_0="http://www.example.com"
    url_1="http://www.ibm.com"
    target_0="example"
    target_1="ibm"
-->
```

When this tag is parsed, it returns the following HTML output to the Web browser, where the value of the `MyList` variable is `Example,IBM`.

```
<table>
<tr><td>
<a href="http://www.example.com" target="example">Example</a>
</td><tr>
```

```
<tr><td>
<a href="http://www.ibm.com" target="ibm">IBM</a>
</td><tr>
</table>
```

title attribute

This attribute specifies the value of the `title` attribute in the HTML `span` element (`table` element, in case of the list tag and orgnodes tag).

The Web browser displays the contents of this attribute when a user moves the mouse over the `span` element (`table` element, in case of the list tag and orgnodes tag). You can use this attribute to provide hover help (ToolTip) for the operator view.

Note: In case of the scalar tag, this attribute is only recognized if the value of the `format` attribute is `string`, `url` or `action`. In case of the list tag and orgnodes tag, this attribute is recognized for all display formats.

Table 62 shows the properties of the `title` attribute.

Table 62. title attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes
Indexable	No

Example of using title attribute in scalar tag

The following example shows how to set the `title` attribute in the HTML `span` element that contains the scalar value.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="string"
    title="Some tooltip help here"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Testing`.

```
<span id="MyString" name="MyString"
title="Some tooltip help here">Testing</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using title attribute in list tag

The following example shows how to set the `title` attribute in the HTML `table` element that contains the list of values.

```
<!--showdata:
    var="MyList"
    type="list"
    format="string"
    title="Some tooltip help"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `one,two,three,four`.

```
<table>
<tr><td>one</td><tr>
<tr><td>two</td></tr>
<tr><td>three</td></tr>
<tr><td>four</td></tr>
</table>
```

Example of using title attribute in orgnodes tag

The following example shows how to set the `title` attribute in the HTML `table` element that contains the data items.

```
<!--showdata:
    var="MyContacts"
    type="orgnodes"
    format="customtable"
    title="Some tooltip help here"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyContacts` variable is an array of three data items and each data item contains fields named `First`, `Last`, `Email` and `Phone`.

```
<table id="table-element" name="table-element">
<tr>
<th>First</th>
<th>Last</th>
<th>Email</th>
<th>Phone</th>
</tr>
<tr>
<td>Peter</td>
<td>Abduallah</td>
<td>pabduallah@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>Mary</td>
<td>Du</td>
<td>mdu@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
<tr>
<td>John</td>
<td>Oalaleye</td>
<td>joalaleye@example.com</td>
<td>123-456</td>
<!-- RIGHT ACTIONS for this row --><td><table><tr></tr></table></td>
</tr>
</table>
```

Here, the `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

url attribute

This attribute specifies a target URL.

If the value of the `format` attribute in the tag is `url`, the scalar value is formatted as a link using the HTML `a` tag (in case of a list tag, the values in the list are formatted as links). The value of the `href` attribute in the tag is the corresponding specified target URL.

Note: In case of a list tag, you can only use this attribute to specify the same target URL for every item in the list. If you want to specify or override different target URLs for each item, you must use the `url_index` attribute as described below.

This attribute is only recognized if the value of the `format` attribute is `url` (scalar tag and list tag) or action (list tag).

Table 63 shows the properties of the `url` attribute.

Table 63. url attribute properties

Property	Description
Type	String
Applies To	Scalar tag, list tag
Required	Required if the value of the `format` attribute in the smart tag is `url`. Otherwise, this is not recognized.
Default	None
Overridable	Yes
Indexable	Yes, in case of the list tag
Index type	Default replacement

Example of using url attribute in scalar tag

The following example shows how to specify a URL target for a scalar value that is formatted as a link.

```
<!--showdata:
    var="MyString"
    type="scalar"
    format="url"
    url="http://www.example.com"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Example`.

```
<span id="MyString" name="MyString">
<a href="http://www.example.com">Example</a>
</span>
```

The `id` and `name` attributes contain the name of the `var` attribute in the smart tag as a default.

Example of using url attribute in list tag

The following example shows how to specify a URL target for values in a list that are formatted as links.

