Quest

Foglight® 8.1.0

Web Component Guide

© 2025 Quest Software Inc.

ALL RIGHTS RESERVED.

This guide contains proprietary information protected by copyright. The software described in this guide is furnished under a software license or nondisclosure agreement. This software may be used or copied only in accordance with the terms of the applicable agreement. No part of this guide may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording for any purpose other than the purchaser's personal use without the written permission of Quest Software Inc.

The information in this document is provided in connection with Quest Software products. No license, express or implied, by estoppel or otherwise, to any intellectual property right is granted by this document or in connection with the sale of Quest Software products. EXCEPT AS SET FORTH IN THE TERMS AND CONDITIONS AS SPECIFIED IN THE LICENSE AGREEMENT FOR THIS PRODUCT, QUEST SOFTWARE ASSUMES NO LIABILITY WHATSOEVER AND DISCLAIMS ANY EXPRESS, IMPLIED OR STATUTORY WARRANTY RELATING TO ITS PRODUCTS INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT. IN NO EVENT SHALL QUEST SOFTWARE BE LIABLE FOR ANY DIRECT, INDIRECT, CONSEQUENTIAL, PUNITIVE, SPECIAL OR INCIDENTAL DAMAGES (INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF PROFITS, BUSINESS INTERRUPTION OR LOSS OF INFORMATION) ARISING OUT OF THE USE OR INABILITY TO USE THIS DOCUMENT, EVEN IF QUEST SOFTWARE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Quest Software makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. Quest Software does not make any commitment to update the information contained in this document.

If you have any questions regarding your potential use of this material, contact:

Quest Software Inc.
Attn: LEGAL Dept.
4 Polaris Way
Aliso Viejo, CA 92656

Refer to our website (https://www.quest.com) for regional and international office information.

Patents

Quest Software is proud of our advanced technology. Patents and pending patents may apply to this product. For the most current information about applicable patents for this product, please visit our website at https://www.quest.com/legal.

Trademarks

Quest, the Quest logo, and Join the Innovation are trademarks and registered trademarks of Quest Software Inc. For a complete list of Quest marks, visit https://www.quest.com/legal/trademark-information.aspx. "Apache HTTP Server", Apache, "Apache Tomcat" and "Tomcat" are trademarks of the Apache Software Foundation. Google is a registered trademark of Google Inc. Android, Chrome, Google Play, and Nexus are trademarks of Google Inc. Red Hat, JBoss, the JBoss logo, and Red Hat Enterprise Linux are registered trademarks of Red Hat, Inc. in the U.S. and other countries. CentOS is a trademark of Red Hat, Inc. in the U.S. and other countries. Fedora and the Infinity design logo are trademarks of Red Hat, Inc. Microsoft, .NET, Active Directory, Internet Explorer, Hyper-V, Office 365, SharePoint, Silverlight, SQL Server, Visual Basic, Windows, Windows Vista and Windows Server are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. AIX, IBM, PowerPC, PowerVM, and WebSphere are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Java, Oracle, Oracle Solaris, PeopleSoft, Siebel, Sun, WebLogic, and ZFS are trademarks or registered trademarks of Oracle and/or its affiliates in the United States and other countries. SPARC is a registered trademark of SPARC International, Inc. in the United States and other countries. Products bearing the SPARC trademarks are based on an architecture developed by Oracle Corporation. OpenLDAP is a registered trademark of the OpenLDAP Foundation. HP is a registered trademark that belongs to Hewlett-Packard Development Company, L.P. Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both. MySQL is a registered trademark of MySQL AB in the United States, the European Union and other countries. Novell and eDirectory are registered trademarks of Novell, Inc., in the United States and other countries. VMware, ESX, ESXi, vSphere, vCenter, vMotion, and vCloud Director are registered trademarks or trademarks of VMware, Inc. in the United States and/or other jurisdictions. Sybase is a registered trademark of Sybase, Inc. The X Window System and UNIX are registered trademarks of The Open Group. Mozilla and Firefox are registered trademarks of the Mozilla Foundation. "Eclipse", "Eclipse Foundation Member", "EclipseCon", "Eclipse Summit", "Built on Eclipse", "Eclipse Ready" "Eclipse Incubation", and "Eclipse Proposals" are trademarks of Eclipse Foundation, Inc. IOS is a registered trademark or trademark of Cisco Systems, Inc. and/or its affiliates in the United States and certain other countries. Apple, iPad, iPhone, Mac OS, Safari, Swift, and Xcode are trademarks of Apple Inc., registered in the U.S. and other countries. Ubuntu is a registered trademark of Canonical Ltd. Symantec and Veritas are trademarks or registered trademarks of Symantec Corporation or its affiliates in the U.S. and other countries. OpenSUSE, SUSE, and YAST are registered trademarks of SUSE LCC in the United States and other countries. Citrix, AppFlow, NetScaler, XenApp, and XenDesktop are trademarks of Citrix Systems, Inc. and/or one or more of its subsidiaries, and may be registered in the United States Patent and Trademark Office and in other countries. AlertSite and DéjàClick are either trademarks or registered trademarks of Boca Internet Technologies, Inc. Samsung, Galaxy S, and Galaxy Note are registered trademarks of Samsung Electronics America, Inc. and/or its related entities. MOTOROLA is a registered trademarks of Motorola Trademark Holdings, LLC. The Trademark BlackBerry Bold is owned by Research In Motion Limited and is registered in the United States and may be pending or registered in other countries. Quest is not endorsed, sponsored, affiliated with or otherwise authorized by Research In Motion Limited. Ixia and the Ixia four-petal logo are registered trademarks or trademarks of Ixia. Opera, Opera Mini, and the O logo are trademarks of Opera Software ASA. Tevron, the Tevron logo, and CitraTest are registered trademarks of Tevron, LLC. PostgreSQL is a registered trademark of the PostgreSQL Global Development Group. MariaDB is a trademark or registered trademark of MariaDB Corporation Ab in the European Union and United States of America and/or other countries. Vormetric is a registered trademark of Vormetric, Inc. Intel, Itanium, Pentium, and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries. Debian is a registered trademark of Software in the Public Interest, Inc. OpenStack is a trademark of the OpenStack Foundation. Amazon Web Services, the "Powered by Amazon Web Services" logo, and "Amazon RDS" are trademarks of Amazon.com, Inc. or its affiliates in the United States and/or other countries. Infobright, Infobright Community Edition and Infobright Enterprise Edition are trademarks of Infobright Inc. POLYCOM®, RealPresence® Collaboration Server, and RMX® are registered trademarks of Polycom, Inc. All other trademarks and registered trademarks are property of their respective

owners.

Legend

WARNING: **A WARNING icon indicates a potential for property damage, personal injury, or death.**

CAUTION: **A CAUTION icon indicates potential damage to hardware or loss of data if instructions are not followed.**

IMPORTANT NOTE, **NOTE**, **TIP**, **MOBILE**, or **VIDEO:** An information icon indicates supporting information.

Foglight Web Component Guide
Foglight Version - 8.1.0

Contents

Introducing the Web Component Framework 8
Configuring Views 8
Browser Interface Views 9
The Browser Interface 9
Anatomy of a Typical Dashboard 10
The Web Component Framework (WCF) 10
The Design Tab 11

The Web Component Framework 13
Core Concepts 13
Modules 13
Observations 16
Context 19
Parameters in Bindings 19
On Null Values 20
Renderers 20
Default Values 20
Data Sources, Data Types, and Data Objects 20
Paths 21
Properties 21
Using the Web Component Framework 21
The Web Component Framework Editor 21
An Example Page 22
Managing Dashboards 23
Definitions Panes 23
Data and Data Sources Pages 24
Definitions Pane 24
Customizing the UI Quickly 25
Finding Pages: Bookmarks 25
Additional Documentation 25

Configuring Views and Context 27
Configuring Views 27
Creating a New Container View 27
Stamping Views 29
Searching for Definitions 31
Definitions Page for a View 34
Definitions Pane Settings Tabs 37
General Tab 38
Roles 40
Context tab 41
Configuration Tab 41
Flow Tab 44
Layout Tab 50

Views Tab 50
Context Tab 50
Context Tab 52
Context Types 52
Additional Context 53
Dynamic Context 54
Flow Context Mappings 55

Queries 57
Overview of Query Definitions 57
Creating a Query in Foglight 58
Query Definition Settings 58
Conditional Types 64
Sequence of Evaluation 67
Parameters in Queries 69
Summary of Creating a Query 70

Functions 73
Java Functions 73
Map Functions 74
Script Functions 74
Script with Map Functions 75
Using Functions 76
Functions as Converters 77
Caching results of Functions 77
Information available from a Script, Script with Map or Java Function 77
Invoking a Function from a Function 78
Invoking a Query from a Function 79
Time Range used for testing a function 79

Bindings 80
Simple Types 80
Binding Types 81
Details of each Binding 82
Context 82
Writable Data Object 83
Query 85
Theme 87
Icon 87
String Template 88
Rich Text and Rich Text Template 88
Image Reference 89
Function 89
Localized String 89
Data 90
Data Object Property 90
Data Object Property (and Property) 90
Date 91

List . . . 91
Return Types . . . 91

Additional Components . . . 94
Files . . . 94
Icons . . . 95
Associations . . . 95
Tab Associations . . . 96
Question Associations . . . 96
Category Associations . . . 96
Domain Associations . . . 96
Message Associations . . . 97
Customizable Associations . . . 97
Hierarchy Associations . . . 97
Alarm Associations . . . 97
Activity Associations . . . 97
Action Associations . . . 98
Node Associations . . . 98
Edge Associations . . . 98
Creating Associations . . . 98
Creating Alarm Associations: Example . . . 101
Renderers . . . 106
Default Renderer can be Overridden . . . 106
Input Value . . . 107
Determining the Appropriate Renderer for a Binding . . . 108
Type Mappings . . . 108
Types Tab in the Module List Pane . . . 109
Unit Mappings . . . 109
Tasks . . . 109
Types . . . 110
Creating a Data Type in a Web Component Framework Module . . . 110
Data Object Types . . . 111
Creating a Data Object of your Type . . . 112
Saving Data Objects . . . 113
Setting up RSS Feeds . . . 114
Theme and Module Resources . . . 114
Module-Specific Images . . . 114
Theme-Specific Images in Module Definitions . . . 116
Printing . . . 118
PDF Generation . . . 118
Reports . . . 119
Remote Access to Views . . . 119
Portlet . . . 120
SharePoint Web Part with Windows Single Sign-On . . . 120
Performance Improvements . . . 122
Optimizing Data Access with the Batch API . . . 122

About Us . . . 125

We are more than just a name 125
Our brand, our vision. Together. 125
Contacting Quest 125
Technical support resources 125

1

Introducing the Web Component Framework

This *Web Component Guide* provides configuration instructions, conceptual information, and instructions on how to use the Web Component Framework.

This guide is intended for any user who wants to extend Foglight® using the tools provided by the Web Component Framework to design additional user interface (UI) component

The Web Component Framework (WCF) provides the underlying structure from which you can build browser-based Foglight and Foglight-like user interfaces. WCF is a set of graphic user interface (GUI) controls called components. WCF provides capabilities for interaction and navigation among these components. You can use WCF to build a browser interface for performing specific tasks, such as monitoring services.

The query-based model used by WCF enables you to bind components to data from any source. WCF makes it easy to deploy dashboards and their drilldown views into various application environments. By configuring these views, you can display data in a variety of tabular and graphical formats.

This document provides an introduction to these components and describes the underlying mechanisms that allow them to display data retrieved from Foglight or other sources with the same data structure. The full list of properties for each WCF component is given in the view pages that are accessible from the Help menu on the browser interface.

For a quick introduction to how dashboards are built and populated with sample views, try the *Web Component Tutorial*.

- Configuring Views
- Browser Interface Views
- The Browser Interface

Configuring Views

Foglight® has a configurable Web-based interface. By performing your own custom configurations, you apply your detailed knowledge of your system to augment or replace the views that Foglight shows by default.

You can arrange or modify the existing components in the browser interface by:

- Creating a custom dashboard by dragging existing views or data from the action panel to the display area. This is the simplest way of creating a new view.

  You can add any view that is designated as a portlet, thus building a custom page. The Data tab on the action panel presents choices from which you can drag metric charts and position them on the page.

  In the action panel, you can adjust the width of the views placed on a page by choosing the number of columns.
- Creating a report.
- Adding bookmarks.

- Adding dashboards to your user module (in My Definitions) or to system modules (requires the Cartridge Developer role). This requires more expertise, but it gives you access to the framework, so you can define completely new views.

Browser Interface Views

WCF provides you with the means to customize the browser interface. You can access the browser interface's component framework and create custom views. You can populate these views with other display components, such as charts and tables, and connect them to data sources. It is the same data that the agents are configured to collect, but organized in a way that best fits a given business model and the information needs particular to that model.

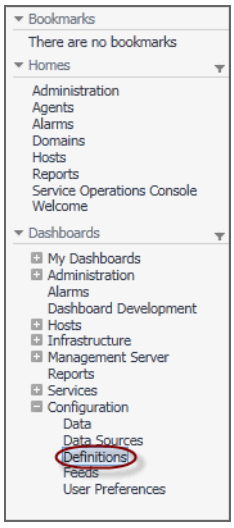
The end result is a monitoring system that organizes data to mirror the business model. Real-time monitoring data is easily viewed and fosters better control of the monitored components' availability. Custom views help inform application and IT managers about end-user service levels, notify stakeholders when those service levels are violated, and allow problem resolution tasks to be assigned to the appropriate domain experts. Custom views that focus on known trouble spots can help establish processes for quick recovery from system failure.

The Browser Interface

The *Foglight® User Guide* describes the overall appearance of the browser interface. This document describes the part of the interface used to define, view, and edit elements of WCF. The Definitions dashboard allows you to examine and work with all existing entities in WCF which you can use to build all the views in the browser interface.

You access the Definitions area by clicking **Definitions** under **Dashboards > Configuration** in the left navigation panel.

Figure 1. Definitions area on the navigation panel



The Design tab on the action panel is available on any non-portal dashboard if you have the Dashboard Designer role. This tab shows a hierarchical list of all the views on the dashboard.

In the Definition tab, select a view from the upper pane to display information about it in the lower pane. You can also access its child views from this pane. Click the **Inspect** icon to switch to viewing information about the dashboard or view in the Definitions editor. You can see the current bindings for the view's context keys in the Context tab.

Anatomy of a Typical Dashboard

- A dashboard consists of a container, which in turn contains views. See View Components on page 10 for more information about containers.
- Views are assembled from WCF components.
- View components are often required to display specific data, which might be set at design time or might be based on user interaction. This demands query-based data retrieval and a mechanism for passing information in the form of parameters. WCF provides both.
- In WCF, the query mechanism retrieves data from a data source, for example, from the Monitoring data source.
- Queries can be parameterized, which gives them an extra degree of flexibility with regard to the dynamic data that they retrieve.
- A flow mechanism permits pages to be updated or linked to other pages.
- A context mechanism enables values, which may be objects, to be passed to dependent pages. Thus, dynamically retrieved data on a parent page can be passed to a dependent page.

The starting place for working with WCF components is the Definitions dashboard. You may find it useful to have the browser interface open and navigate to the Definitions dashboard so that you can refer to it as you continue reading.

The Web Component Framework (WCF)

WCF consists of a structure for hosting related views called view components, and container services that host data sources. It is a superset of the View Component collection that contains other control components, such as renderers. It is used to build thin client interfaces for products that are primarily (but not necessarily) in the systems management domain.

WCF is written in Java™ and is capable of running in a Web container such as Apache Tomcat. It can be used on contemporary Web browsers without requiring the use of a plug-in. It is portal-like, but is not a JSR-168 standard portal.

WCF supports multiple data sources. With it, you can configure multiple data queries and display the retrieved data using views. Queries are the primary mechanism used to extract data from WCF data sources. A view can use more than one query to extract data for display, and a query can use more than one data object.

Apart from creating browser interface elements using WCF, there are several important considerations:

- The Dashboard interface (default views)
- Data interface (data representation, relationships in the system, ability to query data)
- Persistence interface (data storage)
- Permissions interface (ability to set rules and privileges).

See the Foglight® core documentation set for more information about these elements.

View Components

View components are elements of WCF. They are the visible components in the browser interface. Multiple components can be arranged on a page, and some components can be nested within others. A view contains both view components and configuration information. Views can be interactive and respond to user actions.

Types of view components:

- Containers, such as various layouts, splitters, and reports. Layouts control the placement of the components they contain, which are known as child components. For more information, see the *Web Component Reference*.

- Data visualization components, such as charts, tables, gauges, labels, and trees. These views can manage child views using links on the navigation or action panels, or if defined as customizers. For more information, see the *Web Component Reference*.
- Input components, with batch submit capability for use in forms.
- Specialized components, such as RSS feeds. For more information, see the *Web Component Reference* and Setting up RSS Feeds on page 114.

The configuration settings include flow control, contextual inputs, data binding, and query specification.

The *Foglight® User Guide* covers the following topics in greater detail.

Pages can be decorated with headers, which may contain:

- Breadcrumbs, which include the present page name preceded by other page names
- Time Region, which may contain
  - Timestamp, if no time range is applied to the page
  - Time Range, if available and applies to all views
  - A Zonar, if available, permits a choice of the time interval
  - Nothing, if multiple time ranges are represented
- Optional Page Scope Actions, such as:
  - Time Range, changes the time range for components on the page

The Module List Pane

Navigate to **Dashboards > Configuration > Definitions** and you will see the module list pane at the upper left. Foglight view components are grouped into modules that potentially contain views, queries, functions, renderers, and so on. Modules are an organizational framework for keeping related view components together. Top-level modules are created by Foglight® designers, but it is possible to create sub-modules in any of the system modules. User modules, such as My Definitions, the current user's own workspace for creating a new view component, cannot have sub-modules.

The Design Tab

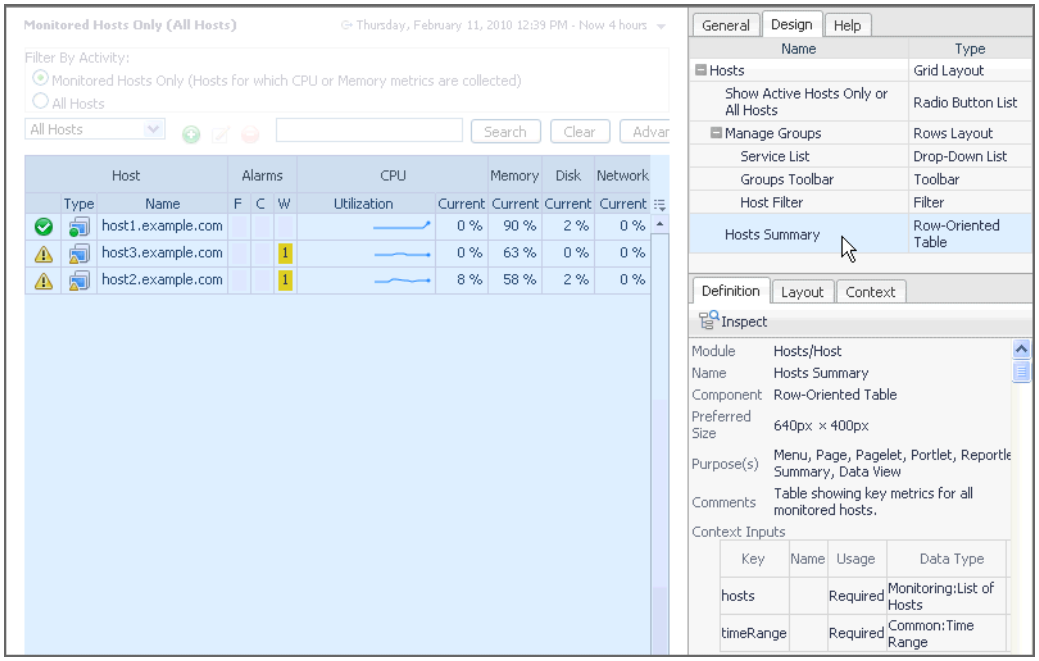
Design tab is available on any non-portal page to properly-authenticated users such as those with the dashboard designer role. The Design tab shows a hierarchical list of all views of a dashboard, definition details, and a context editor that is useful for debugging.

The top pane of the Design tab displays the name of the dashboard currently being viewed, and the names of all its component views. The items in the pane respond to the mouse pointer, As you hover over a name, the component is highlighted. This facilitates finding a component that is contained in another component.

When you select a particular view you can:

- See the definition of the current view.
- Click Inspect to analyze the view in the definitions editor.
- View the current run-time values that are available in the context for that view.

Figure 2. Design tab



In the above image, the Row-Oriented Table contained in the Grid Layout is selected. The portion of the grid that contains the table is highlighted, and Definition tab displays the same information that is found in Configuration > Definitions for that component. There is an Inspect button directly underneath the Definition tab. If it is selected the Definitions area opens, permitting you to edit the component's properties.

The Layout tab displays layout information for the contained component, if there is any.

The Context tab displays the context keys for the chosen component and shows their current values.