```
<!--showdata:
    var="MyList"
    type="list"
    format="url"
    url="http://www.example.com"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyList` variable is `Example 1,Example 2`.

```
<table>
<tr><td>
<a href="http://www.example.com">Example 1</a>
</td><tr>
<tr><td>
<a href="http://www.example.com">Example 2</a>
</td><tr>
</table>
```

Indexed url attribute

The `url` attribute postfixed with an index can be used in a list tag. It allows you to specify or override a different target URL for each item in the list.

Any value that you specify using this attribute overrides the `url` attribute as it applies to the item.

The syntax of this attribute is as follows:

`url_`*`index=targeturl`*

Where *`index`* is an integer that identifies the item in the list and *`targeturl`* is the URL. Index values for this attribute are zero-based.

This attribute is only recognized if the value of the `format` attribute is `url`.

Example of using indexed url attribute

The following example shows how to specify individual URL targets for list values that are formatted as a link.

```
<!--showdata:
    var="MyList"
    type="list"
    format="url"
    url_0="http://www.example.com"
    url_1="http://www.ibm.com"
-->
```

When this tag is parsed, the following HTML output is returned to the Web browser, where the value of the `MyString` variable is `Example,IBM`.

```
<table>
<tr><td>
<a href="http://www.example.com">Example</a>
</td><tr>
<tr><td>
<a href="http://www.ibm.com">IBM</a>
</td><tr>
</table>
```

update_delay attribute

This attribute is used to specify delays in HTTP calls by the operator view.

A Web page cannot make more than two simultaneous HTTP calls. This can create a problem if you have more than two smart tags that are set to refresh at the same update interval. You can use the `update_delay` to specify update delays for smart tags so that simultaneous HTTP calls are not made.

The following table shows the properties of the `update_delay` attribute.

Table 64. update_delay attribute properties

Property	Description
Value Type	Integer
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	0
Overridable	Yes, but only on initial display page load
Indexable	No

The following example shows how to use the update_delay attribute to stagger HTTP calls:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_interval="10"
      update_delay="0"
   -->

<!--showdata:
      var="cost"
      type="scalar"
      format="string"
      update_interval="10"
      update_delay="3"
   -->

<!--showdata:
      var="quality"
      type="scalar"
      format="string"
      update_interval="10"
      update_delay="6"
   -->
```

When these tags are parsed, each one is updated at different intervals. The time tag updates at 0, 10, 20, 30 seconds, and so on. The cost tag updates at 3, 13, 23, 33 seconds, and so on. The quality tag updates at 6, 16, 26, 36, seconds, and so on.

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_effect attribute

This attribute is used to apply a preset effect from the JavaScript library on updated content.

The following table shows the properties of the `update_effect` attribute:

Table 65. update_effect attributes properties

Property	Description
Value Type	String. This refers to one of the available effect types listed below.
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes, but only on initial display page load
Indexable	No

The following values are valid for the `update_effect` attribute:

`pulse`
Quickly fades the content in and out one time.

`pulse2`
Quickly fades the content in and out two times.

`pulse3`
Quickly fades the content in and out three times.

`shake`
Causes the content to shake horizontally.

`highlight`
Turns the content white and then fades it back to normal.

`highlight-black`
Turns the content black and then fades it back to normal.

`highlight-red`
Turns the content red and then fades it back to normal.

`highlight-blue`
Turns the content blue and then fades it back to normal.

`highlight-dark blue`
Turns the content dark blue and then fades it back to normal.

`highlight-light blue`
Turns the content light blue and then fades it back to normal.

`highlight-green`
Turns the content green and then fades it back to normal.

`highlight-yellow`
Turns the content yellow and then fades it back to normal.

`highlight-orange`
Turns the content orange and then fades it back to normal.

`highlight-purple`
Turns the content purple and then fades it back to normal.

Example of using update_effect attribute

The following example shows how to use the `update_effect` to have the updated content pulse two times after an update:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_interval="30"
      update_effect="pulse2"
      -->
```

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_interval attribute

This attribute specifies how frequently, in seconds, to automatically refresh your operator view page.

The following table shows the properties of the `update_interval` attribute.

Table 66. update_interval attribute properties

Property	Description
Value Type	Integer
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	-1. This option signals that there is no interval refresh.
Overridable	Yes, but only on initial display page load
Indexable	No

Example of using update_interval attribute

The following example shows how to automatically refresh your operator view every 30 seconds:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_interval="30"
    -->
```

This tag gets refreshed every 30 seconds when it is parsed. This example is almost identical for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_label attribute

This attribute is used to change the text that is displayed in the refresh link or button in the operator view.

The following table shows the properties of the `update_label` attribute.

Table 67. update_label attribute properties

Property	Description
Value Type	String

Table 67. update_label attribute properties (continued)

Property	Description
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	Refresh
Overridable	Yes, but only on initial display page load
Indexable	No

Example of using update_label attribute

The following example shows how to use the update_label attribute to change the refresh button text in the operator view:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_option="button"
      update_label="Get Current Time"
      -->
```

When these tags are parsed, a **Get Current Time** button is displayed in the operator view.

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_option attribute

This attribute creates either a refresh link or button in the operator view.

Note: The tag and other smart tags that are listed in the `update_tags` attribute are refreshed. For more information about the update_tags attribute, see "update_tags and *_override_tags attribute" on page 136.

The following table shows the properties of the `update_option` attribute:

Table 68. update_option attribute properties

Property	Description
Value Type	String (either "link," "button," or "none")
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	"None"
Overridable	Yes, but only on initial display page load
Indexable	No

Example of using update_option attribute

The following example shows how to insert a refresh link into the operator view:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_option="link"
   -->
```

The following example shows how to insert a refresh button into the operator view:

```
<!--showdata:
      var="time"
      type="scalar"
      format="string"
      update_option="button"
    -->
```

When this tag is parsed, a refresh button is displayed in the operator view.

These examples also work for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_params attribute

This attribute provides a way for you to send dynamic data with each AJAX call. It references a list of strings that specify IDs on a Web page with contents that you want to send through EventContainer accessible parameters.

The policy can use this during an AJAX update. The elements on the Web page that the IDs reference in the `update_params` attribute are not necessarily AJAX-updated sections. They can be static `<div>` or `<span>` elements.

The following table shows the properties of the `update_params` attribute.

Table 69. update_params attribute properties

Property	Description
Value Type	Comma delimited list of Strings that refer to Web page element IDs
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes, but only on initial display page load
Indexable	No

Example of using update_params attribute

In the following example, there are three smart tags that are used for the display page. The lat, long, and station_address tags all have a refresh interval of 20 seconds. However, the station_address tag makes a Web service call to a gas station provider and uses the `update_params` attribute to update the latitude and longitude coordinates.

```
<!--showdata:
      var="lat"
      type="scalar"
      format="string"
      update_interval="20"
```

```
      update_tags="lat,long"
      -->

<!--showdata:
      var="long"
      type="scalar"
      format="string"
   -->

<!--showdata:
      var="station_address"
      type="scalar"
      format="string"
      update_interval="20"
      update_delay="15"
      update_policy="WS_GasStation"
      update_params="lat,long"
   -->

<!--showdata:
      var="temp"
      type="scalar"
      format="string"
   -->
```

The `update_params` attribute accepts a comma delimited list of ID referenced elements for a page. When these tags are parsed, the latitude and longitude coordinates are updated.

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_policy attribute

This attribute is used to call a different policy than the one that is associated with the current display page.

The following table shows the properties of the `update_policy` attribute.

Table 70. update_policy attribute properties

Property	Description
Value Type	String
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	The policy that is associated with the current display page.
Overridable	Yes, but only on initial display page load.
Indexable	No

Example of using update_policy attribute

In the following example, the smart tag was originally coded to call the `GlobalTime` policy for successive time updates. However, this example shows how to use the `update_policy` to use a different policy for updates:

```
<!--showdata:
      var="time"
      type="scalar"
```

```
      format="string"
      update_option="button"
      update_label="Get Local Time"
      update_policy="LocalTime"
      -->
```

When these tags are parsed, the `LocalTime` policy is used for every successive time update.

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_precall and update_postcall attributes

Before and after any AJAX call is made, you have the option of executing your own JavaScript code. You can use this to change any parameter data that the AJAX call needs to pull, or to determine a new set of tags to update using the `override_tags` option.

You can also use JavaScript code to add your own graphical effects. All operator view pages include the JavaScript library by default.

The following table shows the properties of the `update_precall` and `update_postcall` attributes:

Table 71. update_precall and update_postcall attributes properties

Property	Description
Value Type	String. This is the name of an available JavaScript function.
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes, but only on initial display page load
Indexable	No

Example of using update_precall and update_postcall attributes

The following example shows how to create your own JavaScript functions and then pass the function name into the `update_precall` and `update_postcall` attributes:

```
<script language="javascript">
function fadeOut() {
  Effect.Fade($("time"));
}
function fadeIn() {
  Effect.Appear($("time"));
}
</script>

<!--showdata:
      var="time"
      type="scalar"
      format="string"
```

```
      update_interval="30"
      update_precall="fadeOut"
      update_postcall="fadeIn"
      -->
```

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

update_tags and *_override_tags attribute

This attribute is used to simultaneously update a number of smart tags on the current display page.

The following table shows the properties of the `update_tags` attribute.

Table 72. update_tags attribute properties

Property	Description
Value Type	Comma delimited list of strings that refer to Web page element IDs to update through AJAX calls.
Applies To	Scalar tag (This excludes format="plain" scalar tags), list tag, orgnodes tag
Required	Optional
Default	None
Overridable	Yes. You can override the tag using either of the following options: • At the initial page load by the standard policy variable override. This type of override can be done only at the initial display page load. • Using the *_override_tags property. This method of overriding can be used at any time after the initial load. The *_override_tags property is specified within the HTML file as an attribute inside of a `<div>` tag that specifies a specific ID. The following example shows how to do this for the "location" smart tag: `<div id="location_override_tags" style="visibility:hidden"> location,temperature</div>`
Indexable	No

Example of using update_tags attribute

In the following example, the smart tag was originally coded to call the `GlobalTime` policy for successive time updates. However, this example shows how to use the `update_policy` to use a different policy for updates:

```
<!--showdata:
      var="location"
      type="scalar"
      format="string"
      update_interval="60"
      update_tags="location,wind,sky,temp,pressure,humidity"
      -->

<!--showdata:
      var="wind"
      type="scalar"
      format="string"
```

```
   -->

<!--showdata:
      var="sky"
      type="scalar"
      format="string"
   -->

<!--showdata:
      var="temp"
      type="scalar"
      format="string"
   __>

<!--showdata:
      var="pressure"
      type="scalar"
      format="string"
      -->