The Web Component Framework

The Foglight® user interface is built using the Web Component Framework (WCF). This same framework is available to you to develop your own specialized views. Existing Foglight views are configurable from the user interface without the need to resort to the full editing resources provided by WCF, but if you decide to build your own specialized dashboards, the Web Component Framework is available for your use.

- Core Concepts
- Using the Web Component Framework
- Managing Dashboards

Core Concepts

There are certain core concepts that you will encounter while using WCF:

- Modules
- Observations
- Context
- Parameters in Bindings
- On Null Values
- Renderers
- Default Values
- Data Sources, Data Types, and Data Objects
- Paths

Modules

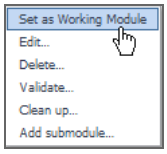
The default WCF definitions (views, queries, renderers, tasks, icons, files, types, and units) in Foglight are organized into the modules and sub-modules seen in the module list pane of the Definitions editor. A module contains a collection of related definitions. If you have the Dashboard Designer role, you can add entities to any of the existing modules. If you have access to the Definitions choice (in the navigation panel under **Dashboards** > **Configuration**), you can create entities in their own modules. You can then save these modules by exporting them to a file using *fglcmd*. Again using *fglcmd*, you can import modules that have been created on another Foglight® server.

For information on importing and exporting modules, see the *Command-Line Reference Guide*.

The Edit Module popups

Click the triangle icon ( ) at the right of a system module name and a menu appears.

Figure 3. Edit module popups



The choices are:

- **Edit**. Displays the **Edit Module** dialog box that alloes you to change the following module properties:
  - **Name**: Display name of the module
  - **Description**: Text that appears as a tooltip when hovering over the module name
  - **Logo**: A logo at the left of the screen for dashboards in this module
  - **Alternate Text**: Alternate text for the logo
  - **Relevant Roles**, **Allowed Roles**: For more information, see Relevant Roles and Allowed Roles on page 67.
  - **Parent**: Allows the reassignment of the sub-module so that it appears under a different module from the one in which it appears by default.
  - **Main View**: The view that is to be displayed by selecting the module in the Dashboards tree. By specifying a Main View, the module name itself becomes active. You can use this functionality to provide a shortcut to the most frequently chosen dashboard in the module.
- **Delete**: Deletes all the view components in the module. You are warned that this operation cannot be undone.
- **Validate**: Runs tests on the module to check such things as unused context entries. You can choose to run validity checks on all the module's components, or just a selected set. For instance you could choose just the queries and the functions for testing. A report shows the result of the validity check.
- **Set as Working Module**: Sets the module as the working module. This module becomes the active module when you navigate to the Definitions area.
- **Cleanup**: Removes unused resources from the module.
- **Add submodule:** Creates a sub-module entry so that it appears under the current module

Validating Modules

You can validate a system module and check all the definitions within it for errors or warnings. Click the small triangle at the right of the module's name in the module definition pane and choose **Validate**. This is useful for catching errors that might have crept in through hand coding or prior incompatibilities in the WCF tooling that allowed poorly-configured views to be saved. This functionality can be invoked from the module's menu.

There is a **Clean up** option that can be used to remove unused resources.

The other choices in the popup shown by clicking the small triangle at the right of the module's name are **Edit**, **Delete**, and **Add Submodule**. Clicking **Edit** launches a dialog box that allows you to assign relevant and allowed roles to the module, which lets you control its visibility and its access rights.

Definitions and Entities

Definitions are the configurable units in Web Component Framework. They include Views, Queries, Functions, Renderers, Tasks, Type Mappings, Type Mappings and Theme and Module Resources.

Entities are the types of definitions that can be referred to by an ID identifier in other definitions. Views, Queries, Renderers, Tasks, Type Mappings are entities.

NOTE: Views and their properties are described in the *Web Component Reference*.

Public Entities

Marking an Entity as public indicates that it can be used by any definition in any other module. For this reason the editors do not allow you to delete a public entity. However, you have the option of replacing a view containing a public entity.

Entities that are not public can only be referenced by definitions that have a shared module ancestry. Definitions have a shared module ancestor if they have a parent, grandparent, and so on, module that is the same or one is an ancestor of the other.

Only public entities are shown in modules that do not have a common ancestry with the module containing the current definition. Keep this in mind when attempting to select an entity, especially if you think one should be there and it isn't.

Copying Entities

When copying entities, whether entities are public or not needs to be taken into consideration.

You can make a shallow copy of an entity that is not public. However, if that entity refers to a non-public entity, that is, refers to a query via a Query Binding, then the copy will not be allowed if that non-public entity is not accessible from the target module to which you want to save the copy.

The View's *deep copy* feature can be used to copy a view that references a non-public entity that is not accessible from the target module. However, a warning message will be issued that the inaccessible, non-public entity will be copied and the reference will be changed to refer to the copy. This may not be what you want. You may actually want the references to be maintained because you intend to deep copy a view to a temporary module, make the desired changes to the view and its related entities, and then deep copy the view back to its original module.

You can perform either a copy of the view or a deep copy of the view without having to use the Add View work flow to invoke the copy. Moreover, if you wish to do a simple copy, but are not permitted to because of private reference restrictions, then a deep copy can optionally be invoked to minimize the duplication required to copy the specific view.

Moving Entities

Entities that are not public may be moved to another module that has a common ancestry. Movement is restricted to such modules because the entity may have a reference to another non-public entity and thus only if it is moved to a module with common ancestry would that reference remain allowable. The tree of target modules that an entity may be moved to reflects this restriction. The only modules that are selectable in the tree are those to which the entity may be moved.

Unit Testing Entities

Function and Query Entities can be unit tested. When you select a Function or a Query in the Definitions Editor, a Unit Tests toolbar button is available. When you click that button, a dialog box appears, allowing you to create, edit, remove, and run unit tests.

NOTE: A dashboard is available in the Dashboard Support module that allows you to run multiple unit tests.

Writing unit tests

Unit tests have three attributes:

- **name**: The name of the test. Must be unique per function/query.
- **timeout**: The maximum time (in seconds) that the unit test is allowed to run. If the test exceeds the maximum, then the run will be stopped automatically
- **script**: The script of the unit test.

Unit test scripts have a `TestHelper` class available to them. It contains useful methods for unit tests.

For functions, the `testHelper` is an instance of `FunctionUnitTestHelper`. For queries, the `testHelper` is an instance of `QueryUnitTestHelper`.

The only difference between these two helpers is that the `invoke(Object... arguments)` method has different return types and throws different exceptions.

Noteworthy test helper methods are listed in the following table.

Table 1. Test helper methods

Method	Description
`invoke(Object... arguments)`	Use this method to invoke the function/query being unit tested. It will take into account the current `SpecificTimeRange`.
`setSpecificTimeRange(SpecificTimeRange)`	Use this method to set the current `SpecificTimeRange`. If a helper method is subsequently called which needs a `SpecificTimeRange`, a `TimeRange` or a timestamp, one will be used that reflects the current `SpecificTimeRange`.
Assertion methods	There are a number of JUnit-like assertion methods for checking conditions in your unit test. For more information, see the `FunctionUnitTestHelper` and `QueryUnitTestHelper` API documentation.
`checkIfCancelled()`	If your unit test can potentially take a long time to run, use this method to check if the test has been cancelled (either by a manual stop or because it has exceeded the TestCase's timeout).

Observations

In general, the Web Component Framework and Foglight® are concerned with the collection of information over time. These collections are called observations. Observations can be of any kind of object. Observations that are handled specially by the Web Component Framework are Metrics and Enum Observations. The scope and quality of the information returned by an observation in the Web Component Framework is determined by the time range used to retrieve the information.

Time Range Related to Observations

A TimeRange for a Metric Observation can be specified in various ways, but it is always ultimately composed of a range of date-time objects, and a granularity. The granularity can be RAW, which means that data observations should be shown in the metric history with the smallest available granularity, or a number of milliseconds, for example, 300,000 for 5 minutes. It can also be AUTO, for example, a numeric value of -2, meaning the code will pick the best granularity based on the time range.

The list of history Metric Values must be in ascending order of date-time.

When the granularity is RAW, indicated by the numeric value -1, history observations will be added with the smallest available granularity. Different observations within the history may cover differently-sized time intervals, and different metrics in the same time range using RAW granularity may have different numbers of history observations in the case of unrelated time intervals.

When the granularity is a number of milliseconds, in each history metric value, the difference between the end time and the start time (in milliseconds) must equal the granularity. Each successive history object's start time should be the previous history object's end time. That is, each history object must cover exactly one granularity interval.

Exactly how the start times and end times of the history objects are related to the start time and end time of the entire non-RAW time range is not specified as a *Web Component Framework* requirement. For one thing, the length of the entire time range may not be an even multiple of the granularity. Reasonable options include starting the first history object's start time at the exact start time of the time range, or starting it at the start time of the time range less half the granularity. However, the start time of the first history object should be no greater than the start time of the time range, and the end time of the last history object should be no less than the end time of the time range.

When there are multiple metrics calculated for the same non-RAW time range (whether they are different metrics on the same object, or the same metric on different objects), they must have exactly the same number of history objects as each other, and the start times and end times of the history objects at the same indices in their lists must be identical.

If there are no agent observations that correspond to a history object within a certain granularity interval in a non-RAW time range, a history object is still created, with the appropriate start time and end time, but none of the value properties are filled in.

Metric Observations

An observation of a numerical value is called a metric.

Data collected by Foglight® agents is generally expected to be packaged into a Metric data object. Certain components, such as the chart components, must be bound to Metric objects to function.

Metric definitions must specify the *unit* property in the Metric's containing Property class. The unit is used to display numeric data. In particular, the unit's *precision* property is used by some components (such as charts) and renderers to ensure the correct number of decimal places are used when rendering data. If the Metric's precision is desired to be different than its unit's precision, you can set the precision directly on the Metric's containing Property class using the *precision* property.

To view the current, latest, or history values for a topology object:

1. From the navigation panel, under **Dashboards, click Configuration** > **Data**.
2. In the Data dashboard, drill-down to the topology object to view the metric values.
3. Click on the metric to view more details about the topology objects in the Property Viewer. Each value entry for an observation has a **Start time**, **End time**, and **Value**.

Metrics must conform to the following schema:

Metric

Table 2. Metric properties

Property	Type	Description
uniqueId	`String`	Unique ID.
name	`String`	Name of the metric, localizable.
period	`MetricValue`	Value for entire time range.
latest	`MetricValue`	Value of the last recorded sample.
current	`MetricValue`	Value for last interval in time range.
history	`MetricValue; is-many = true`	List of values for each interval in the time range.

MetricValue

Table 3. MetricValue properties

Property	Type	Description
uniqueId	`String`	Unique ID.
startTime	`Date`	Start time of the interval, exclusive.
endTime	`Date`	End time of the interval, inclusive.
sampledPeriod	`Long`	Period covered by this interval in milliseconds. (This value may be lower than the difference between `endTime` and `startTime`, if the agent does not collect data during the entire interval.)
count	`Long`	Number of agent samples in this interval.
min	`Number`	Minimum value of all agent samples in this interval.
max	`Number`	Maximum value of all agent samples in this interval.
average	`Number`	Average value of all agent samples in this interval.

Table 3. MetricValue properties

Property	Type	Description
sum	`Number`	Sum of all agent samples in this interval.
sumSquares	`Number`	Sum of the squares of all agent samples in this interval.
standardDeviation	`Number`	Standard deviation of all agent samples in this interval.
baselineMin	`Number`	Baseline minimum value of all agent samples in this interval.
baselineMax	`Number`	Baseline maximum value of all agent samples in this interval.
thresholds	`ThresholdValue is-many = true`	List of thresholds for this interval.

ThresholdValue

Table 4. ThresholdValue properties

Property	Type	Description
uniqueId	`String`	Unique ID
upperBound	`Bound`	Upper bound of threshold range
lowerBound	`Bound`	Lower bound of threshold range
state	`Enum`	State associated with this threshold range

Bound

Table 5. Bound properties

Property	Type	Description
uniqueId	`String`	Unique name.
value	`Number`	Value of *bound*. If set to *Double.POSITIVE_INFINITY*, the threshold is assumed to stretch to infinity.
exclusive	`Boolean`	`False` if the bound includes value, `True` if it does not.

Enum Observations

Data collected by agents is sometimes collected into an *EnumObservation* data object. Certain components must be bound to *EnumObservation* objects to function. *EnumObservations* must conform to the following schema:

EnumObservation

Table 6. EnumObservation properties

Property	Type	Description
uniqueId	`String`	Unique name
name	`String`	Name of metric, localizable
period	`EnumObservationValue`	Value for entire time range
current	`EnumObservationValue`	Value for last interval in time range
history	`EnumObservationValue; is-many = true`	List of values for each interval in time range

EnumObservationValue

Table 7. EnumObservationValue properties

Property	Type	Description
uniqueId	`String`	Unique name
startTime	`Date`	Start time of interval, exclusive
endTime	`Date`	End time of interval, inclusive
index	`Number`	Index of this observation
value	`Number`	Enum value of this observation

Context

Context is the collection of data that defines the essential dynamic information that the view requires. If a view is supposed to display information about one application server, then the context specifies which application server.

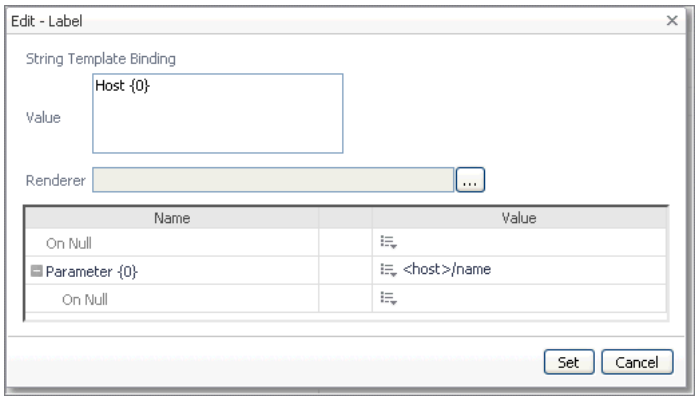
If the context of a view has been set, then the view can use that value by specifying the named value of the context via a key. Similarly, if a view has a given named value in its context, and a link takes it to another view, then that new view can get access to that value by specifying its name as a context input.

For more information, see Context Tab on page 51.

Parameters in Bindings

Most types of bindings have parameters. These are placeholders nested within the binding that are also evaluated at run time. An example is a String Template binding set to *Host: {0}*. In this case *Host:* is a fixed string, while *{0}* is a reference to a parameter that evaluates to the name of the host server by configuring that association to a particular dynamic context at the time when the String Template binding is being defined. Each description of the binding type explains how to use parameters.

Figure 4. Binding parameters



CAUTION: Any quotation mark you use in the parameterized string must be escaped. For example, if the text string that you want to use is The host's name is: {0}, you must escape the apostrophe using the back slash character '\' as follows: The host\'s name is: {0}.If the text string includes a back slash character '\', use double back slashes, "\\" to represent the back slash in the string.

For more information, see Bindings on page 80.

On Null Values

Most types of binding have an associated On Null binding. This provides an alternate value that can be used if the main binding evaluates to null (nothing) or an empty list of values.

Renderers

A renderer may be specified on each type of binding. A renderer determines how the evaluated value is displayed. For example, a limit to the length of a string, the number of decimal places, or the date and time format. If no renderer is specified, then a default renderer based on the object's type is used. Therefore, in most cases, an explicitly set renderer is not required.

Null and error renderers, noted earlier in the list of simple types, are special cases of renderers. They are only used to display no data or error information. Unlike normal renderers, they do not base their behavior on the data being rendered, but instead have fixed outputs. A drop-down menu lists all the available renderers. Generally, you can determine which renderer to select based on its name. If no renderer is specified, renderers are looked up from the type, property and unit of the value. If no renderer is found, a default renderer is used. Therefore, in most cases, a renderer is not required.

For more information, see Renderers on page 106.

Default Values

If a property has a default value it is displayed as text beside the edit icon in the Value column of the tree table in the Configuration tab. A property that has been configured to use a specific value can be set back to its default property by clicking the **Edit** icon, then selecting **Clear to default** from the drop-down menu.

Data Sources, Data Types, and Data Objects

The data that is displayed by the application comes from a data source, as set up in Foglight. The data from a data source is held in data objects as properties. For example, some of the Host data-object properties are the name of the host, its IP address, and the number of fatal events. The data type is a data-object template, and determines the structure of a data object. Examples of data types are Host, AppServer, WebSphere®, Agent, and Event. For more information about type properties, see the Schema Browser. This dashboard is accessible from the Tools and Dashboard Support dashboard.

TIP: The Cartridge Development role grants access to this dashboard.

Data Sources

Data sources encapsulate all that the system knows about the data and yet cleanly separates knowledge of the data from how it is presented.

The data source is organized as a dynamic graph of objects, starting from a root that represents the entire data model.

"Objects" are defined in the API and are not tied to the creation of any particular Java™ Object.

The Data node is populated from queries that are marked as UI Query when they are defined in the Definitions editor.

Data Source Queries

- The syntax of a data source query resembles SQL, but queries return a list of objects.
- Data source queries are strongly typed.

- Data source queries are not free form. In Foglight, they are configured on the browser interface's Definitions page by using an editor provided for that purpose.

Paths

When creating queries and defining bindings, you set the values by specifying the data object and the properties in the Path field. Paths traverse the structure of the data object. They are similar to directory paths in Windows® or UNIX®, and are comprised of a series of one or more property names, separated by forward slashes.

NOTE: You can replace a long path with a key by setting it as a context value.

One minor complication is that property names are often displayed with localized names instead of their actual property names. For example, a drop-down tree may show the properties as Name or CPU Usage, but when selected they display in the Path field as *name* or *cpuUsage*.

The following are some example paths:

Table 8. Paths

Path	Meaning
/hosts	An absolute path for all hosts, under the top of the tree of data.
cpuUsage	The cpuUsage metric object (under a host object)
cpus/utilization/current/average	The current average value of the cpuUsage metric object (under a host object)

Properties

You edit the properties of a view in the Configuration tab. If **Show Advanced Properties** is false, only the ones needed to get you started are shown. If **Show Advanced Properties** is true, all the component's properties become available for editing. Required properties are shown in bold font.

Properties that have been changed from their defaults are highlighted. Properties that are not set are dimmed. In edit mode, dwelling over a property shows a tooltip that contains the same description as that in the Web Component Reference page for the component. In display mode, only the properties that have been changed from their default values are shown.

Using the Web Component Framework

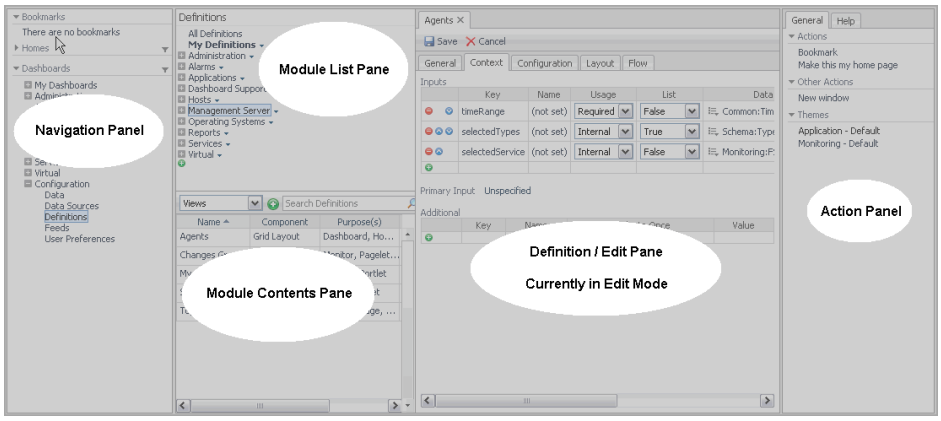
Before you start working with the Web Component Framework you need to define what objectives you are trying to accomplish. For example, if you want to monitor a specific application server at certain intervals and send alerts to a specific email address, you need to choose the components necessary to create this type of workflow.

If you have already created a query that retrieves what you want from existing data, you can start creating views to display the data. If you do not have the proper data to work with, you need to create queries to return the data. For more information about queries, see Queries on page 57.

The Web Component Framework Editor

You navigate to the editor by clicking **Configuration > Definitions** in the navigation panel. Both the navigation and action panels are described in the Foglight® *User Guide*.