<!--showdata:
      var="humidity"
      type="scalar"
      format="string"
     -->
```

When these tags are parsed, new content is displayed for the location, wind, sky, temp, pressure, and humidity tags.

This example also works for a list tag and orgnodes tag, except that the type element is of `type="list"` value for a list tag and `type="orgnodes"` for an orgnodes tag.

var, type, and format attributes

The `var`, `type` and `format` attributes are common attributes that are shared by all the advanced smart tags.

These attributes are always required. For information about `var`, `type` and `format` attributes, see "Common attributes" on page 29.

Note: For the OrgNodes tag, the value of the format attribute can be `customtable`, which displays the data items as rows in a table, or `peritem`, which displays each data item as a separate table.

Appendix A. Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully. These are the major accessibility features you can use with *Netcool/Impact* when accessing it on the *IBM Personal Communications* terminal emulator:

- You can operate all features using the keyboard instead of the mouse.
- You can read text through interaction with assistive technology.
- You can use system settings for font, size, and color for all user interface controls.
- You can magnify what is displayed on your screen.

For more information about viewing PDFs from Adobe, go to the following web site: http://www.adobe.com/enterprise/accessibility/main.html

Appendix B. Notices

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

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

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785 U.S.A.

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

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
1623-14, Shimotsuruma, Yamato-shi
Kanagawa 242-8502 Japan

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

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement might not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

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

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

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

IBM Corporation
2Z4A/101
11400 Burnet Road
Austin, TX 78758 U.S.A.

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

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

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

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

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

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

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

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

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to

IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

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

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at “Copyright and trademark information” at www.ibm.com/legal/copytrade.shtml.

Adobe, Acrobat, PostScript and all Adobe-based trademarks are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

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

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

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

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

Other product and service names might be trademarks of IBM or other companies.

Glossary

This glossary includes terms and definitions for Netcool/Impact.

The following cross-references are used in this glossary:

- See refers you from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
- See also refers you to a related or contrasting term.

To view glossaries for other IBM products, go to www.ibm.com/software/globalization/terminology (opens in new window).

A

assignment operator
An operator that sets or resets a value to a variable. See also operator.

B

Boolean operator
A built-in function that specifies a logical operation of AND, OR or NOT when sets of operations are evaluated. The Boolean operators are &&, || and !. See also operator.

C

command execution manager
The service that manages remote command execution through a function in the policies.

command line manager
The service that manages the command-line interface.

Common Object Request Broker Architecture (CORBA)
An architecture and a specification for distributed object-oriented computing that separates client and server programs with a formal interface definition.

comparison operator
A built-in function that is used to compare two values. The comparison operators are ==, !=, <, >, <= and >=. See also operator.

control structure
A statement block in the policy that is executed when the terms of the control condition are satisfied.

CORBA
See Common Object Request Broker Architecture.

D

database (DB)
A collection of interrelated or independent data items that are stored together to serve one or more applications. See also database server.

database event listener
A service that listens for incoming messages from an SQL database data source and then triggers policies based on the incoming message data.

database event reader
An event reader that monitors an SQL database event source for new and modified events and triggers policies based on the event information. See also event reader.

database server
A software program that uses a database manager to provide database services to other software programs or computers. See also database.

data item
A unit of information to be processed.

data model
An abstract representation of the business data and metadata used in an installation. A data model contains data sources, data types, links, and event sources.

data source
A repository of data to which a federated server can connect and then retrieve data by using wrappers. A data source can contain relational databases, XML files, Excel spreadsheets, table-structured files, or other objects. In a federated system, data sources seem to be a single collective database.

data source adapter (DSA)
A component that allows the application to access data stored in an external source.

data type
An element of a data model that represents a set of data stored in a data source, for example, a table or view in a relational database.

DB See database.

DSA See data source adapter.

dynamic link
An element of a data model that represents a dynamic relationship between data items in data types. See also link.

E

email reader
A service that polls a Post Office Protocol (POP) mail server at intervals for incoming email and then triggers policies based on the incoming email data.

email sender
A service that sends email through an Simple Mail Transfer Protocol (SMTP) mail server.

event An occurrence of significance to a task or system. Events can include completion or failure of an operation, a user action, or the change in state of a process.

event processor
The service responsible for managing events through event reader, event

listener and email reader services. The event processor manages the incoming event queue and is responsible for sending queued events to the policy engine for processing.

event reader
A service that monitors an event source for new, updated, and deleted events, and triggers policies based on the event data. See also database event reader, standard event reader.

event source
A data source that stores and manages events.

exception