Figure 5. Web Component Framework editor



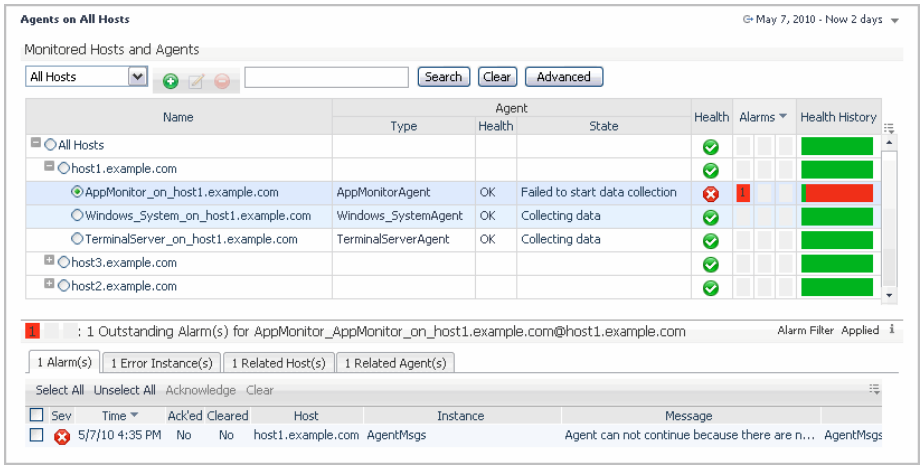
The Web Component Framework editor consists of the following panes:

- **Module List**: This pane displays the existing modules. Most modules are defined by the application developer. You can add your views to the module called My Definitions, which is associated with your login name, although you can add your views anywhere you choose (if you have the proper permissions). Other users can add views as well. If such a user was creating views in his or her My Definitions workspace, you would locate these views under Other User Definitions, where the modules are arranged by login name.
- **Module Contents**: This pane consists of nine tabs. Within each tab you can create the following entities for the selected module: views, queries, functions, renderers, tasks, icons, files, types, and units.
- **Definitions/Edit**: It functions as an editor pane, and as an item definition pane once the item has been configured. You use this pane to edit a new or existing definition. Once the definition is saved the pane switches to a view of the definition showing all the properties that have been changed from their default values.

In Foglight, you access these panes from **Dashboards** > **Configuration** > **Definitions** in the navigation panel.

An Example Page

Figure 6. Sample page



A page may be:

- A context-free dashboard
- A dependent page, which optionally requires a context
- A two-pane browser: navigator and page

All of the preceding are assembled from view components.

The page may contain:

- A variable number of view components
- Page decorations, such as a time range control
- Tabs or a splitter, depending on the container being used
- A customizer, which is one way of linking to other pages

Views can be configured using any of the following mechanisms:

- **Context**: What is shown depends on the context passed to the page or component
- **Queries**: Queries provide data binding
- **Flows**: Action to be performed based on user input, such as a drill down page

Other data binding choices exist. For more information, see Bindings on page 80.

View components and their containers have properties that are described in the *Web Component Reference*. Refer to it for details about these properties.

Managing Dashboards

The following section describes the Dashboard, Definitions, and Data Sources pages.

Definitions Panes

From the Module List pane, accessed in the navigation panel under **Configuration > Definitions**, you can examine the dashboards that have been defined for all the System and User modules to which you have access. After familiarizing yourself with the available dashboards, you can decide whether a custom one needs to be built, or you can perhaps copy and modify an existing one, instead of creating a new dashboard from scratch.

System and User Dashboards

In Foglight®, there are two types of dashboards: system and user. System dashboards are preconfigured and delivered with Foglight. User dashboards are created or modified by users. Depending on your permissions, System dashboards may or may not be displayed in the Dashboards page that you see.

Foglight comes with preconfigured views, queries, renderers, tasks, icons, files, types, and units. Users can view and copy these definitions, and with the proper permissions they can change them as well. Alternatively, you can create and modify your own User modules. Similar to System modules, you can view and copy modules created by another user but you cannot modify those definitions.

As mentioned previously, you can copy and make modifications to an existing view, or you can create an entirely new view. The choice depends upon how similar to an existing one the new page must be. These remarks apply to queries as well. It is better to copy and modify a complex query rather than recreating it if only a few changes are required.

See the *Web Component Tutorial* for instructions on creating new views.

Dashboard Page Navigation

You can select items in the views that refresh the page (drop-down menus or radio buttons) or click links to display pages with more detail. A list of links to previous pages (breadcrumbs) is located at the top of the page. The current page is highlighted in bold. At any time, you can move back using the breadcrumbs.

You navigate around Foglight® dashboards from summary dashboards down to detailed pages (drill-down) and back up the hierarchy. Navigation is controlled through menus, links, drop-down lists, and option buttons.

CAUTION: The pages display dynamic data. Therefore, you should use the links, and not the built-in navigation controls of your browser. Using the links refreshes the data on the page.

You can place views on the navigation panel. The contents of the view should be related in some way to navigation. For instance, you might provide a list of items, and each item would load a corresponding view in the console's main view area. When a page adds a view to the navigation panel it is persisted. Clicking an item in the list reloads the parent page corresponding to the item.

You can place views on the action panel as well. A customizer can be placed here, as well as other views that provide additional information and a legend for the page in the console's main view area.

Data and Data Sources Pages

The *Data* node is only visible to users with administrative privileges in Foglight®. On the left side of the page is a navigation tree. Each node represents a Root Query that has been configured to return a data object hierarchy. You can add nodes by configuring a new query and marking it as a UI Query.

The Data page displays some of the available data objects that have been instantiated on the currently running monitored system for which views have been created and for which root queries have been defined. See Queries on page 57 for more information about the query mechanism.

The Data Sources tab allows you to add or remove multiple data sources. Using the query mechanism, WCF components look for data in the data source to populate the views.

Definitions Pane

The *Definitions* pane is your workspace for creating, viewing, and editing items in the *Web Component Framework*.

NOTE: If a user is not granted to the Cartridge Developer role, they cannot see the Data or Definitions dashboards.

To open a definition:

NOTE: Views may be opened for editing from the Design tab in the action panel.

1. Open **Dashboards** > **Configuration** >**Definitions** in the navigation panel.
2. You can use the Definitions pane to view or edit these components after you have selected them in the Module Contents pane. In the Definitions menu choice, the Module List pane displays the available modules, while the Module Contents pane displays the contents of the selected module. It contains a set of nine tabs, each listing a particular grouping, such as views, queries, or renderers.
3. In the Module Contents pane, open the item whose definition you want to view by selecting it. Alternatively you can create a new view or query. See Configuring Views on page 27 for information about creating a new view or see Creating a Query in Foglight on page 58 for information about creating a new query.

   The various configuration options are:

Table 9. Configuration options in a definition

Option	Description
Views	Views are the Web Component Framework objects that display data. The components that are shown in the Module List pane are those that have been defined for the node you have selected in the Module Contents pane. See the *Web Component Reference* online help pages.
Queries	Queries allow you to select a set of data objects of the same type. Pages and views extract and use data from the queries for values and context settings. See Queries on page 57.
Functions	Function definitions allow you to return data of a declared output type given an expected input.
Files	Upload files such as images to the module's public directory. See Files on page 94.
Icons	Each icon can be a collection of images of different size. See Type Mappings on page 108.
Associations	Associations provide a mechanism to organize references into entities. See Associations on page 95.
Renderers	Renderers determine how values is displayed. Various renderer types are available for customization. See Renderers on page 20.
Type Mappings	Type mappings associate entities such as renderers, icons and flows to a specific data type or data type property. At run-time, the Web Component Framework will look up the entities mapped to these types and properties to use as defaults if more specific entities have not been specified. The name of the mapping is a combination of the data source name and the data type name. See Type Mappings on page 108.
Unit Mappings	Maps the unit of measurement to a data source and a renderer. The name of the mapping is a combination of the data source name and the unit name. See Theme and Module Resources on page 114.
Tasks	Tasks are applications external to the *Web Component Framework* that can be launched from within a view. See Tasks on page 109.

4 When a definition opens for one of the items in the module list, view and reconfigure the settings, as required.

Alternatively, create a new definition and configure it using the definition tabs, as required. For more information see, Definitions Pane Settings Tabs on page 37.

Customizing the UI Quickly

See the Foglight® *User Guide* for information about the *Create dashboard* functionality.

Finding Pages: Bookmarks

See the Foglight® *User Guide* for information on adding *Bookmarks*.

Additional Documentation

For more information about the *Web Component Framework* see the following documentation:

- *Web Component Tutorial*: This document walks you through examples of how to configure some simple views.

- *Web Component Reference*: Online help pages. These pages contain a complete list of properties for each item, object, or component in the framework.
- Foglight® *User Guide*: This document describes the default dashboards and how to create dashboards and reports from existing views. For more information about individual components, see the *Component Reference Guide* in the online help.
- *Command-Line Reference Guide*: This document contains information on scripting Foglight using DOS command or UNIX® shell scripts.

3

Configuring Views and Context

For a user interface component to be useful it must display relevant information. In most cases this information is not static, but is supplied by agents collecting data on an ongoing basis. Thus, there are two major requirements for a user interface framework—a rich set of visual components and a data binding mechanism. Queries are useful for retrieving data from a data source, but context is the usual mechanism for sharing the data among related views. This chapter discusses the tools for building visual components and describes how context is used to pass data among the various views that have a need for that data.

- Configuring Views
- Context Tab

Configuring Views

If you have a data source you can start creating views. Views are collections of components that display data. The components of a view are self-contained, so you can add them to a page, remove them, or move them around according to your needs.

The following concepts apply to views:

- Containers hold the view components that present your data. A container view is comprised of multiple components that are organized by related content.
- Views are added to a container through the use of the Module Definitions editor.
- View components are the underlying configurable building blocks on which views are based. Each view component has both unique and common properties. For example, a table has a column property, a chart has a time-axis property, and both have a customizer property.

The view component groups available in Foglight® are summarized in this guide. For more information, see Anatomy of a Typical Dashboard on page 10.

Creating a New Container View

A container view is used to house data presentation views, such as tables, charts, and common UI components (for example, check boxes or labels). You can declare a container view to be a dashboard and use it to observe system performance at run-time. For more information, see Anatomy of a Typical Dashboard on page 10.

The choice of container depends on the components it is designed to contain and on the layout you want.

You can create a new container, but you cannot add views unless you have defined them previously, so it is advisable to start by choosing the view components you need based on the data you want to present, and then decide on the container that is the best choice to present these views.

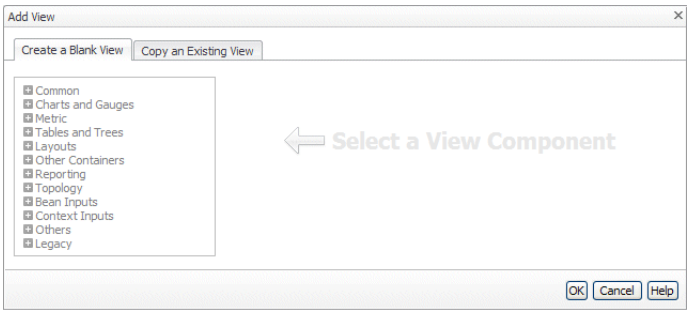
To create a container view:

1. From the navigation panel under Dashboards, click **Configuration > Definitions**.
2. In the Module List pane, choose the module in which you want to work. If you are creating views for your own use, choose **My Definitions**.
3. From the Module Contents pane, click **Add**.

4 In the **Add View** dialog box, ensure that the **Create a Blank View** tab is selected.

5 Select a view component from the tree of view types.

Figure 7. Selecting a view component

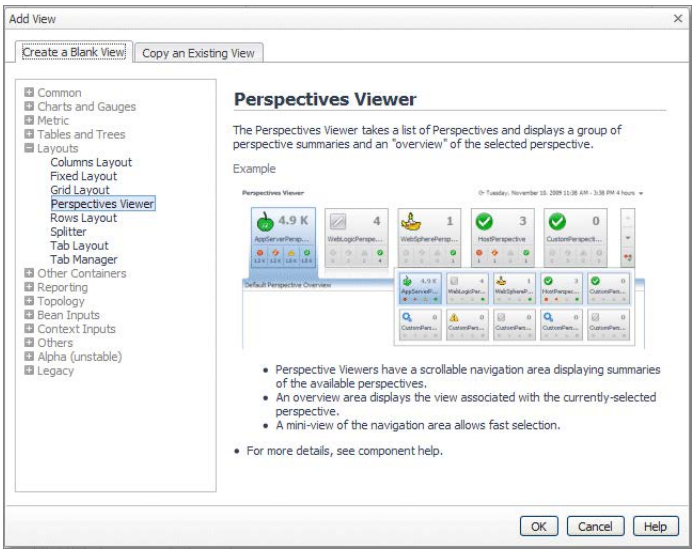


6 From the navigation tree, select **Layouts** and click ⊕ to expand the list.

7 Select the view.

The list box refreshes to show the newly-selected view type:

Figure 8. Selected view type



8 To close the **Add View** dialog box, click the **OK** button.

9 The new, unnamed view appears in the editor pane.

10 Configure the container by filling in the required fields in the editor pane. The fields for any particular view are described in the *Web Component Reference* help pages. The *Web Component Tutorial* contains an introductory example of creating a container and various views for a dashboard.

Creating a New View Based on a Copy of a View

Copying is a fast way of creating a new view. It is also a way to create a modified version of a *System* view. You can copy any view, including your own *User* view, a *System* view, or a view created by another user.

You cannot create a copy of a view and give it the same name as another view in your module. If you enter a name that is already in use, a tooltip message is displayed beside the field and the **Save** button is disabled.

You can perform a normal or a deep copy of a view from the Definitions pane. For more information, see Deep Copying Views on page 35.

To create a new view based on a copy of an existing view:

1 From the navigation panel under **Dashboards**, click **Configuration > Definitions**.

2 In the Module List pane, choose the module in which you want to work. If you are creating views for your own use, choose **My Definitions**.

3 From the Module Contents pane, click **Add**.

4 In the **Add View** dialog box, ensure that the **Copy an Existing View** tab is selected.

5 From the navigation tree, select a view type and click the ⊞ to expand the list.

6 Select the view.

The list box refreshes to show the newly-selected view type.

7 Click **OK.**

The Module Definitions pane switches to a new view in edit mode. The various fields contain the settings based on the copied component.

8 Make the changes you want, including renaming the component, and click Save.

The component is saved in the module that was active when you clicked **Add**, which is normally your user space under Foglight.

Stamping Views

A collection of views can have their properties updated at the same time using the Stamp Views feature. Use it to quickly perform a batch update of view properties.

To stamp views:

1 From the navigation panel, under **Dashboards**, click **Configuration > Definitions**.

2 In the Module List pane, select the module containing the views whose properties you want to edit.

3 In the Module Contents pane, issue a search for views using some common criteria.

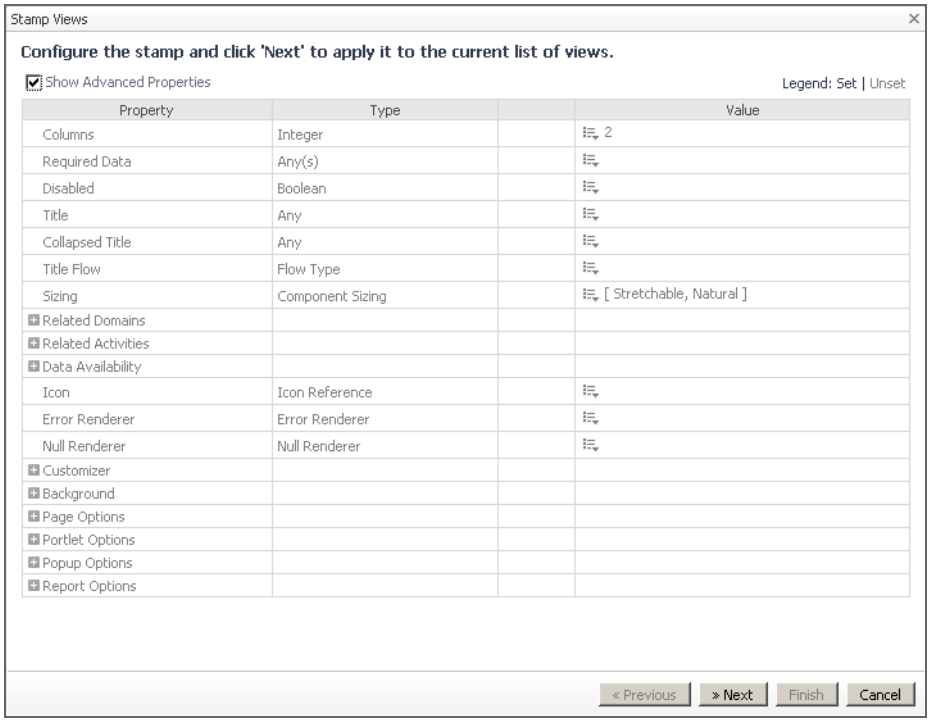
The list of views in the Module Contents pane refreshes, showing the views that satisfy the search criteria.

TIP: The Stamp Views button becomes enabled, indicating that the views listed in the Module Contents pane can be selected for stamping.

4 In the Module Contents pane, click .

The **Stamp Views** wizard appears, showing a list of properties common to all views that are currently selected for stamping.

Figure 9. Stamp Views wizard



TIP: Select or clear the **Show Advanced Properties** check box to switch between basic and advanced view properties, as needed.

5 Edit one or more properties, as required.

a To edit a property, in the **Stamp Views** wizard, click in the **Value** column, and proceed with making the edits, as required.

> **IMPORTANT:** View stamps cannot be applied to dynamic context keys. This is because view stamps are meant to be used for dynamically changed data, and as such do not need to refer to dynamic context.

> **NOTE:** If a stamp results in changing view columns, the column property changes are propagated to all of the views that are selected for stamping, replacing the existing columns. For example, if view A has two columns and view B has three columns, and if the stamp has one column, instead of applying that one column to each of the views columns, both views result in having one column each, as set by the stamp.

> **TIP:** If you have any properties set, you can clear their values by selecting Clear if set from the menu.

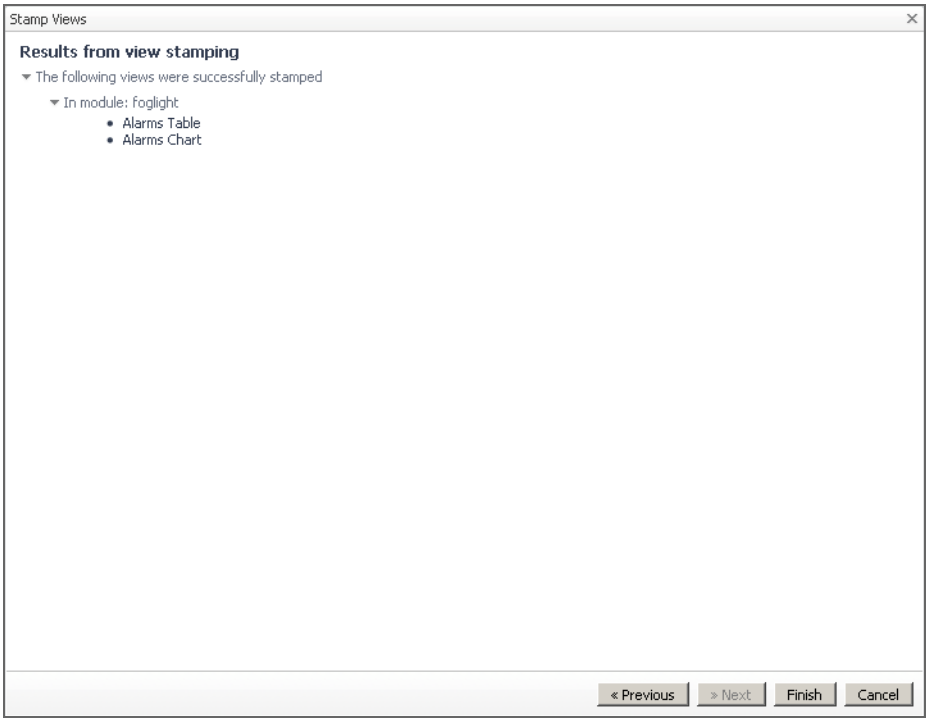
b Click **Next**.

The views that are successfully stamped are validated. If the validation results in any errors, the property changes are not saved to the related views. However, if a stamped view generates a validation warning, the changes are saved.

At this point the property change cannot be reverted by cancelling the flow in the wizard. This is because while checking whether a stamp is applied successfully, the module the views belong to needs to be stored. When the store is complete, the module can not be reverted to its previous state.

The **Stamp Views** wizard refreshes, showing the outcome of the view stamping process. It lists the views that are successfully stamped and validated, and any errors, if applicable.

Figure 10. Stamp Views results



6 Click **Finish**.

The **Stamp Views** dialog box closes.

Searching for Definitions

The Module Contents pane on the Definitions dashboard shows the dashboards that you can access based on your user permissions. Every dashboard can include one or more preconfigured views, queries, functions, files, icons, associations, renderers, types, type mappings, unit mappings, and tasks. The search feature allows you to search for various dashboard definitions contained in one or more selected modules, such as functions, queries, renderers, tasks, types, and views.

Selecting **All Definitions** in the Module List pane allows you to search for components across all modules and all definition types. Selecting a specific module limits the component search to that module and any of its child modules, if they exist.

There are two modes for searching: by typing the desired text into the search box (simple mode), or by specifying one or more search rules (advanced mode).

In simple mode, the following definition elements are searched:

- Definition names and IDs
- Function return types
- Query return types
- Renderer components
- Task components
- Super types of a type
- View components

In advanced mode, you specify one or more search rules, and indicate if you want one or all rules to return a match. The following elements can be searched for in a rule:

- Definition names and fully-qualified IDs
- Module names or IDs
- Definition components (views, renderers and tasks only)
- Definition type-specific text, including:
  - Function return types
  - Query return types
  - Renderers (value data source type, value data type)
  - Super types of a type
  - View purposes, custom purposes, relevant roles, and allowed roles
- Comments (excludes file components) and descriptions (types only)
- Definition's context help (excludes types and files)
- Last modified time (excludes types and files)
- Public fields (excludes types, files, type mappings, and unit mappings)
- Deprecated fields (excludes types, files, type mappings, and unit mappings)

TIP: The User Preferences setting, **Show deprecated definitions**, indicates if deprecated entities are considered in a search. However, if a rule in an advanced search uses the **Deprecated** field, the **Show deprecated definitions** setting is ignored, and the search results include definitions that match the rule with the **Deprecated** field.

To issue a simple search:

1 From the navigation panel, under **Dashboards**, click **Configuration > Definitions**.

2 In the Module List pane, select the module containing the definitions you want to search.

TIP: To search through all definitions, select **All Definitions**.

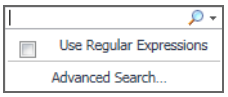
3 If you selected a specific module, in the Module Contents pane, select the definition type that you want to search. Click **Views** and select the desired definition type. For example, to look for a specific query, select **Queries** in the list.

4 **Optional**—Use a regular expression to issue a search.

a In the Module Contents pane, in the **Search Definitions** box, click .

A menu appears.

Figure 11. Search menu



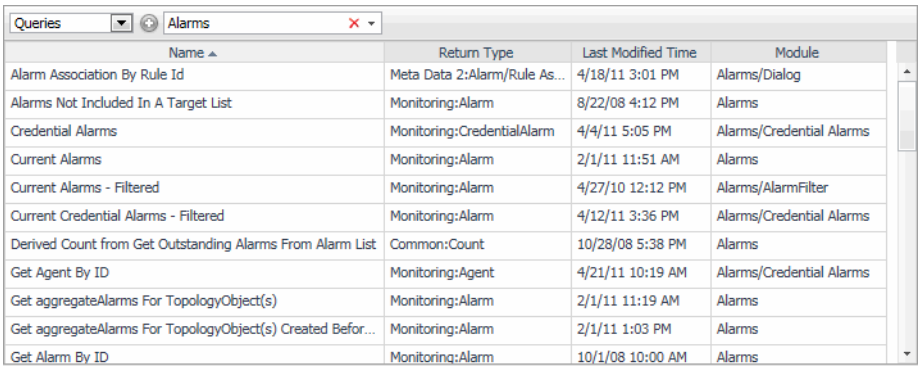
b Select **Use Regular Expressions** in the menu.

5 In the Module Contents pane, type the text pattern in the search box.

6 Click .

The Module Contents pane refreshes, showing a list of definitions that match the text pattern.

Figure 12. Definitions matching a specified text pattern



Name	Return Type	Last Modified Time	Module
Alarm Association By Rule Id	Meta Data 2:Alarm/Rule As...	4/18/11 3:01 PM	Alarms/Dialog
Alarms Not Included In A Target List	Monitoring:Alarm	8/22/08 4:12 PM	Alarms
Credential Alarms	Monitoring:CredentialAlarm	4/4/11 5:05 PM	Alarms/Credential Alarms
Current Alarms	Monitoring:Alarm	2/1/11 11:51 AM	Alarms
Current Alarms - Filtered	Monitoring:Alarm	4/27/10 12:12 PM	Alarms/AlarmFilter
Current Credential Alarms - Filtered	Monitoring:Alarm	4/12/11 3:36 PM	Alarms/Credential Alarms
Derived Count from Get Outstanding Alarms From Alarm List	Common:Count	10/28/08 5:38 PM	Alarms
Get Agent By ID	Monitoring:Agent	4/21/11 10:19 AM	Alarms/Credential Alarms
Get aggregateAlarms For TopologyObject(s)	Monitoring:Alarm	2/1/11 11:19 AM	Alarms
Get aggregateAlarms For TopologyObject(s) Created Befor...	Monitoring:Alarm	2/1/11 1:03 PM	Alarms
Get Alarm By ID	Monitoring:Alarm	10/1/08 10:00 AM	Alarms

7 To clear the search results, click .

The Module Contents pane refreshes, showing a list of all definitions of the selected type.

To issue an advanced search:

1 From the navigation panel, under **Dashboards**, click **Configuration > Definitions**.

2 In the Module List pane, select the module containing the definitions you want to search.

TIP: To search through all definitions, select **All Definitions**.

3 If you selected a specific module, in the Module Contents pane, select the definition type that you want to search. Click **Views** and select the desired definition type. For example, to look for a specific query, select **Queries** in the list.

4 In the Module Contents pane, in the **Search Definitions** box, click .

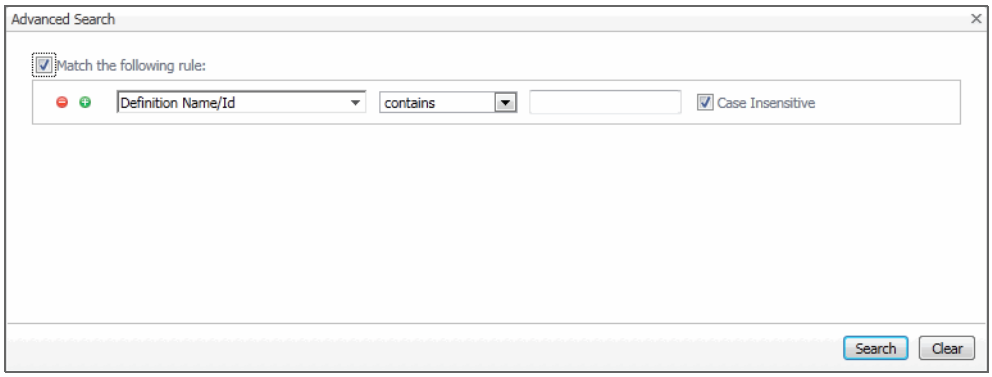
A menu appears.

Figure 13. Search menu

5 Click **Advanced Search** in the menu.

The **Advanced Search** dialog box appears.

Figure 14. Advanced Search



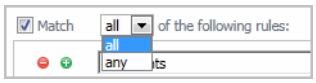
6 Specify one or more rules to search for definitions.

For example, to search for the definitions that contain the word “alarm” in the **Comment** field, compose the following rule:

Figure 15. Advanced Search rules

- To add or remove rules form the search, in the **Advanced Search** dialog box, use the Delete Rule and Add Rule buttons , as required.
- If you have multiple rules in the search, indicate if you want one or all rules to return a match by making the appropriate selection.

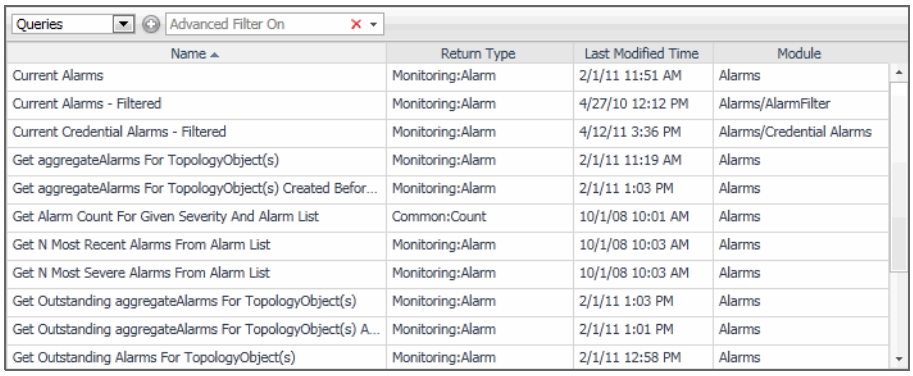
Figure 16. Advanced Search parameters



7 Click **Search**.

The Module Contents pane refreshes, showing a list of definitions that match the advanced search.

Figure 17. Matched definitions



TIP: The search box shows **Advanced Filter On**, indicating that the entries in Module Contents pane are retrieved by an advanced search.

8 To clear the search results, click .

The Module Contents pane refreshes, showing a list of all definitions of the selected type.

Definitions Page for a View

View Commands in the Definitions Area

The following commands are available for working with existing views in the Definitions area:

- **Add**: Create a new view.(for creating new views only. This option appears in the Module Contents pane).
- **Convert To**: Convert the view type. Some view components allow conversion to another component type. For example, Grid Layout, Columns Layout, Rows Layout, and Splitter Layout can be converted to a different layout type. Additional view components may also be enabled for conversion.

  TIP: To find out which conversion options are available for a particular view component, select it in the Module Contents pane. If a **Convert To** button appears in the **Definitions Pane**, the selected component is enabled for conversion. To find out to which view components you can convert the selected component, click **Convert To** and review the options in the menu that appears.

- **Copy**: Make a copy of an existing definition. See Deep Copying Views on page 35.

- **Edit**: Switch the Definitions pane from view mode to edit mode.
- **Move**: Relocate a non-public view to another module with a common ancestry, for the purpose of changing its ID. Public views cannot be moved. The tree of target modules that appear available for selection reflect that restriction. The movement is restricted to modules with a common ancestry because the view may have a reference to another non-public entity, and moving it outside of that tree makes the reference invalid.
- **Relocate views**: Only for System modules, and the view must be public to move it outside its current module hierarchy.
- **Remove**: The component from the module.
- **Test**: Show the completed view in a test container.
- **Validate**: Check that the view meets certain consistency requirements.

These commands are available in edit mode:

- **Cancel**: In edit mode, cancel editing and do not save changes.
- **Config Wizard**: Only for tables, allow column assignment using a wizard.
- **Save**: In edit mode, save the definitions for this component.
- **Test**: Show the view in a test container while it is under construction.

You can find these functions in the Definitions pane in **Configuration > Definitions**.

Editing a View

Select the *My Definitions* node containing your view from the Module List pane, select the view you want to modify, then click **Edit** (Edit) in the Definitions Pane toolbar.

NOTE: For more information, see “Definitions Pane Settings Tabs” on page 44.

Saving a View

To save a view, the name field and required properties must be defined. Ensure that the view you want to save is in the **Definitions Pane**, and then select **Save (** Save **)** from the **Definitions Pane** toolbar.

Removing a View

Ensure that the view you want to delete is in the **Definitions Pane**, and then select **Remove (** Remove **)** from the **Definitions Pane** toolbar. A dialog box is displayed asking you to confirm the deletion. You can only delete views for which you have the proper permissions. You cannot delete a view if there are references to it. A public view cannot be deleted because modules from other cartridges that are not currently loaded might reference it. It is possible to delete a public view by forwarding. The view is deleted, but it remains as a pointer to the newer replacement view.

Deep Copying Views

Ensure that the view you want to copy is in the **Definitions Pane**, and then select **Copy** > **Deep** from the **Definitions Pane** toolbar.

Figure 18. View Copy menu

Deep copying a view copies the selected view and the related entities, including any localized strings that are referenced using that view to the current module. References to public definitions in the same module as the view being deep copied are copied as well (unless copies of such already exist in the target module). References to public definitions in other modules are not copied.

It takes into account:

- Context entries (view, flow, window) that are bindings, which may have references to queries, renderers and localized strings.
- Configuration property values that are set to bindings, which may have references to queries, renderers and localized strings.
- Configuration property values of type renderer, which when set have references to renderers.
- Flow, which may have references to tasks and to other views.
- Derived queries that are based on other queries.
- Container views that have references to other views.
- Function scripts that may reference other functions or queries.

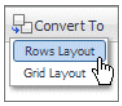
  Notes:

  - Views containing non-public components are not supposed to be copied, but they can be deep copied. Also, you can perform a normal copy and choose which of the private views you require.
  - The view to be deep copied must not be in the current module.
  - All referenced views to be included in the deep copy must have their allowed roles satisfied by the roles of the current user.
  - User modules do not have their own resource bundle, therefore if you deep copy a view into a user module, references to localized strings are not deep copied. The references are left as is and continue to refer to the original localized string.

Converting a View

The **Convert To** button allows you to convert one view component to another without the trouble of copy and replace operations.

Figure 19. View Convert To menu



For example, if a Rows Layout component is configured and then you decide a Grid Layout is more appropriate, simply convert the Rows Layout component to a Grid Layout. The types of view components that you can convert to depend on the selected component. For example, a Rows Layout can be converted to a Columns Layout, Splitter Layout, or Grid Layout, while a Grid Layout can only be converted to a Rows Layout or Columns Layout. Additional conversion options may be available.

TIP: To find out which conversion options are available for a particular view component, select it in the Module Contents pane, in the Definitions Pane click Convert To, and review the options in the menu that appears.

When the conversion is complete, the converted component should be verified to ensure it displays as expected. For example, when a layout is converted, check if the child views are laid out properly.

Nested Views in the Add View Dialog of a Container

You have the option when adding views to a container of choosing **Select existing view** or **Create a nested view**. Many Nested views are private views that are contained only in the specific container.

When creating a nested view, the **Context** tab allows the user to define additional context for the nested component. All context inputs of the nested view are read-only.

A context input of a nested view might be converted to an optional, implicit context input. This can happen if the nested view has an action configured that has flow context mapping that is used for input which causes an implicit (optional) context input to be created.

Replacing Views

Since system views are predefined views that are shipped with a WCF-based application, they may be referred to by other views, which means that the IDs of these system views should not change. However, if you want to replace a system view with a new view that is based on a different component, you are not able to edit the view and change its component type. Instead, you create a new view and then replace the old view with a reference to the new view.

In order to do this, select the view to be replaced in the Definitions editor, then click the **Remove** button. Regardless of whether the view is removable (whether it has dependents or not), you have the option of replacing the view. If you choose to replace it, you then see a tree of views that includes only the views that have context inputs that are compatible with the required context inputs of the view to be replaced. Once the replace is confirmed, the view that was replaced no longer appear in the Definitions editor. Any references to it are automatically be converted to refer to the selected replacement view.

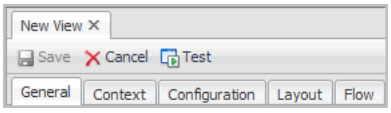
Testing a View

You can test a view to ensure that it is configured properly.

Definitions Pane Settings Tabs

When you add a new view and begin to edit it, or when editing an existing view, the Definitions pane displays a series of tabs. These tabs enable you to choose the pages that let you configure the view definition properties. Each view is defined by a group of settings and each of these groupings has its own tab in a definition. These are the tabs in edit mode:

Figure 20. View tabs



The type of View Component that you are creating determines which View Definition tabs display. See the *Web Component Framework* in the online help for Foglight for the complete list of properties for each view.

NOTE: The Views tab exists for some containers. The Layout tab is not available for some views.

Configuring Properties in the Definitions Pane

- General Tab
- Context tab
- Additional Context
- Configuration Tab
- Views Tab (displays for container views only)
- Flow Tab (some views may implement flows indirectly)
- Layout Tab (displays for some types of container views only)

General Tab

As you are creating a new view, you need to configure its properties by specifying the following items and fields.

Table 10. General settings

Item or Field	Description
Module	The module in which this component resides. The module is set by selecting a module in the Module List pane before clicking the Add button in the Module Contents pane.
Component	The specific type of the view component, such as a Tree Table. The component is set in the **Add View** dialog box after you click the Add button in the Module Contents pane.
Name	The view name. You can choose any name you wish.
Public	Determines how this component can be referenced by other components. If true, the component can be used in any module. If false, the component can only be used in its own module or sub-modules of that module.
Deprecated	Marks a view that cannot be used in any new definitions, but that should not be deleted because some other module might be using it. The component is removed from any drop-down list that allows components to be selected. For instance, it becomes unavailable for inclusion in a grid's layout.
Preferred Width	The width of the view in pixels. It is used as the default setting in various places, such as charts and reports.
Preferred Height	The height of the view in pixels. It is used as the default setting in various places, such as charts and reports.
Refresh Interval	The time interval after which the page refreshes itself by requesting new data from the server. This overrides the default refresh interval set in User Preferences.
Priority	The *Priority* setting is used to control which views are presented in the view pane of the Data tab when a node in the list pane is selected. The Data tab presents another way to browse Foglight data. A running Foglight system instantiates as many objects as it can from the data models defined in its agents and in its core system. These data objects are presented in the list pane. When one of these data objects is selected, the Foglight® system searches for any dashboard or monitor views that have as their sole context input a data object of this type. If a priority has been set for the view, it is displayed in the view pane. High priority views are shown first, and then those of lesser priority. If a priority has not been set, the view does not appear even though it has the proper input. Choices for the *Priority* setting are: None, 1 - 9, (Low=8, Medium=5, and High=2)
Purpose(s)	See Purpose of Views on page 39.
Custom Purpose(s)	See Purpose of Views on page 39.
Relevant Role(s)	See Roles on page 40.
Allowed Role(s)	See Roles on page 40.
Comments	A brief description about the view.
Context Help	A detailed description about the view. The description that you provide here becomes the online help for the component. When a user chooses this component from the drop down list on the Help tab in the action panel, the text is displayed. For this reason the text you provide here should fully describe the component from the user's point of view.

After the component has been saved the editor pane reverts to the definition pane. The following additional properties are presented:

Table 11. Additional settings

Field	Description
Context Inputs	A table of context settings for the view.
Reference Id	The internal name for the component by which is referenced. Useful for exporting and importing custom views.
Last Modified Time	The time at which this view was last edited.

Purpose of Views

Each view can be marked as having one or more purposes. The standard shipped purposes are:

- **Dashboard**: A dashboard is a page that cannot have required context inputs. The filtered list of views that you can add to a dashboard can contain any view that is purposed as a Pagelet as long as the dashboard contains the appropriate context to make the view useful. Any view marked as a dashboard can be accessed byselecting it in the navigation panel.
- **Data View**: A data view is a view that is presented in the list of view choices in the data browser (accessible by choosing **Dashboards > Configuration > Data** from the navigation panel) when an object compatible with the single required input of this view is selected from the data tree. Note that a priority (other than none) must also be set.
- **Diagnostic**: Useful for identifying views that are meant to be the beginning of a diagnostic work flow. This is the opposite of Monitor. If it is used as a drill-down page, it freezes its time range so that even if the page is updated the time range does *not* update.
- **Dialog**: A view that is designed to *not* update automatically. Useful for identifying views that are designed (in shape, size, and interactivity) to be Dialogs, either temporary or full.
- **Feed**: This view is useful as a syndication feed (for example, Really Simple Syndication (RSS) or Atom) used to frequently update content. Only a view based on Feed View Component should be purposed as a feed. A component purposed as a Feed can be used as a way to query data from Foglight because it provides a well-defined XML stream that you can parsed to extract the information you want. For more information, see Setting up RSS Feeds on page 114.
- **Global Action**: When set, a link to this view is shown in the right panel (page/action panel) under *Other Actions* as long as the user is permitted to see the view. A Global Action view cannot have required inputs.
- **Home Page**: Useful for identifying views that are designed to be home pages. It cannot have required context inputs. A link to a view with this designation appears in the Homes list in the navigation panel.
- **Menu**: Useful for identifying views that are designed to be menus.
- **Monitor**: A view marked as a monitor is meant to reload itself periodically, since it is meant to monitor the data it is presenting. By default, the refresh interval is obtained from the user preferences but can be specified using the refresh interval property of the monitor view. The policy for refreshing views is as follows:
  - Child views that are not marked as monitors refresh at the rate of their ancestor.
  - Child views that are marked as monitors but are set to refresh at less than the rate of their ancestor still refresh at the rate of their ancestor.
  - Child views that are marked as monitors whose refresh rate is more frequent than their ancestors refresh at their own rate.
  - Child views marked as monitors but whose input *timeRange* do not have an idea of *now* (the current time) do not get refreshed. For a child page's *timeRange* to have a notion of *now* it must be passed a *timeRange*.

IMPORTANT: Monitor purpose affects the type of label that you see in the center of the page header as shown in the following policy code:

```
IF (there is a timeRange menu (topview has timeRange as a context input))
```

```
IF (topview is a monitor and none of its children have a smaller update
interval.) OR (topview is not a monitor and all other children also are
not monitors)
   Show TimeRange
ELSE
   show no label
ELSE Show just time stamp
```

- **Page**: A page is a view that is designed to be used as a page – one that is drilled down to from another page or dashboard or portlet. It can have required context inputs.
- **Pagelet**: A pagelet is a view that is designed to be added to a page or dashboard.
- **Portal**: The result of using Create Dashboards in the action panel. A portal is page that a user can customize by dragging views onto it. The filtered list of views that you can add to a portal can contain any view that is purposed as a portlet as long as the portal contains the appropriate context to make the view useful. If used in a remote interface, for example, in a WebLogic, a drill-down to a portlet takes place in the remote window. A drill-down to a page with any other purpose directs the browser to the Foglight® browser interface.
- **Portlet**: A portlet is a view that is designed to be added to a portal.
- **Report**: A report does not require any input. The filtered list of views that you can add to a report can contain any view that is purposed as a reportlet as long as the Report contains the appropriate context to make the view useful. Scheduled reports that have been generated appear in the Reports view.
- **Reportlet**: A reportlet is a view that is designed to be added to a report.
- **Summary**: Useful for identifying views that are designed to be small read-only summaries of data. Often useful for dwell actions.