A condition or event that cannot be handled by a normal process.

F

field A set of one or more adjacent characters comprising a unit of data in an event or data item.

filter A device or program that separates data, signals, or material in accordance with specified criteria. See also LDAP filter, SQL filter.

function
Any instruction or set of related instructions that performs a specific operation. See also user-defined function.

G

generic event listener
A service that listens to an external data source for incoming events and triggers policies based on the event data.

graphical user interface (GUI)
A computer interface that presents a visual metaphor of a real-world scene, often of a desktop, by combining high-resolution graphics, pointing devices, menu bars and other menus, overlapping windows, icons and the object-action relationship. See also graphical user interface server.

graphical user interface server (GUI server)
A component that serves the web-based graphical user interface to web browsers through HTTP. See also graphical user interface.

GUI See graphical user interface.

GUI server
See graphical user interface server.

H

hibernating policy activator
A service that is responsible for waking hibernating policies.

I

instant messaging reader
A service that listens to external instant messaging servers for messages and triggers policies based on the incoming message data.

instant messaging service
A service that sends instant messages to instant messaging clients through a Jabber server.

IPL See Netcool/Impact policy language.

J

Java Database Connectivity (JDBC)
An industry standard for database-independent connectivity between the Java platform and a wide range of databases. The JDBC interface provides a call level interface for SQL-based and XQuery-based database access.

Java Message Service (JMS)
An application programming interface that provides Java language functions for handling messages.

JDBC See Java Database Connectivity.

JMS See Java Message Service.

JMS data source adapter (JMS DSA)
A data source adapter that sends and receives Java Message Service (JMS) messages.

JMS DSA
See JMS data source adapter.

K

key expression
An expression that specifies the value that one or more key fields in a data item must have in order to be retrieved in the IPL.

key field
A field that uniquely identifies a data item in a data type.

L

LDAP See Lightweight Directory Access Protocol.

LDAP data source adapter (LDAP DSA)
A data source adapter that reads directory data managed by an LDAP server. See also Lightweight Directory Access Protocol.

LDAP DSA
See LDAP data source adapter.

LDAP filter
An expression that is used to select data elements located at a point in an LDAP directory tree. See also filter.

Lightweight Directory Access Protocol (LDAP)
An open protocol that uses TCP/IP to provide access to directories that support an X.500 model and that does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). For example, LDAP can be used to locate people, organizations, and other resources in an Internet or intranet directory. See also LDAP data source adapter.

link An element of a data model that defines a relationship between data types and data items. See also dynamic link, static link.

M

mathematic operator
A built-in function that performs a mathematic operation on two values. The mathematic operators are +, -, *, / and %. See also operator.

mediator DSA
A type of data source adaptor that allows data provided by third-party systems, devices, and applications to be accessed.

N

Netcool/Impact policy language (IPL)
A programming language used to write policies.

O

operator
A built-in function that assigns a value to a variable, performs an operation on a value, or specifies how two values are to be compared in a policy. See also assignment operator, Boolean operator, comparison operator, mathematic operator, string operator.

P

policy A set of rules and actions that are required to be performed when certain events or status conditions occur in an environment.

policy activator
A service that runs a specified policy at intervals that the user defines.

policy engine
A feature that automates the tasks that the user specifies in the policy scripting language.

policy logger
The service that writes messages to the policy log.

POP See Post Office Protocol.

Post Office Protocol (POP)
A protocol that is used for exchanging network mail and accessing mailboxes.

precision event listener
A service that listens to the application for incoming messages and triggers policies based on the message data.

S

security manager
A component that is responsible for authenticating user logins.

self-monitoring service
A service that monitors memory and other status conditions and reports them as events.

server A component that is responsible for maintaining the data model, managing services, and running policies.

service
A runnable sub-component that the user controls from within the graphical user interface (GUI).

Simple Mail Transfer Protocol (SMTP)
An Internet application protocol for transferring mail among users of the Internet.

Simple Network Management Protocol (SNMP)
A set of protocols for monitoring systems and devices in complex networks. Information about managed devices is defined and stored in a Management Information Base (MIB). See also SNMP data source adapter.

SMTP See Simple Mail Transfer Protocol.

SNMP
See Simple Network Management Protocol.

SNMP data source adapter (SNMP DSA)
A data source adapter that allows management information stored by SNMP agents to be set and retrieved. It also allows SNMP traps and notifications to be sent to SNMP managers. See also Simple Network Management Protocol.

SNMP DSA
See SNMP data source adapter.

socket DSA
A data source adaptor that allows information to be exchanged with external applications using a socket server as the brokering agent.

SQL database DSA
A data source adaptor that retrieves information from relational databases and other data sources that provide a public interface through Java Database Connectivity (JDBC). SQL database DSAs also add, modify and delete information stored in these data sources.

SQL filter
An expression that is used to select rows in a database table. The syntax for the filter is similar to the contents of an SQL WHERE clause. See also filter.

standard event reader
A service that monitors a database for new, updated, and deleted events and triggers policies based on the event data. See also event reader.

static link
An element of a data model that defines a static relationship between data items in internal data types. See also link.

string concatenation
In REXX, an operation that joins two characters or strings in the order specified, forming one string whose length is equal to the sum of the lengths of the two characters or strings.

string operator
A built-in function that performs an operation on two strings. See also operator.

U

user-defined function
A custom function that can be used to organize code in a policy. See also function.

V

variable
A representation of a changeable value.

W

web services DSA
A data source adapter that exchanges information with external applications that provide a web services application programming interface (API).

X

XML data source adapter
A data source adapter that reads XML data from strings and files, and reads XML data from web servers over HTTP.

Index

P

R

S

T

U

V

W

Printed in USA

SC27-4853-01