Custom Purposes

You can enter custom purposes, using whatever names you want. Enter the custom purposes as a comma-separated list in a single text field.

No actions are associated with custom purposes in this release. In future releases, custom purposes will be used in various places in the GUI for restricting the list of views that can be seen to those who have been assigned the same custom purpose.

Roles

For more information, see Relevant Roles and Allowed Roles on page 67.

Comments

Creators of view components use this field to communicate information that might be helpful to other developers in creating similar components.

Context Help

Use this field for context sensitive help text. If the component is a top-level view, whatever you type here appears in the right action pane under the **Help** tab.

If the view is within a container and its border is showing, context sensitive help is available by clicking on the help icon in the view's title bar.

For a dashboard, the help text appears as a tooltip when the mouse pointer hovers over its entry in the navigation panel. It also appears when a user chooses this component from the drop down list in the Help tab in the action panel.

Context tab

Context packages the values that the page or view requires to present its data. For more information, see Context Types on page 52.

Pages and views are similar to forms waiting to be completed. Their definitions can specify the type of data that they display, just as a form can require a name or an address, which are filled in when the form is used. They do not set specific values for the item(s) that specify the subject of the form. For views, this is provided by the context data during run time, which is declared in the Context tab and is set in the Additional Context group for the current view or page. If a context has been set by some view, it can be used in other views by declaring it in that view's context.

Context Inputs

If the context of a page has been set, then a view can use that value by specifying the named value of the context. (For more information, see Context Tab on page 51.) For example, a *Host* page sets its current host into the context using the key *host*. A flow to another view uses that context by setting its context input (the expected context value) to use *host* as its key.

Similarly, if a page has a given named value in its context, and a link takes it to another page, then that new page can get access to that value by specifying its key as a context input.

Context can be passed when moving from one page to another. For example, click on a link in the *Hosts* view, and the *Host Monitor* page is displayed with data for the same host. In this case, the *Host Monitor* page expects that:

- a list of *Host* data objects is required (*hosts* context)
- a specific *Host* data object may be provided (*host* internal context input)

The *host* is set in the Context Inputs section and *hosts* is set in Additional Context.

If a view is not passed the required context input, it uses the persisted value, if there is one. For example, if a child view is a table and *selectedRow* is passed on an Update flow to its container, the container must have *selectedRow* defined as an Internal key. For a discussion of the types of context, see Context Types on page 52.

Additional Context

Additional Context for a view allows you to set values on context keys for the view to use. If it is a container view, the additional information can be used by its children.

Outputs (Flow Context Mappings)

Some components may generate context values based on actions that the user performs on the component. These are listed in the component reference help, under *Global Generated Contexts* , and in the *Generated Context(s)* section under *Actions*. You can use this section of the Context editor to define a key for one or more of these context outputs if you need that information in a dependent view.

Configuration Tab

Configuration properties are those that control the overall look of the component and where the component gets its data. For instance, in a Splitter you can control whether the split is horizontal or vertical. In a simple component like a Label, two important configuration properties are the label text and background color. In a complex component like a row-oriented table, there are a large number of properties for controlling how the rows are configured. In addition to the main property for setting values on rows, there are others for sorting, setting headers on columns, controlling width, and many others. For information on configuring the properties of individual components, see the *Web Component Reference* pages.

For an introductory overview of the configuration process, see the *Web Component Tutorial*.

NOTE: You can use the Edit View Properties dialog box to edit some of general properties of views without going to the Definitions pane. Click the General tab in the action panel. Under Actions, choose Properties > Edit Basic Properties. You can set relevant and allowed roles, the page refresh rate, and the context help text for the page.

Cooperative Layout

Some container views (for example, Rows Layout, Columns Layout, Fixed Layout, Grid, Iterator, and Tab Layout) allow cooperative layout to be turned on using the Cooperative Layout property. When this feature is set, eligible child components (all chart components) of the same size align themselves within the container. In the case of the chart components, charts of the same width align their *y* axes to the same horizontal location within the charts and charts of the same height align their *x* axes to the same vertical location within the charts. Individual charts may be exempted from cooperative layout by setting the **Cooperative Layout** property on the chart to false.

By default most containers have cooperative layout set to **inherit**, which allows a parent container with cooperative layout enabled (if one exists) to include the subcontainer's children components in its cooperative layout. (The exception is the Tab Container, which defaults cooperative layout to disabled for performance reasons, and the Portlet container, which defaults to on.) Setting cooperative layout to **off** means that the container's child components will not be laid out cooperatively, no matter what the setting of its parent container is.

CAUTION: Set the Cooperative Layout property on the parent (for example, the grid), not on the child, since it is the parent that is responsible for laying out the child.

Common Properties in the Configuration Tab

There are a number of common properties for each view: Data Availability, Customizer, Background, Page Options, Portlet Options, Popup Options, Report Options, and Sizing.

See any component in the Reference Pages for a description of the properties available in these nodes.

Table 12. Common properties

Property	Description
Background	Sets an optional background color and/or background image for the view.
Customizer	The properties for a view used as Customizer.
Data Availability	Option that allows asynchronous data retrieval. This is useful if the data displayed in this view can take a long time to load. By using this mechanism, the page can finish drawing and then check back periodically to see if the data is ready to display or not.
Error Renderer **Null Renderer**	Renderers to use if an error occurs when attempting to load the component or if null is returned.
Icon	An icon can be associated with the component. It appears if Page Title in **Page Options > Page Style Options** is set to **Descriptive**.
Page Options	Options that apply when a view is a page (a top level view).
Popup Options	Options that apply when a view is used as a popup.
Portlet Options	Options that apply when a view is used as a portlet.
Report Options	Options that apply when a view is associated with some of the report template views.
Sizing	Options that are set on any view for use by certain layout components, for example, by the Grid, Rows, or Columns layouts). The **Horizontal Behavior** and **Vertical Behavior** setting are also typically used by the Perspectives Viewer, Tab Manager, and switchable layouts. These options help ensure that the view is displayed correctly. See Horizontal Behavior and Vertical Behavior, Minimum Height and Minimum Width, and Maximum Height and Maximum Width below for more information.
Title Flow	The flow to execute when the title is selected. The component must not be a top level view.

Horizontal Behavior and Vertical Behavior

These options tell the layout component whether the view can be resized (is stretchable) or has a fixed size and requires scroll bars if the layout cannot allot it sufficient space. In addition to the Grid, Rows, and Columns layouts, these options are typically used by the Perspectives Viewer, Tab Manager, and switchable layouts[what are switchables?].

The **Width** and **Height Behavior** values of the **Sizing** property can be set to one of the following options:

- **Natural**
  - The component is sized to its natural dimensions, based on its data and configuration.
  - Usage examples include labels and natural tables.
  - The **Minimum** and **Maximum** values of the **Sizing** property override the natural size, which is useful for labels.
  - The **Preferred** value is not relevant, unless the component (such as a Time Plot Chart) has no natural size; in that case, the specified value is used as the natural size.
- **Specified**
  - The component's size is the is the size you specify.
  - The size is typically set by a developer, based on specific design considerations.
  - The **Preferred** value is used as the specified size.
  - The **Minimum** and **Maximum** values are not relevant.
- **Stretchable**
  - The component can be stretched or compressed to accommodate the size of its parent.
  - Examples include Chart components that are stretchable in both directions.
  - AJAX tables are naturally stretchable vertically.
  - The **Preferred** value is used if the container has no notion of how big it should be (for example, in popups).
  - The **Minimum** and **Maximum** values are used to ensure a component is not squeezed or stretched in a manner that makes unsightly or unusable.

Use the following considerations when setting the **Width** and **Height Alignment** values of the **Sizing** property:

- **Start**, **Middle**, and **End** correspond to left, middle and right if applied horizontally, or top, middle and bottom if applied vertically.
- Determine how to align a component within its container if there is more space available than needed. This occurs when the available space is greater than the **Natural** or **Specified** size, or in the case of **Stretchable**, the **Maximum** size.

Minimum Height and Minimum Width

The minimum size options are only applicable if the view is stretchable. These options set a limit to how small the layout can make the view when the layout stretches its contents to fit the space available. The combined minimum and fixed sizes of a layout's children in a particular direction effectively also create a minimum size for the layout.

These options are hints about sizing that are provided by the view to the layout component. They are overridden by the layout if you set a minimum size on the view's window properties within the layout.

Maximum Height and Maximum Width

The maximum size options are only applicable if the view is stretchable. Maximum size limits the growth of a view to an upper bound. The view does not stretch beyond this. This is not a hard upper limit on the size of the rows or columns in the view. The view can contain other components permitted to take more space.

A view's **Maximum Height** and **Maximum Width** options are not overridden by the layout. The layout honors these settings. For example, if you create a view and set its maximum width to 100px, provide it with content wider

than 100px, and place it in a grid that has layout properties that permit the view to stretch, the view continues to have a maximum width of 100px even if more space is available.

CAUTION: It is possible to select non-metric value when setting a metric. If you do select a non-metric value, no values display in the chart. For example, the *Host* property *CPU Usage* is a metric that belongs to a parent, but the *Average* is not. It is derived from the metric. Both metrics and non-metric data are selected from a dropdown tree. Currently, there is no indication whether or not an item is a metric.

Flow Tab

When you click a link on a page, what displays next depends on a common set of actions called *Application Flows*. These actions are set in the view editor's Flow tab.

Flows can be set on parts of the component, such as a table's rows or individual cells.

The resulting action depends on the current dynamic context. In the case of a row action, the view switches to a new page that shows the details for the object being used to populate the cells of the currently-selected row. These actions are set in views, but the targets may be popups, or other pages, or even other applications. The actions themselves rely on a context to give them information they need.

NOTE: When selecting a view as a target for a flow, only entities that have their Public property enabled are shown in modules that do not have a common ancestry with the module of the current definition. Thus, views in the current hierarchy can be set as targets if their Public property is not set, but views in other modules must have their Public property set if they are to be available as targets.

Configuring Flows for Views

The following sections provided details about how to configure flows for a view.

Flow Types

The flow types are:

- Update
- Next Page
- Popup
- Previous
- Sequence
- External URL
- Choose Value
- Choose Type
- Select Type Flow
- Select Tagged View
- Invoke Task
- Show Help
- Create View

NOTE: For the Next Page, Popup, Select Type Flow, and Select Tagged View flow types, a diagnostic workflow is available via a check box. If you mark a flow as diagnostic and if the time range is connected to "now" (real time), it will automatically remain unchanged so that the data and the data window you are looking at does not shift until you cancel this option by resetting the time range.

Update

Update refreshes the page with the new context. Where the context key is restricted using an internal input, the change is restricted to components that are children and grandchildren of the view marked with the internal context input. If a child page's context is marked as optional, user action that changes this key does not automatically propagate to the parent unless the parent explicitly declares the same key and sets it to *Internal*.

This action refreshes the context for all the dependent views. For example, if you change the context on *view A*, then the data in *view B* is updated, and the data in *view C* is updated. By default, the context is initialized the first time a page or view is displayed. You may want to re-initialize context when more than two views are dependent on each other. If *view A* is not set to re-initialize its context, then *view B* updates, but *view C* does not change.

There is a *Preserve current timestamp* option, which if enabled allows an update to the page without changing the timestamp. This can improve response because fewer components need to be re-rendered.

CAUTION: Context keys must be explicitly specified in the update flow action otherwise they will not be passed on. This allows control over which context entries are updated and which are not. This is different than the next page or popup reactions, where any context that is needed is automatically forwarded.

NOTE: A view whose context is marked as optional cannot be added to a container that has the same input, otherwise if a flow of type Update is triggered in this component, the context is retrieved from its container. The view's context entry won't replace the one in the container.

Next Page

Next Page sets a specific target page. Select the target from a tree of pages. The tree is filtered to only include non-deprecated pages, and dashboards (that have their required context inputs (if any) satisfied by the definition being edited). The purpose of the target page must be *Page* or *Dashboard*. This means that the type and the list property what is list-ness?of the inputs must be satisfied. A target page must have its public attribute set unless it is in the same module hierarchy as the page that references it.

Popup

The tree of target views is filtered to only include non-deprecated views that have their required context inputs (if any) satisfied by the definition being edited (and their public attributes set unless they are in the same module hierarchy as the referencing page). This means that the type and the list property of the inputs must be satisfied. Also, the list of options is filtered to match views whose purpose includes *Pagelet, Summary, Menu and Dialog*. A target page must have its public attribute set unless it is in the same module hierarchy as the page that references it.

A Popup has four different behaviors:

- **Dwell**: If the popup is associated with a dwell action then it shows the target view in a transitory read-only window. This disappears when the user clicks the mouse anywhere.
- **Transient**: If the action is not dwell and the selected popup type is "transient" the target view is displayed in a transitory window, but in this case user interactions are allowed. The target view disappears if the users clicks outside of it or if another action is fired.
- **Dialog**: If the action is not a dwell then the target view is created in a popup dialog box which the user can interact with, move about, and dismiss. This new popup view is part of the page and disappears along with the rest of the current page if you traverse to a different page.
- **Modal Dialog**: A modal dialog is like a regular dialog box, except that the rest of the UI is frozen until the modal dialog is dismissed.

Previous

When this action is invoked the page reverts to a previous state. There are two separate notions of the meaning of previous:

- If the view that initiates the reaction is somewhere on a page then the page switches to the previous one in memory. This is synonymous with clicking the previous page in the breadcrumbs trail at the top left of the page.

- If the view is a top level view (that is, it has not been drilled into or the only breadcrumb is the name of the view you are looking at) it has the same effect as using update on that page with the exception that the context is not updated.

  IMPORTANT: When configuring a Previous flow type, the Update check box is cleared. If it is selected, the context is updated as the previous page is reloaded.

If the root view causing the firing of the previous action is contained in a popup dialog box, then previous causes the dialog box to pop down.

Sequence

The Sequence flow type is used when there is a requirement to do more than one type of flow, so it is used to execute a list of reactions sequentially. The order in the list is important and the execution of one reaction affects the next one, including changing the context.

This allows you, for example, to update the current page with a selection from a table (therefore causing the selected row to be remembered), and then go to another page. If you return to the first page, it is the updated page that is displayed. In this case the first reaction in the list should be Update and the second should be Next Page (reversing the order would cause the update to affect the next page, which is not the desired effect).

CAUTION: A Task Component that calls the method ActionContext.setResponseData cannot be used in a sequence flow type because sending two different data types in the same response is not supported. The Task components in WCF that call this method are: Download File and Template Generator.

Select Tagged View

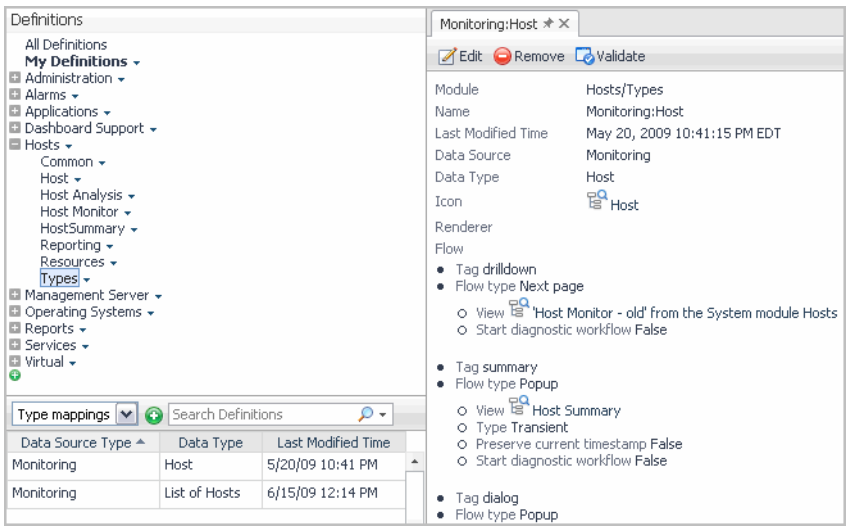
The Select tagged view reaction is used in Foglight to implement Rule Based Drilldown. Its use is application specific, and is intended for internal development purposes.

Select Type Flow

A Type Flow is defined in a Type Mapping using the **Type mappings** choice in the Module List drop-down. You can associate a default UI flow (via a tag) with a specific data type. For example, you can define a default flow for a Foglight Host to be a drill-down to the *Host Summary* page.

Type Flow links to selected pages when you specify a type of object. The figure shows a typical configuration. In this case, all cases flow to the same view, the *Host Summary* page. You have the option of marking the popup as temporary or permanent, which is the reason for having two drill downs to the same page. Also, the flow might be based on a menu choice.

Figure 21. Type mappings



To enable this Type Flow in a view, use Select Type Flow when setting up the flow type. For example, a cell selection on the Host column triggers a Type Flow based on the data type of Monitoring:Host, the host that is currently selected. If the preceding Type Flow configuration is in place and the selected item is a Host instance, the system drills down to the *Host Summary* page.

If *Select Type Flow* is the flow type, Foglight looks for all Type Mappings available in the system for a match. If an exact match is not available, the search continues up the data type inheritance tree until a match is found; an error is reported if this fails. For example, if a WebLogicServer Type Flow is not defined, the system follows its inheritance tree to look for a match: *WebLogicServer* > *JavaEEAppServer* > *JavaEECollectionModelInstance* > *CollectionModelInstance* > *ModelInstance* > *TopologyObject*. If the search reaches the root without finding a match it reports an error.

The tagged flow mechanism allows more than one flow to be configured for each data type. Using tags you can specify a default dwell, a default temporary popup and a default drill down for each data type.

To define the flow for a view:

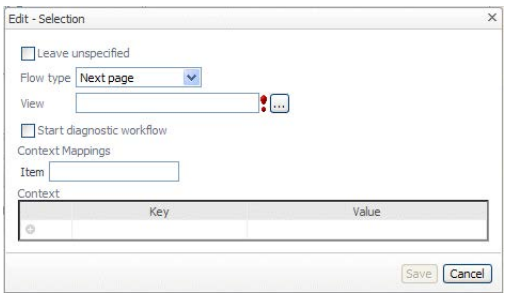
1 Open the **Flow** tab.

2 Click one of the actions.

The Edit window opens and displays a set of input fields that allow you to further define the flow.

3 Clear the **Leave unspecified** check box.

Figure 22. Edit Selection



4 Click **Flow Type** and select one of the options.

IMPORTANT: The contents of the dialog box depend on the type of flow that is chosen.

5 If the type is Select type flow, choose one of the tags from the drop down list or type in a tag name of your choice.

6 To use the type flow to drill down from a context key:

a Add a node to the navigation tree.

The settings of this node determine the drill down behavior.

b To edit the settings of the node:

c Specify the metric referenced by the view. Use the same metric that is used in the chart configuration.

d To choose from existing type flows, click **Flow type** and choose **Select type flow** from the list.

e Specify the type whose flow you want to use. This is the name of the view's context input key whose data type, matches the type flow.

NOTE: Because Foglight allows for a maximum of one type flow for each Foglight 5.0 data type, specifying the type flow is sufficient.

f To apply your changes to the drill down, click **OK**.

7 To use a *Next page* action to configure the drill down, complete the following steps:

a Add a node to the navigation tree in the Module Contents pane.

b Select the **Update** node.

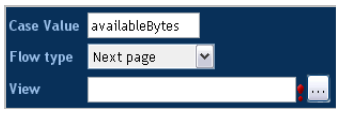
c Specify the metric that will be passed down to the view. Use the same metric that defines the metric in the chart.

d To do that, in the right pane, in the **Case Value** box, type the name of a metric.

e To open another view when the user clicks the area in the chart that shows a metric, click **Flow Type** and select **Next page** from the list.

The right pane refreshes and shows the **View** box under **Flow type**:

Figure 23. View box displayed



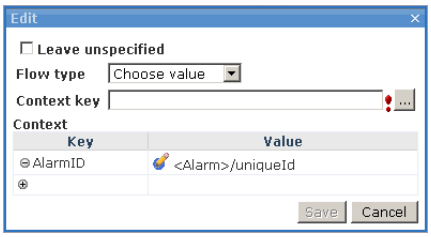
f Specify the view that you want to appear when the user clicks the area in the chart. To choose from existing views, click the **Browse** button to the right of the **View** box.

g Click **Apply**.

8 To save your changes, click **Save**.

The Edit window refreshes and the **Context key** box appears below **Flow Type**:

Figure 24. Context key box displayed



9 Click the **Browse** button to the right of the **Context Key** box and choose a name from the list that appears.

10 Click **Save**.

Choose Value

Choose Value lets you select different actions and targets based on case values. For instance, you might want to have a different reaction depending on which cell of a table the user chooses. You would set a Choose Value flow type on the Cell Selection flow and then you would add cases for the various columns. You choose a name key for the column and associate a flow with it. This key name must be the same one that you assigned to the column ID under Column in the table's configuration tab. Thus a different reaction can be associated with each column in the table.

Follow the procedure given below to configure a case.

To set the Choose action:

1 Select the **Choose Value** action from the *Flow type* drop-down list.

2 Select a key from the *Context key* drop-down list. This is the key that you have configured to have different bindings that are going to be used as case values.

3 Click **Save.**

The context is added to the action in the tree.

To add a case:

1 Click ⊕, whose tooltip is *Click to add a case*.

A new row for the case is added under the default row.

2 Click the new row.

3 Type a context key in the *Case Value* text field and specify a Flow type in the drop-down list.

4 Enter the context key for the case by selecting the field and then entering the key.

5 Select the flow type and add context entries if needed.

6 Click **OK**.

To remove a case

- Click ⊖ on the row of the case.

There is also a *Context* group box where you can specify any additional context to be passed to the target view. This may be necessary if the target view has a required context that is not already defined in the calling view's context.

Choose Type

Choose Type is similar to Choose Value, but instead of directing actions based on the value of a context element you can do this by the Type of the object. Often the reason to do this is that you have a generic object and depending on the more specific nature of the run time type you want to perform a different action.

For example, use *Choose Type* if your type is Host and you want to go to different pages depending on whether the actual object is a WindowsHost or an AIXHost.

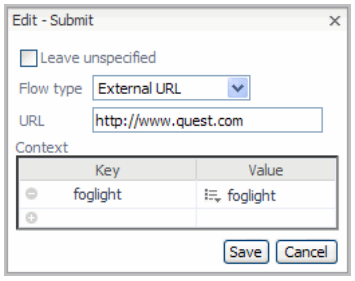
Invoke Task

Invoke task allows you to invoke a task when an action occurs. For example, you acknowledge an alert or launch a program on your desktop with information from the Foglight® application. The tree of tasks is filtered to only include non-deprecated tasks that have their required context inputs (if any) satisfied by the definition being edited. This means that the type and the list property of the inputs of the task to be invoked must be satisfied.

External URL

Instead of flowing to another Foglight view, any Web page can be referenced by specifying its URL.

Figure 25. External URL



External URL lets you set any URL address as the link target. If you set the external URL to the string {url} the system looks in the context for a key URL and uses its value as the URL to go to. This is useful in conjunction with a template allowing you to customize the URL with runtime data from Foglight.

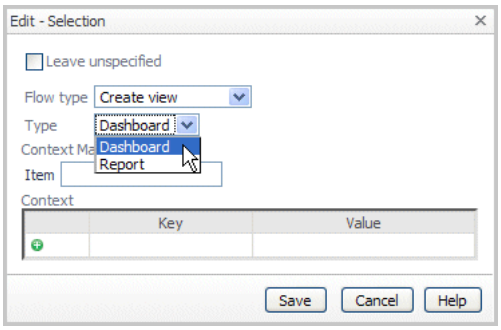
Show Help

This choice is used by Foglight developers to launch the help associated with a specific help key. This choice is useful for connecting alarm states or in deciding what to do with a particular section of the help.

Create View

This choice provides a link to the Create Dashboard or Create Report wizards.

Figure 26. Create View



It is useful if you want to give the user an opportunity to create one of these views via a direct link.

Layout Tab

The Layout tab is used in the Fixed Layout and Grid, and HTML Table Layout containers. The tab contains a layout frame. At the top of the frame is a toolbar with these functions:

- **Add**: Add an existing component to the layout.
- **Define Customize**: Place a link to a view in the General tab of the action panel.
- **Properties**: Show the placement properties dialog box for the selected component.
- **Inspect**: Inspect the chosen component by opening it in the Definitions pane.
- **Remove**: Remove the selected component from the layout.

The Layout frame shows the views that are on the container and their approximate sizes. The guide lines in the background show you how the views would fit in browser windows with different screen resolutions.

Views Tab

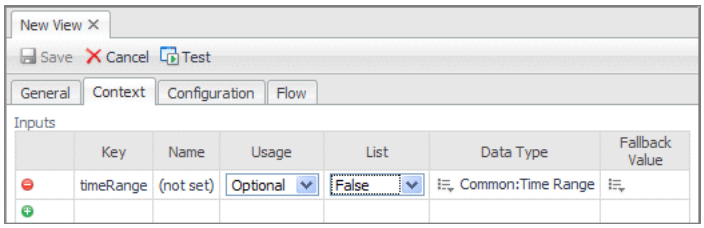
The Views tab is used in the Iterator (where it is called an Iterator View), PDF Layout, Switch, Tab Container, Topology, and Type containers. The tab incorporates an editor for setting views for the container.

Context Tab

Context is the term used for the information available at a given point in time to the various components used by the Web Component Framework. Context in the *Web Component Framework* is similar to a collection of variables that are used in programming languages or an operating system's environment variables.

Context Inputs are defined in the Context tab when editing the definition of a component.

Figure 27. Context Inputs



Every element of the context has a key (name), a corresponding type, and a run time value in an active Foglight instance.

Figure 28. Context parameters

Context consists of:	
Key	The key used to reference the object in other dialog boxes or scripts. Note that the *timeRange* key should be kept in the list of defined keys even if the page is not dependent on the time range.
Name	A more descriptive name for the key.
Usage	Choices are Required, Optional, or Internal.
List	True if the key is associated with a list, False if it is associated with a single item, and Unknown if it might be either.
Data Type and Data Source Type	The data type of the object and the source of the data object referenced by the key. The choices are in the **Select Type** dialog box.
Fallback Value	A fallback value must be set if the Usage is Optional or Internal. The value is set using the binding editor.

For example, a view that is configured to display alarm data for any one host must be provided with a specific host to give it context. Similarly, a Memory Usage page, which displays memory data for a specific host given a time range, must be provided with two context inputs. In the first example, the host is the context. In the second example the host and the time range together comprise the context.

Pages and views can be compared to forms whose fields are filled in at run time. The context definitions determine the actual data that is specified by the Context, which is set in the following ways:

- A time range, which is available to all components. For more information, see TimeRange on page 56.
- The Context tab in the View editor: You declare context inputs to the component here. You can define additional context entries here as well.
- The Configuration tab in the View editor: For example, by setting row/cell properties and choosing an available built-in context such as <*currentRow*>.
- The Configuration tab in the View editor: By setting a parameter's value in the **Edit - Value** dialog box or a component.
- In the Flow tab of the View editor: You can set additional context here, or map from internal component sources into the context.
- In the Flow Type editor for Page Options > Actions: You can define Additional Context items here.

For example, a page is designed to display information for a selected host. When you click a link, the host object is passed to a detail page. The detail page uses that information as its initial context settings. The Context tab is where you set up the list of required context data. Using a required Context setting means that the page or view is dependent for its data on that context value being available.

Context Tab

Context allows you to define the list of information that is required for a particular view to display the data that the page is designed to show. When a context element is defined it is given a name key and it is assigned a data type.

Global Context

In any application there is global or application level context, which is the information that is available throughout the application. An example of this type of context is *timeRange*. Each application can decide to make arbitrary information available to all views.

Dashboard Context

Context is implemented so that changes to one context element affect other elements in a hierarchy. If a context element changes in a nested level, the change traverses up the hierarchy to the parent level, assuming that an uninterrupted chain of key definitions exists.

User Context

When a user interacts with views the interaction can cause changes to the context. For example, when selecting a particular item in a drop down list, by default the *Web Component Framework* saves the last known values so that when you return to the page, it displays with that same item selected.

NOTE: You can disable this behavior in User Preferences.

Context Types

See also Context tab on page 41.

There are three types of context entries:

- Required Context
- Optional Context
- Internal Context

If a view has only one context input other than *timeRange* or only one required context input (and zero or more optional inputs), it becomes its primary context input. The primary context input and its priority controls whether it appears in data browsers when a node of that type is selected.

Required Context

If the input is marked as required, then it must be provided for the view to operate. You cannot add a component to a parent unless the parent can provide the required context. Thus, the parent view must do one of the following:

- Include that context as a required input.
- Specify that context by defining it as additional context.
- Specify that element in an instance context of one of its child components.
- Specify the element as an optional or internal input and contain another child that promises to provide a value for it.

Optional Context

If the input is marked as optional, then the view takes the data if present, otherwise it has some mechanism for inferring or calculating the data. Often this comes in the form of a drop-down list or selectable table that auto-

selects the first element in the data list if a selected item is not provided. Optional Context items are those inputs that are used if they are present but are derived in some other way if they are not present. For example:

- Define a fallback value for the input. This fallback value can take the form of a query or some other mechanism. If the optional input is not available, the fallback value is taken the first time the page is loaded. Once the page has been visited, there is a persisted value. That value takes precedence over the fallback value.
- Some components such as the Drop-Down List component and Row-Oriented Table, can provide a value if one is not there. For example, if you have a drop down component with a list of Host objects you can automatically select one and post it into a Host key using the Flow Context. Adding a component such as this to a composite view then causes the optional context input value to be populated.

Internal Context

Marking a context as Internal means that the input is required for internal operation of the component and is much like the optional input, except that changes to this value by the component are *not* propagated to the rest of the page on an update. However, the value *is* propagated to the children of the component.

Internal Context items behave like optional inputs, except that when a context element changes in a nested level, the changes do not traverse up the hierarchy to the parent level. This makes it possible to have similar and dynamic views that use the same key names and types co-existing on the same page without adversely affecting each other.

You can use this type of context in a portlet with a customizer. For example, the customizer allows you to select which Host is displayed by the portlet. It is important for the component to mark the Host as internal to make sure that if two identical views are on the page at the same time, but each show data from a different host, that they do not conflict with each other.

> **CAUTION: A view with an internal context cannot be added to a container that has the same context input, because if the view's context changes and the page is updated, this update are not be passed on to the container. Internal Context is not for propagating context from one view to another within a page. It is designed to stop propagation of context when different parts of the page have different values for the same context.**

For more information see Configuring Views on page 27.

Additional Context

Additional Context is context that is added by the view designer. The order of the additional context entries is important because the order that they are listed determines the order in which the values are evaluated (if the values are bindings).

You can add context using the Additional Context tab in the Module Contents pane to the following:

- Views
- Specific Instance of a View
- Flows

Additional Context for a View

Additional Context for a view allows you to supply additional information that your view can use inside its configuration or additionally, if it is a container view, for use by its children.

If you know that you are going to refer to a particular child of an existing context key often, you can create a new key that shortcuts the reference. For example:

`agent = <host>/agent`, where `host` is an object that has a property called `agent`.

This is also useful if you are building a container view and want to include children that require particular keys that are not explicitly present. Example: the child view has a key called *host*, but your parent view uses *selectedHost*. Solution: in additional context, map *host* to *selectedHost*. This section needs work. Find out what “a particular child of an existing context key” is, and what is meant by keys that are not explicitly present. The text above comes from the Wiki.

Table 13. Additional context parameters for a view

Additional Context for a view consists of:	
Key	The key used to reference the object in other dialog boxes or scripts.
Name	A more descriptive name for the key.
Evaluate Once	A Boolean value. If this check box is cleared, the key is evaluated each time the page is rendered. This is useful if the key is set to a binding whose value changes dynamically. If the box is checked, the key is evaluated when the page is created, but never after that. This is useful if the key is set to a static value or if reevaluating the key would cause errors. For example, if the key is used to create a new writable data object that should persist across multiple references to the page. If Evaluate Once is false, a new writable data object would be created each time the page is referenced.
Value	The value is set using the binding editor. It can be a static value or a binding.

Additional Context for an Instance

When adding a view to a container view, it sometimes happens that the available context does not have the right information for the view that you want to include. In this case, it is possible in the container view to selectively create the right context. This is called *additional context for an instance* or *instance context.* The context is for that particular instance of the view and is not available when you use the view somewhere else. It would be more useful to show where to set the additional context, and to give an illustrative example.

Table 14. Additional context parameters for an instance

Additional Context for an instance consists of:	
key	The strongly typed, case insensitive string that references a particular context element. Type checking is done at design time.
value	The value stored in the context element, which can be either a static value or a binding.

NOTE: You would create the additional context of the view itself if you wanted it available to every instance of the view.

Additional Context for a Flow

The progression from one view to another is called a flow. How the new view is made to appear depends on the flow action. The types of flow actions are described in Flow Types on page 44.

Often these views may have input requirements that exceed what is available in the context of the component from which you are coming. In this case, it is possible to define further *Additional Context* that applies to the flow. An illustrative example would be good here too.

Dynamic Context

Dynamic context keys are supplied by the component itself, and the value usually depends on some action taken by the end user. Dynamic context is only available for some components, such as Bar or Pie Chart, Cluster Bar Chart, Time Bar Chart, Time Plot Chart, Drop-Down List, Filter, Radio Button List, Row-Oriented Table, Tree Table, and Topology. The value of a dynamic context item changes depending on what is being rendered at the time, such as a row in a Row-Oriented table, or what is being selected, such as an item in a list. For example, in the Row-Oriented Table the dynamic context key *current Row* is available. When a row is processed for display, that

row's *current Row* value changes depending on which row is currently being rendered. Similarly, when an action is defined for row selection on a table, the selected row's *current Row* value can be passed to the target of the action.

Dynamic context keys are not taken into consideration when the tree of views available for next page or popup flow types are filtered. Thus, they are essentially not available to be used as inputs for other views, they are just available for internal references when a view is being rendered.

Examples of dynamic context keys are:

Figure 29. Dymnamic context keys

Dynamic Context Key	In Component
metric	Bar Gauge
currentDataParent	Cluster Bar Chart
currentItem	Drop-Down List, Filter
currentRow	Row-Oriented Table
currentNode	Tree

There are other components that have properties associated with dynamic context keys. See the *Web Component Reference* pages for details.

Flow Context Mappings

You can create context keys that relate to a flow action. For example, suppose you want to select a host from a table showing a list of Hosts. To get access to the selected Host you have to define a context element for use as a Flow Context for table rows. To do this, define the *Selected Row* built-in context as *Host.* Then you can use it to change pages to a view that requires *Host* as an input. See Additional Context on page 53 to learn about adding more context elements to the flow.

Validation of Context in the Editors

When checking the required inputs of a contained view, optional and internal inputs are provided at runtime by either the container or another contained view.

Type Casting Context

The strict typing nature of the *Web Component Framework* disallows arbitrary casting of one type to another, since in any responsible UI development you need to detect and prevent incorrect input.]

On-null

Many places in the flow and context definition allow you to specify the value to use if the incoming value is an unexpected null value.

TimeRange

Some special behaviors of the context element *timeRange* are:

- It is always available; even if it not is explicitly specified, the low level data access calls use its current application-level value.
- Explicitly including *timeRange* as a required Context on a view that becomes a page causes a time range drop-down to appear at the top right of the page. This enables the user to change the time range used in the page.
- Marking a *timeRange* as an optional Context in a component permits it to be independently updated from the rest of the views on a page. This permits multiple views on a page to each have their own time range.

TIP: If you don’t want the Time Range drop-down to appear on a page, remove timeRange from the context. Just be sure that the page is indeed static and does not need a time range input.

4

Queries

In Foglight®, queries are the preferred method for retrieving data from a data source. For example, you can set up a query to retrieve a list of objects that represent all agent instances running in a monitored environment whose host name matches the name of a particular *Host* object.

NOTE: You can also retrieve data in a more limited way using a Data binding object and you can use the context mechanism to get data from other views. It is also possible to use functions to return data.

The objects that appear in the Data menu choice under Configuration are those that are returned by root queries. A number of default Root Queries have been defined. These allow you to browse the common data objects. Using this approach you can build your own query that returns objects of interest to you and drag them on to your custom dashboard to create pages that meet your specific needs.

NOTE: You can build your own root queries. Simply enable the UI Query check box in the New Query definition editor.

These topics discuss how to work with query definitions and provide explanations of the choices a user encounters.

- Overview of Query Definitions
- Creating a Query in Foglight
- Parameters in Queries

Overview of Query Definitions

Queries let you select a set of data objects of the same type. Pages and views extract and use data from the queries for values and context settings.

Each query defines:

- The object type to be returned by the query
- The specific data source root
- The path to the data object (from the root location from which the query starts its search)
- Parameters that may be passed to the query

Optionally, a query can search from the root, drilling down until it finds objects of the desired type. In addition, it can restrict the objects it finds using conditions (or restrict them to the first *N* objects), it can sort the objects, and it can aggregate the objects into maximums, minimums, averages, and other aggregates.

Queries return a list of data objects of the same data type (such as Host or Alarm). For example, a query on the Host data type may return a list that contains all the available Host data objects from the data source.

Queries always return a list of data objects. Even when there is only one object found that satisfies the query, the result is a one-element list. A query that does not find any objects returns an empty list.

Just like Views, Relevant and Allowed Roles can be set on Query Definitions. These are taken into account when filtering Root Queries by role in the Data Browser and when using *Create dashboard* or *Create report*. For more information, see Roles in Query Definitions on page 67.

Creating a Query in Foglight

You begin the process of creating a query by clicking the Add button after selecting Queries in the drop-down list in the Module Contents pane. The **Add Query** dialog box appears with these choices:

- **Create a Blank Query**: Invokes the editor to create a new query. Creating New Queries on page 68
- **Copy an Existing Query**: Copies a query to a new module. Copying a Query on page 70
- **Derive from an Existing Query**: Creates a variant of an existing query with a different name, usually to change some but not all of the original's definition. Deriving a Query from Another Query on page 70

NOTE: For more information about these functions, see View Commands in the Definitions Area on page 34. For more information about configuring queries, see Query Definition Settings on page 58.

Query Definition Settings

Unlike views, a query definition does not have separate tabs for different groups of settings. All of the settings are on a single page.

Name

This is a required field. You cannot create a new query with the same name as another query in the same module. A warning icon (!) is displayed if this happens.

ID

Each query is tagged with an ID that is used as an internal identifier. Normally you would leave this field with its default setting, which is (*auto*).

UI Query

UI queries have the following features:

- Are marked as UI Query in the definitions pane when editing a query
- Must not have any required parameters
- Must have a relevant role set
- Have IDs that are set automatically, but you can set a more descriptive ID. This can be useful in certain cases. For instance, if the ID appears in a log message it is easier to identify the component if it has been given a descriptive ID.
- Appear in the Data browser, and in the Data tab when using *Create dashboard* and *Create report* as the starting point for exploring contained data.
- Act as a useful filter by domain to help users find data
- Act as a weak security mechanism.

Select the **UI Query** box to mark the query as a UI Query. UI queries (and their execution results) are listed in the *Data* browser and the *Data* tab when using *Create dashboard* and *Create report*. They provide a way to organize the data presented in those locations.

Hide Root

The **Hide Root** check box is only enabled if **UI Query** is selected. If selected, when the results of the UI query are shown in the Data Browser or the Data tab using *Create dashboard* or *Create report*, the first item in the results list is hidden. If the query results list does not contain one item, then this option is ignored. This is useful when you

know that the results of the query will always return one item and you want to save users having to expand the node corresponding to the one item in order to see its children. For example, if you have a query that returns the root of a data source, hiding that root is useful so that users can more easily get to the data that is really of interest.

If you have a Dashboard Designer or a Cartridge Developer role, you can see the entire data structure in **Dashboards > Configuration > Data > Management Server > All Data**.

Public

By default, queries are accessible from the module in which they are defined. If a query is marked as *Public*, it can be accessed from any module. For more information, see Public Entities on page 15.

Deprecated

Marking a query as Deprecated allows existing views that use the query to continue to work, but the query is not available for use in any new view that is being created. Legacy views continue to work, but new views cannot use this query.

Relevant Roles and Allowed Roles

Relevant Roles and Allowed Roles are used in UI queries by the filtering mechanism. For more information, see Relevant Roles and Allowed Roles on page 67.

Object Type

Each query operates on a given type of data source, for example, the Monitoring data source. The Data Source Type must be selected before you start editing. You cannot edit the Data Source Type while you are editing the query, as changing the type also changes other settings.

Required Parameters

This section of the query definition is used to specify any parameters that are required by the query and supplied at run time by Query bindings that modify this query.

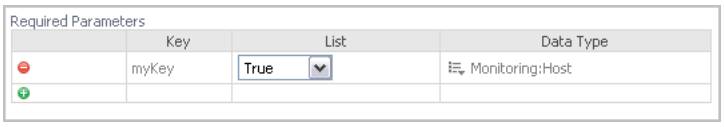
To add a Required Parameter:

1. Click the ⊕ button.

   A Required Parameter row is displayed with the following fields:

   - **Key** input box
   - **List** drop-down list
   - **Data Type** from the **Select Type** dialog box
2. Define a key, by which the parameter value can be referenced.
3. You can select **True, False**, or **Unknown** from the *List* drop-down. These options indicate if the data in the parameter is a list, not a list, or might be a list. If the parameter is a list, it must be a list of objects of the selected *Data Type*.
4. Select the data type from the **Select Type** dialog box. This is the type of value that the required parameter can have. For example, it can be a data type from the Monitoring data source, such as Host, or a common type like String.

Figure 30. Required parameters



In the preceding example, the *myKey* node is the name of the list of objects of the data type *Host*. A query always returns a list of data objects of the given type. Even if there is only one data object to return, the query returns a list with one item.

Object Type

The Object Type is selected in the **Create a Blank Query** tab as the first step of defining a new query. You select from a list of the available types for the query's Data Source Type, such as Host or Agent.

Root Reference

This section of the definition sets the root of the path from which the query searches for objects of the given Data Source Type. The From field can be chosen from data sources or parameters that you have set in *Required Parameters*. When Monitoring is chosen in the From field, the Path menu lists the entire Monitoring schema. A query search starts at the root of the object hierarchy, denoted by "/", or at the object declared in the required parameters.

Aggregations

A query returns a list of data objects. Instead of directly returning these data objects, the **Aggregations** settings creates new data objects that contain only aggregated values. For example, if a query returns a set of alarms, you can replace these data objects with an Alarm data object containing the maximum severity of those alarms.

NOTE: In most cases, the list contains only one aggregated data object.

Aggregation data objects are of the given data-object type, and they contain aggregated values of certain properties within the data objects. The aggregation types are:

- **Maximum**
- **Minimum**
- **Sum**
- **Average**
- **Weighted Average**
- **Count**

Unlike the other aggregation types, a count aggregation returns a Count data object with only two properties:

- **Value**: The count of returned data objects
- **Type Name**: The name of the type of data object being selected by the query and counted

Count is also different from the other aggregation types in how it handles a query that does not return any data objects. The other types will not create an aggregated object in this case. Count does create an aggregated object, with Value set to 0.

Creating an Aggregation

You can create more than one aggregation on the same query. If you do this, multiple aggregated data objects are returned in the results of the query: one for each aggregation, in the order specified. Leave the field blank if you do not require an aggregation on a property.

To add an aggregation:

1 Click the ⊕ button under **Aggregations**.

A new set of fields for an aggregation is added.

2 In the **Calculate** drop-down list select how you want the property field calculated.

3 In the **Property** drop-down list, select the type of property used in the aggregation.

4 Click the ⊕ button with the tooltip *Add Aggregation* if you need to define another aggregation.

5 Continue with Identifying Values on page 62.

If you aggregate on Metric properties, such as *cpuUsage* under Hosts, it creates a data object of the type Host, with a *cpuUsage* property, and that property has current, period, and history properties. Each of those last three (the *MetricValues* within the Metric) are aggregated, as follows:

- For a *max* aggregation, it aggregates the *max* property of the metric values, and stores it back into the *max* property in the aggregated *MetricValue*.
- For a *min* aggregation, it aggregates the *min* property of the metric values, and stores it back into the *min* property in the aggregated *MetricValue*.
- For an *average* aggregation, it aggregates the *average* property of the metric values, and stores it back into the *average* property in the aggregated *MetricValue*.
- For a *weighted average* aggregation, it aggregates the *weighted average* property of the metric values, and stores it back into the *average* property in the aggregated *MetricValue*. The count property of a metric is used for the weight. For example, if the collected metric is Response Time, its average should be calculated by weighting the various observed values by how many times each different value occurred. On the other hand, if the system is monitoring two hosts, a simple average of CPU usage is appropriate to guard against the case where the sample rates for collecting data on the two hosts are different. If sample counts were used to weight the average, the host that was sampled more frequently would skew the result.
- For a sum aggregation, it aggregates the average property of the metric values, and stores it back into the *sum* property in the aggregated *MetricValue*.
- The only situation in which each history object within the Metric is not aggregated is if the *timeRange* used for the Query uses RAW granularity, which means that each observation from the agent gets its own history row. In this case, the history lists from the various metrics being aggregated do not match up (in number, or in date/times), so there is no way they could be aggregated.
- The most likely properties that you will be interested in, when doing aggregations on Metric properties, are as follows:
  - For a *max* aggregation, the *period/max* under the Metric, which gives the maximum value for the whole *timeRange*.
  - For a *min* aggregation, the *period/min* under the Metric, which gives the minimum value for the whole *timeRange*.
  - For an *average* aggregation, the *period/average* under the Metric, which gives the average value for the whole *timeRange*.
  - For a *sum* aggregation, the *current/sum* under the Metric, which gives the sum of the values on the most current observations.

NOTE: You are not limited to using these values if something else is desired.

Removing an Aggregation

To remove an aggregation:

- Click ⊖ at the right side of the aggregation type.

Identifying Values

The expected use case for Identifying Values is as a label for a query that returns an aggregated result.

Within each aggregation (except for ones using the operation count), you can also create Identifying Values for the object returned by the query. This is a way to assign fixed string values to other properties within the data object that is created as the result of choosing an aggregation. For instance, if you created an aggregation of events to get their maximum severity, you might add Identifying Values that returns just the Name property of this object. The identifying value can then be used as a label for the row of a table that presents this information.

Adding an Identifying Value

Identifying Values settings only appear after you have created an aggregation.

To add an Identifying Value:

1 Click the ⊕ button under **Identifying Values**.

A new set of fields for an Identifying Value is added.

2 Select the **Property** (relative to the Object Type) and the **Text Value** to be assigned to that property.

Removing an Identifying Value:

To remove an Identifying Value:

- Click the ⛔ button located right of the Identifying Value.

Filter Results Based on Top N

The Filter Results Based on Top N property lets you restrict the results of a query. You can restrict the results to the first N or, conversely, exclude only the first N. For example, you may only want all but the first five items of the results from the query. This function is enabled by selecting the **Filter results based on top N?** check box.

The **Value of N** must be a number. You can set this using one of two options:

- **Specific Value**, where you enter a number greater than 0.
- **From Parameter**, where you can select a Required Parameter from the drop-down list and use the value.

> **NOTE:** You can only use a parameter if it returns a number.

The **Results to Include** lets you restrict the results using one of two options:

- **Only the Top N**, where only the first N results are returned.
- **Everything except the Top N**, where everything except the first N results are returned.

Selecting **Everything except the top N** may be useful in conjunction with an aggregation operation where you use one query to get an average of the *top N*, and you want to use another query to get an average of the rest of the data that are not in the *top N*.

You would almost always use Order By when you use Filter Results Based on Top N. Otherwise you would not know in which order the results would occur, and thus it would not be clear which results would be included in the *top N*.

Order By

Order By keys control the sorting of the data objects returned by the query.

> **NOTE:** Sorting is applied before Filter Results Based on Top N is applied.

You can create multiple keys. Order By sorts on the first key, then on the second key, and follows the sequence. Order By sorts only on a second key if the first key has data objects in which the values were identical. In this case Order By sorts the data objects containing those identical values based on the second key. This continues as further keys are specified.

There are three types of views that can use the query's sort order: drop-downs, row-oriented tables, and radio-buttons. As a result, sorting within a query is usually only needed when Filter Results Based on *Top N* is also used.

NOTE: Row-oriented tables also have their own sorting capabilities. If used, these settings override the query's sort order.

If you are sorting on a metric property, such as *cpuUsage* under Hosts, the algorithm automatically sorts on the metric's current/value property.

Adding an Order By Key

To add an Order By key:

1. Click the ⊕ button under **Order By**.

   A new set of input fields is added.
2. Select the **Path** from the drop-down tree.
3. Select **ascending** or **descending**.

Removing an Order By key

To remove an Order By key:

- Click the ⊖ button.

Where

The Where property restricts which data objects are selected by the query, based on criteria such as property values in the data objects.

NOTE: In SQL, the property and its parts are also called Where clauses.

Adding a Where Clause

To add a Where clause:

1. Click the ⊕ button under **Where**.

   A popup appears, displaying the following types of conditions:

   - **Comparison**: Compare by *Path*, *Specific value*, or *From Parameter with Path*. Comparisons can be one of *equals*, *not equals*, *is less than*, *is less than or equals*, *is greater than*, *is greater than or equals*, *like*, *matches* or *in*.

     NOTE: WCF queries can evaluate comparisons in which the left side value is a list and the right side is not a list. The Evaluator iterates over all items in the list and compares each element of the list with the right-side value.
   - **Is Set**: Choose a path to check if it is set.
   - **Sub Type is**: True if the sub type of the object is as specified.
   - **And**: Logical operator whose operands may be any of the types in this list
   - **Or**: Logical operator whose operands may be any of the types in this list
   - **Not**: Logical operator whose operands may be any of the types in this list

2 Select a type and the fields that define the condition are displayed.

Any condition evaluates to either true or false. The conditions in the Where clause are evaluated for each data object found by the query. That data object is included in the final results only if the condition evaluates to true.

Removing a Where Clause

To remove an entire where clause:

- Click the button to the right of the section you want to remove.

Conditional Types

You can configure the following different types of conditions.

Comparison

This is the most common type of condition. A comparison consists of:

- A drop-down tree for the Path
- A drop-down list box for an operator
- A value that is used for comparison

The path on the left of the operator is compared to whatever is selected on the right of the operator, using the selected operator.

The operator can be:

- = (equals)
- != (not equals)
- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater than or equal to)
- like
- in

For the '=', '!=' and 'like' operators, the case insensitive attribute applies (the corresponding check box is enabled). Note that the case insensitive attribute will only be used if *both* the right hand and left hand values are strings.

The comparison value is set in one of two ways:

- If you select the **Specific Value** option, type the desired value into the input field.
- If you select the **From Parameter** option, you can select one of the Required Parameters (that are defined in the query already) from the drop-down list.

You can use the value of the parameter. You can also drill down to a lower-level path in the parameter's type by selecting a node from the **Path** drop-down tree.

For example, if you are selecting Hosts, and want to only select one host named *MainServer*, you would enter the path of name, the operator =, and the specific value *MainServer*.

When the *in* operator is used, the value on the right must be a list. This can occur when parameters are used. The comparison is true for a given data object if the value in the **Path** field is contained in the list specified by the **From Parameter** field on the right.

The like operator is used for wildcard matching, and behaves exactly as it does in standard SQL. The pattern of characters and wildcards in the right field are compared against the entire value of the entry in the Path field.

The wildcards are:

- Underscore (_): represents any one character.
- Percent (%): represents any string of 0 or more characters.

A match is found only if the pattern is a complete match against the Path value. For example, the following patterns will all match MachineOne:

- `M%One`
- `%chi%`
- `Ma_hine_ne`

But `chi%O` is not a match because it only matches a part of MachineOne.

Paths are allowed to have several levels, (such as events/name under Hosts), if an appropriate comparison can be made with the results.

The <, <=, > and >= operators can be used with numbers, strings or dates. If the Path on the left of the comparison evaluates to a number, and a Specific Value on the right of the comparison is a string, the string is treated as a number. If it cannot be converted to a number, the comparison will evaluate to false.

Is Set

Is Set requires a single Path selected from the drop-down tree of data objects. It evaluates to True if the value of that path is not empty (not null) and is not an empty list (in the case when the property always evaluates to a list).

In the rare case where the path points to a list of lists, the Is Set is True if at least one of the sub-lists is not empty. For example, if the elements the query is selecting are WebApplications, and you use *Is Set* with a path of *appServers/slowestRequests*, the condition is True for a given Web application if it contains at least one AppServer that contains at least one request in its slowest requests list.

Sub Type is

Sub Type requires a type to be selected from a drop-down list of types. The listed types are the sub types of the selected Object Type for the query. If there are no sub types of the Object Type, you receive an error message if you try to select this type of condition. It evaluates to true if the object is of the selected type (or one of its sub types). For instance, if you are selecting with an Object Type of Host, and you use a Sub Type is condition with type *WindowsHost*, only the Windows® Hosts are selected.

NOTE: For this usage, you could just have selected WindowsHost as the Object Type and embed this condition in a Not condition, to select all Hosts that were not Windows Hosts.

And

The And condition is true if all of the conditions below it are true, and is false if at least one of them is false. Use And to apply more than one condition.

To add a condition:

1. Select **And**, and click the ⊕ button below it.
2. Create the conditions as required.

Or

The Or condition is true if at least one of the conditions below it is true, and is false if all of them are false. Use Or to apply more than one condition.

To add a condition:

1 Select **Or**, and click the ⊕ button below it.

2 Create the conditions as required.

Not

Use Not to reverse the sense of a single condition. The Not condition is true if the condition below it is false, and false if the condition below it is true.

To add a condition:

1 Select **Not**, and click the ⊕ button below it.

2 Create the one condition as required.

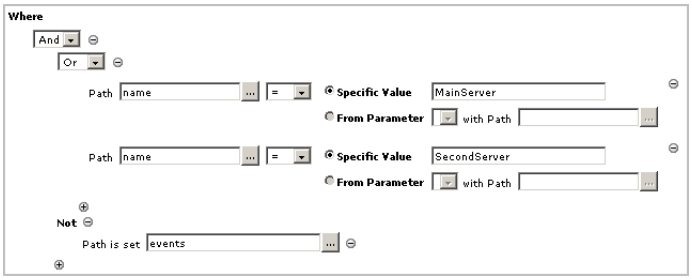
Combining And, Or, and Not Conditions

By using nested combinations of And, Or and Not, you can create complex conditions, such as including a Host if:

- Its name is *MainServer* or *SecondServer*.
- There are no events under it.

The following example displays how the Where section of a query is set.

Figure 31. Query conditions



The And clause contains the Or and the Not clauses. The Or clause itself contains the two comparison clauses, and the Not clause contains an Is Set clause.

Comments

This is an optional field intended for use by developers. It is automatically filled in if you are copying a query.

Context Help

Use this field to tell end users about the purpose of this query. To make the help text available to end users the query must be marked as a UI Query.

If you create a Portal Container by choosing Create Dashboard from the General tab of the action pane, the Data tab becomes available in the action pane. All queries appear here. Context help for the query appears as a tooltip when you hover the mouse pointer over the query's name. Similarly, if you entered text in the Help area of the **Create Dashboard** dialog box, it appears when you hover the mouse pointer over its entry in My Dashboards in the navigation pane.

Relevant Roles and Allowed Roles

Relevant Roles are used to act as filters, showing the views a user is interested in and not showing the others. If some but not all Relevant Roles are set on a view, then in some situations the browser interface may not show that view to a user who does not have that role. Even though the view does not show in, say, a list of dashboards, the user can still access the view if there is a page accessible to the user that has a link to the unlisted view.

Allowed Roles act as a security mechanism. If some but not all Allowed Roles are set on a view, then only users for whom at least one of their roles is also an Allowed Role on the view can see or in any way interact with that view.

Roles in View Definitions

Each view can be marked as having zero or more Relevant Roles, and zero or more Allowed Roles. Users are assigned roles and these are matched against the Relevant Roles and the Allowed Roles for a view. This restriction does not apply if the user has the appropriate permission or if the view is in the user's personal module.

- Each user has Allowed Roles (the roles they are allowed to be in) and Preferred Roles (roles for which this component was designed to be useful). Relevant roles and Allowed roles contain the same list of items as choices.
- If a view has no Relevant Roles marked, it is assumed to be relevant to all roles. If a view has no Allowed Roles marked, it is assumed to be allowed to be used by all roles.

Roles in Query Definitions

Roles are taken into account when filtering Root Queries by role in the Data Browser when using *Create dashboard* and *Create report*.

Each query can be marked as having zero or more Relevant Roles, and zero or more Allowed Roles. This is related to user roles. Users are assigned roles and these are matched against the Relevant Roles and the Allowed Roles for a query.

If a query has no Allowed Roles marked, all roles are allowed to use this query. If some but not all Allowed Roles are set on a query, then only users for whom at least one of their roles is also an Allowed Role on the query can see or in any way interact with that query. This restriction does not apply if the user has CHANGE_SYSTEM_MODULES permission (as determined by the *AuthorizationService*) or if the query is in the user's personal module.

If a query has no Relevant Roles marked, it is assumed to be relevant to all roles. In some situations, the GUI may not show a query as a choice to a user if the user's roles do not match the Relevant Roles of the query. However, there is an option in those situations to see all queries that are allowed, regardless of Relevant Roles.

Module-Level Roles

Roles can be applied globally to all the views and queries in a module by editing its properties.

To edit module-level roles:

1. Select **Dashboards** > **Configuration** > **Definitions** in the navigation panel.
2. Click the triangular button at the right of a module name in the Module List pane.
3. In the drop-down list that appears, click **Edit**.
4. In the **Edit Module** dialog box, select the appropriate **Relevant Role(s)** and **Allowed Role(s)**.

Sequence of Evaluation

In a complex query, the selected data objects are subjected to the different parts of the query in the following order:

1. The data objects are restricted based on **Where**, if specified.
2. They are then sorted based on **Order By**, if specified.

3 Some are then selected based on Filter Results Based on Top N, if specified.

4 The remaining data objects are aggregated (if aggregations are specified).

NOTE: For more information about Queries, see the Web Component Tutorial.

Creating New Queries

The **Add Query** dialog box lets you create a new query or a copy of an existing query.

To create a new query:

1 Select **Configuration > Definitions** in the navigation panel.

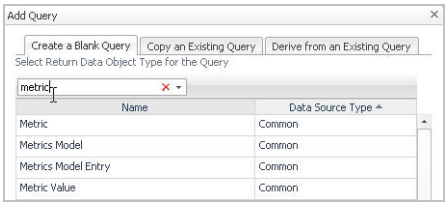
2 In the Module List pane, choose the module in which you want to work. If you are creating queries for your own use, select **My Definitions**.

3 In the Module Contents pane, select the **Queries** option from the drop-down list.

The **Add Query** dialog box appears.

4 On the **Create a Blank Query tab,** select a **Data Source Type** that specifies the type of object to be returned by the query. You can use the search box to narrow the choices.

Figure 32. Matching object types



5 Click **OK**.

A **New Query** tab opens in the Edit mode of the Definitions pane.

For queries, all settings are on this pane. There are no sub-tabs as there are for more complex things, such as *Views*.

The edit pane fills with the query editor, which contains all the fields that can be used to construct a query. You must fill in only the fields that are marked by an exclamation point icon (❗) on the right.

6 Type the name you have chosen for your query in the *Name* field.

Now that the query has been named, it can be referenced by that name when you want to use it.

7 By default, the *ID* field's value is <*auto*>. You may want to give it a more descriptive name, which could be useful for troubleshooting.

8 Select **UI Query** if the query is to appear in the data browser.

9 If it is a UI query and you want to hide the root, check **Hide Root**.

10 Select **Public** if you want to share the query.

11 Ensure that **Deprecated** is cleared. This check box is used to flag outdated queries.

12 Select **Relevant** and **Allowed** roles for this query. For more information, see Roles in Query Definitions on page 67.

13 Add any **Required Parameters** needed by the query.

These are values that are supplied at run time, usually from a context input, that the query will need. You need to specify a Key by which the value may be referenced, whether it is a List (True, False, or Unknown), and choose its Data Type from the **Select Type** dialog box.

14 Select a data source in *Root Reference* from the choices in the *From* field. Typically, its value is Monitoring, or if you want to have the search start from a required parameter, choose it from the *Parameters* node.

15 Select the *Object Type* by clicking the drop-down arrow at the right of the *Path* field.
Choose the object from the drop-down list of data objects. If you have chosen the Monitoring data source, this list is a replica of the nodes in the Data browser under **Management Server > All Data**. For any other data source, it is the list of available entities for that source.

A list of all the objects known to the running Foglight instance appears.

16 Objects are grouped at some level down from the root. Choose the object you want from the drop-down list in the *with Path* field. You might need to set the *Max Search Depth* field as well.

17 You can have the query return aggregate values by setting *Aggregations*.

18 You can filter results by specifying a value for *N*, either by giving it a specific value or by declaring a parameter that returns a value. You can also choose to include *Everything except the top N* for the case where you want to show the top *N* somewhere and the rest somewhere else.

19 You can sort the returned result by choosing a data object's property, such as its name.

20 You can use a *Where* clause to use a condition to limit the returned objects to the ones that match the criterion.

21 You can test the query by clicking the **Test** button at the top of the Edit pane.

The **Query Results** dialog box appears, showing the name or names of the servers returned by the query and, in this case, the *Data Type* of the object, which is Host. The *Value* field is empty because there is no numerical value associated with objects. Simple types have values, objects do not.

> **IMPORTANT:** You can see that the query did return a list of objects by clicking the expand icon (+) at the left any host name in the Query Results dialog box. When you do, you'll see that some properties of the host, such as its topologyObjectId and topologyObjectVersionId, have numerical values, while others, such as agents, which are themselves objects, do not. The Query Results dialog box allows you to see all the properties of the object, which is useful if you want to determine their names for use later on.

> **IMPORTANT:** The topologyObjectId and topologyObjectVersionId properties mentioned in the previous note should not be used to select a particular object. If you do need to use a specific object in a custom view during a running session it is better to set an additional context value of type Data, type in a name for the Key, and navigate to the particular runtime object in the Context Entry dialog box This is a much more efficient way of directly accessing the object rather than causing the system to search all objects until an ID match is found.

22 It is recommended that you fill in the **Comments** and **Context Help** text boxes.

23 Click **Save** to save the query.

Parameters in Queries

Since a Query can have parameters, it can be made aware of bindings. Their values replace the specified *From Parameters* in the query. The Query Binding parameters must exactly match the Required Parameters.

The query attributes that can be replaced by parameter values are the query's Root Reference Path, the Filter Results Based on Top N value, and a comparison condition in a Where clause. The Root Path can only be replaced by a parameter that is a data object or a list of data objects. The value of the parameter becomes the root object for the query execution.

You can add additional path information under a parameter used for the Root Path. In this case, the parameter value is assumed to be a data object, but the value used is the given path into that data object.

For more information about using parameters, see Query Definition Settings on page 58 and Deriving a Query from Another Query on page 70. For information about aggregations see Creating an Aggregation on page 60.

Summary of Creating a Query

Creating a query extracts data from a data source. Creating a query involves defining the following elements:

- The type of objects that you want to select.
- Any parameters that you want to pass to the query.
- The selection criteria, or the *where* clause.

If parameters are passed to a query, you can use them to compare the properties of selected objects to refine the *Where* clause.

Copying a Query

Copying is a fast way of creating a new query. It is also a way to create a modified version of a System query. You can copy any query, including your own User query, a System query, or a query created by another user.

You cannot create a copy of a query and give it the same name as another query in your module. If you enter a name that is already in use, an alert icon (❗) is displayed beside the field and the **Save** button is disabled.

To copy a query:

NOTE: There is no copy button for a query shown in the Definition pane. Instead, select the New Query button and the Add Query dialog box is displayed.

1. Select **Configuration > Definitions** in the navigation panel.
2. In the Module List pane, choose the module in which you want to work. If you are creating queries for your own use, choose **My Definitions**.
3. In the Module Contents pane, click the **Queries** tab.
4. Click **Add**.

   The **Add Query** dialog box is displayed.
5. Select the **Copy an Existing Query** tab.
6. From the list, select a module that you want to copy.
7. Click **OK**.

   The query definition is displayed.

NOTE: To save this as a new query, you are only required to enter a name in the Name field. You can modify the other settings at a later time.

Deriving a Query from Another Query

Deriving a query from another query allows you to create a variant of an existing query. You can derive from any query, including your own User query, a System query, or a query created by another user. You can even derive a query from another derived query. The query from which a derived query is made is called its base query, and is permanently linked to it.

You cannot create a derived query and give it the same name as another query in your module. If you enter a name that is already in use, an alert icon (❗) is displayed beside the field and the Save button is disabled.

A derived query may extend its base query's list of Required Parameters, and replace its Filter Results Based on *TopN*, Aggregations, Order By and Where sections. If the derived query adds Required Parameters, their numerical order comes after that of the base query.

A simple example of where you might want to use derived queries is a base query that selects a group of Events, and a derived query that aggregates those events into a maximum severity.

In the editor, normally you are not allowed to edit or replace the Filter Results Based on Top N, Aggregation, Order By or Where settings in a derived query if those settings are already established in the base query. However, it is possible to get into a situation where both the base query and the derived query contain settings for any of these properties as follows:

- Start with a base query with the property, for example Aggregations, that is not set.
- Derive a query from it, and set Aggregations on the derived query.
- Go back to the base query, and set Aggregations on it.

In this situation, both queries have settings for Aggregations. The query editor will display warnings on both of these queries about that fact. If you edit the derived query again, you will be allowed to modify the Aggregations setting. However, if you remove that setting, it reverts to using the base query's Aggregations setting, and that setting is no longer editable.

NOTE: Parts of the derived query that are inherited from the base query are displayed only in the Query Editor. They are not editable.

To derive a query:

1. From the navigation panel under Dashboards, click **Configuration > Definitions**.
2. In the Module List pane, choose the module in which you want to work. If you are creating views for your own use, choose **My Definitions**.
3. In the Module Contents pane, click the **Queries** tab.
4. Click **Add**.

   The **Add Query** dialog box is displayed.
5. Select **Derive from an Existing Query** and click **OK**.

   The query definition is displayed.

NOTE: To save this as a new query, you are only required to enter a name in the Name field. You can always modify the other settings at a later time.

Editing a Query

Follow this procedure to edit a query. Depending on your permissions, you may not be able to edit a System query or a query created by another user.

To edit a query:

1. In the Module Contents pane, click the **Queries** tab.
2. Select the User query from the Modules Contents pane.
3. Select **Edit** from the Definitions pane.

   The query definition is displayed.

Deleting a Query

Follow this procedure to delete a query. You can only delete your own User queries. You cannot delete a System query or a query created by another user.

To delete a query:

1. In the Module Contents pane, click the **Queries** tab.
2. Select the User query from the List frame.
3. Select **Delete** from the Definitions pane.

   A dialog box appears, asking you to confirm the deletion.

Legacy Custom Query

Use Functions instead of Legacy Custom Queries.

In past releases, a Legacy Custom Query was used to return data that may or may not be bound to a specific data source.

Correctly formed Groovy scripts automatically appear as Legacy Custom Queries.

Web Component Framework examples:

- Time Range chunking
- Remove duplicates

Foglight examples:

- Check User Permissions

5

Functions

Function definitions let you return data of a declared output type given an expected input. They allow users to easily generate possible return values in a way that can be easier than using queries.

The four types of functions are Java™, Map, Script, and Script with Map. They are created in the **Functions** tab of the Definitions editor. Similar to other type of definitions, Functions can be added, edited, removed, or copied. They also support the common entity attributes. For more information, see Definitions and Entities on page 14.

NOTE: It is recommended that you obtain the assistance of Quest Sales Engineers or the Professional Services team before beginning to work with Functions.

NOTE: A handy way to access the API for Foglight is through the browser interface. Under Dashboards in the navigation panel, choose Administration > Tooling > Script Console. In the Query and Scripting Service Tool, click the icon for Script Help, which is at the top right of the Script box. Alternatively, you can type help(server) and click the Run button below the Script box. In the Result area, click Topology Service to see the Foglight® methods.

NOTE: When using Functions it is recommended that you write unit tests to ensure correct ongoing behavior. For more information, see Unit Testing Entities on page 15.

Java Functions

A Java™ Function definition is used to execute a Java method and return a result. It can optionally have parameters to the Java method. Java Functions are meant to be used to call user-defined Java code that has been deployed to the Foglight® server.

Using the editor, these attributes can be specified:

- The fully-qualified name of the class containing the Java method to invoke. In order for the class to be loaded, it needs to exist on the server's classpath or in a Jar file that WCF will be notified about when loaded or unloaded. In Foglight, you would normally accomplish this by adding the jars with your classes as a *wcf-task* component in your cartridge.
- The name of the method to invoke. The method must be public, static, have one parameter of type *FunctionHelper* and not return void.
- The parameters (which are named, have a type and declare whether they are a list or not)
- The output type (including whether it is a list or not)

When a Java Function is executed, the parameter values are evaluated (if they are bindings) and passed into the method and the result is returned. If the declared output type is not a list and the result is not of the declared type, null is returned instead. If the declared output type is a list, then the result is checked to ensure it is a list, but the list items are not checked to determine whether they are of the declared output type.

NOTE: It is possible that a Java Function can be called by multiple threads so implementations need to be thread-safe.

Example of a method suitable for use as a Java Function:

This is the signature of the method for the Check Permission Java Function definition available in the **Dashboard Support > Common** module:

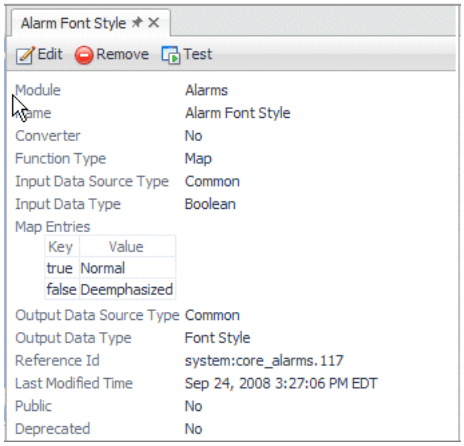
```
public static boolean checkPermission(FunctionHelper helper)
```

Map Functions

A Map Function definition consists of a set of keys and their corresponding mapped values. The function expects a key as its input value and the value corresponding to the key is returned.

An example of a Map Function:

Figure 33. Map function example



Using the editor, these attributes can be specified:

- Input type
- Map entry keys and values
- Output type (including whether or not it is a list)
- Map default value

When a Map Function is executed, the keys are checked (and evaluated if they are bindings) in order, and when one result equals the input, the corresponding Map value is returned (if it is a binding, it is evaluated and then returned). If no matching key is found, and the Map Default Value is specified, the function returns this default value. If no matching key is found, and the Map Default Value is not set, the function returns `Null`.

The editor ensures that all the keys of the map are the same type as the input type and that all the values of the map are the same type as the output type. Also, the map values must match the declaration of whether or not the output is a list.

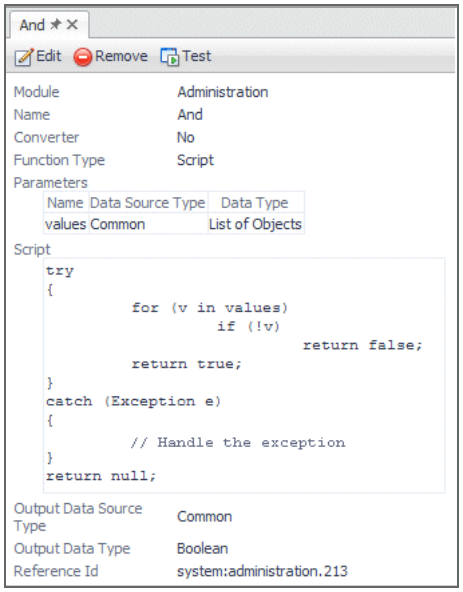
One map entry having a null key is allowed. To specify a null key, simply do not set the map key.

Script Functions

A Script Function definition is used to execute a script and return a result. It can optionally take inputs to the script.

An example of a Script Function:

Figure 34. Script function example



Using the editor, these attributes can be specified:

- Parameters (which are named, have a type, and declare whether or not they are a list). These parameters are used by the function and must be set in the binding that invokes the function.
- Script text. The Groovy code is entered here.
- Output type (including whether or not it is a list). This is the return type for the Groovy function. Normally this is a String status message.

When a Script Function is executed, the inputs are evaluated (if they are bindings) and passed into the script. The script is executed via the ScriptService and the result is returned.

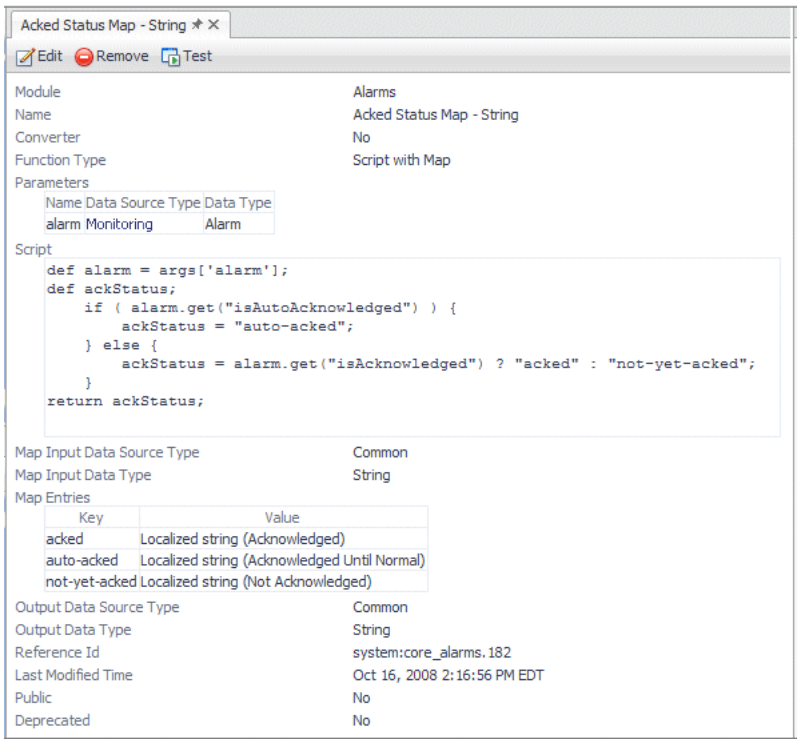
NOTE: The ScriptService is used by the Web Component Framework for evaluating Function definitions of type Script or Map with Script.

Script with Map Functions

A Script with Map Function definition is used to execute a script that accepts a map key and the corresponding map value is returned. The script can optionally take inputs.

An example of a Script with Map Function:

Figure 35. Script with Map Function example



Using the editor, these attributes can be specified:

- Inputs (which are named, have a type and declare whether or not) they are a list
- Script text
- Map input type
- Map entry keys and values
- Output type (including whether or not it is a list)
- Map default value

When a Script with Map Function is executed, the inputs are evaluated (if they are bindings) and passed into the script. The script is executed via the ScriptService. The script result is then checked against the map keys (which are evaluated if they are bindings) in order, and when one result equals the script result, the corresponding Map value is returned (if it is a binding, it is evaluated and then returned). If no matching key is found, and the Map Default Value is specified, the function returns this default value. If no matching key is found, and the Map Default Value is not set, the function returns `Null`.

The editor ensures that all the keys of the map are the same type as the map input type and that all the values of the map are the same type as the output type. Also, the map values must declaration of whether or not the output is a list.

Using Functions

Similar to queries, Functions are called as a binding, in this case the Function binding. For more information, see Bindings on page 80.

Functions as Converters

A function can be marked as a Converter. A Converter is a function that takes data of one type as input and converts it to data of another type.

A registry is automatically kept of such Converters and components can request a Converter for an input/output type when needed.

Only functions that take a single (non-List) input and that have a single (non-List) output are allowed to be Converters. Only one Converter can be registered for a given pair of input/output types. If there are many Converters that can handle an input/output type pair, only the most recently loaded one will be registered and a warning message will be logged. Converters are loaded upon start up of the system, or when a module is imported/deleted, or when a Function is edited and saved.

Caching results of Functions

If a Function is referred to in Additional Context, the result value of the Function may be calculated many times (due to how View Components are initialized).

If you know that your Function will output the same value given the same Time Range and Function parameters, then you can avoid the redundant calculations by checking the Cache Results checkbox in the Function editor for the Function Definition in question.

Information available from a Script, Script with Map or Java Function

From within a Java™, Script or a Script with Map Function, a special `FunctionHelper` class is available for accessing additional information. See the `javadoc` for that class for more details on what information is available.

For Java Functions, `FunctionHelper` objects are passed as a parameter to the specified Java method.

For Script or Script with Map Functions, the `FunctionHelper` class is accessible through the reserved script parameter, `functionHelper`. To make things easier, the argument values of function parameters can also be accessed by directly referencing the name of the parameter. The reserved script parameters are also available:

- `resourceBundle`: The Java Resource Bundle for the current locale with strings loaded from the module's file `strings.properties` (uploaded using the **Files** tab available under **Configuration > Definitions**).
- `specificTimeRange`: The current `SpecificTimeRange`. This value should always be passed when you access property values of any time-sensitive data object. For example:

  To get property `b`, instead of writing:

  ```
  x = a.b;
  ```

  Write the following:

  ```
  x = a.get("b", specificTimeRange);
  ```

Since these parameter names are reserved, they cannot be used as the name for declared Script or Script with Map Function parameters. The reserved script parameters should be referenced using the same notation used for other script parameters.

For Java Functions, these are available via the corresponding getter methods of the *FunctionHelper* parameter. The table shows the methods in *FunctionHelper*:

Table 15. FunctionHelper methods

Return Type	Method Name	Description
WritableData Object	`createDataObject( String typeName, String storage, String objectId)`	Creates a writable data object.
Map<String,?>	`getArguments()`	Returns a Map containing the argument values for the function's parameters.
Locale	`getLocale()`	Returns the user locale.
ResourceBundle	`getResourceBundle()`	Returns the ResourceBundle for the current Locale, loaded from the *strings.properties* file in the module to which the function belongs.
SpecificTimeRange	`getSpecificTime Range()`	Returns a *SpecificTimeRange* whose start/end dates are computed based on the *TimeRange* when the function was invoked.
Calendar	`getTimestamp()`	Returns the current timestamp.
String	`getUser()`	Returns the name of the current user.
String	`getUserModule Name()`	Returns the name of the user module.
String	`getViewId()`	Returns the ID of the view for the current view component.
Object	`invokeFunction( String fqId, Object... arguments)`	Invokes a function. Used for invoking a function within another function.
List<?>	`invokeQuery( String fqId, Object... arguments)`	Invokes a query.

Invoking a Function from a Function

Support is available for invoking a function from the script text of a Script or Script with Map Function, or from a Java™ Function. This is achieved by using this method of FunctionHelper:

```
public Object invokeFunction(String fqId, Object... args) throws
FunctionInvocationException
```

Where:

- *fqId*—Is the Reference ID of the function to invoke. It must not be null.
- *args*—Are the arguments to pass to the function.
- A *FunctionInvocationException* is thrown when there is an error invoking the function.

For Script or Script with Map Functions, this method can be accessed via the *functionHelper* hidden parameter.

For Java Functions, the Java method for that function must have a parameter of type *FunctionHelper*, as previously described, so that the invoke method can be accessed via that parameter.

Example script that uses this functionality (where *user.foglight.1* is a function that takes two integer inputs):

```
functionHelper.invokeFunction("user:foglight.1", 6, 9);
```

NOTE: Due to the way the Groovy scripting environment works, if you are calling a function that has no inputs, you can simply not provide the inputValues parameter at all (as opposed to providing null).functionHelper.invokeFunction("user:foglight.3");

Invoking a Query from a Function

Support is available for invoking a query from the script text of a Script or Script with Map Function or from a Java™ Function. This is achieved by using the method `FunctionHelper.invokeQuery()`.

How this method can be accessed and used is the same as the `FunctionHelper.invokeFunction()` method used for invoking a function from a function (as described in the previous section).

Time Range used for testing a function

When testing a Function through the Web Component Framework's Function editor, be aware that the time range used by the Web Component Framework to evaluate a Function is a fixed (built-in) time range, which is from Jan 01, 2000 to Now (the time you started your test).

CAUTION: Testing is not supported using Internet Explorer 6

6

Bindings

This section describes the configurable properties for pages and views. Configurable properties determine which data items actually display in a View. Properties are set at design time. They are typically simple types, such as numerical and string values. The properties for each View are set in its Configuration tab. Examples are a parameterized title, formatting, and the source of its data values or metrics.

Each configurable property has a specified data type. There are simple types and binding types. Simple types are types such as strings or numbers. For example, the font size property is a number. Simple types are typically for formatting or labeling properties.

The binding types are used to retrieve the dynamic data in a view. The values of these properties are extracted from data objects at run time. For example, all of the monitored hosts can be extracted from the Host data object.

Simple Types

These types are set using the property editors available in the Configuration tab in Definitions.

The following table lists the simple types that are used to configure certain properties:

Table 16. Simple types

Type	Description
Boolean	A fixed value of true or false. The Selectable property uses this type.
Color	An RGB color specification. A color coordinate can be entered or a color can be chosen from the color palette.
Data Object Property	A context key for a data object property.
Data Object Type	The type of a database object.
Date	A Date object. (See Date on page 91.)
Enumerated value	A selection from a fixed list of options. Properties that use this type include: • Header Alignment (options are *Vertical* and *Horizontal*) • Sort Order (options are *asc* or *desc*)
Error Renderer	The name of an error renderer, which displays a value indicating *error* if the attempt to evaluate in the associated bound data results in an error. See Renderers on page 106.
ImageReference	A reference to an image and to its size. The reference can be indexed or non-indexed.
Localized String	A localized string. These strings can be internationalized. The displayed string can be made to be dependent on the locale.
Null Renderer	The name of a null renderer, which displays a value indicating *no data* if the values in the associated bound data are null. See Renderers on page 106.
Number value	Any fixed numeric value, such as 123.45. Commonly used for integer values. Properties that use this type include: • Location • Size • Column or Cell Width (in pixels)

Table 16. Simple types

Type	Description
Property	A Property value in a data type.
Static Value	A non-localizable string value used as a label.
String	A fixed text string. Commonly used for IDs, which are used for flows, filtering and sorting. It can also be used for simple text labels. Properties that use this type include: • The row-oriented table's `Columns > Column > ID` property.
TimeRange	A time range has a start and end time. Predefined time ranges may be tied to the calendar or the current time. Custom time ranges are open without restrictions.
Type	A built-in or user defined data type.
View Reference	A reference to a view.

Binding Types

There are several types of bindings. Each one provides a way to define a value, which, depending on the type and the way it is used, may or may not be bound to data from the underlying system. A binding can evaluate to simple objects such as strings, complex objects, lists of objects, or lists of lists of objects.

The following table lists the available types of bindings. The concepts of parameters and context, which are referred to in some of the descriptions, are explained in Parameters in Queries on page 69 and Context Tab on page 51.

Table 17. Binding types

Type	Description
Context	A value from the Context, and can select parts of the context using a path (For more information, see Context Tab on page 51.).
Data	A bound data value or a list of data values directly, without using a Query. It is always bound to a specific data source. Because it is bound to a specific data instance, it can be used to eliminate an expensive query search (see Data on page 90).
Function	One of the available functions that is to be invoked.
Icon	An icon for display. If no Icon Renderer is specified by the user in the Icon, a default Icon Renderer is used to display the icon at Normal size (see Icon on page 87).
List	A list creation method: creates a list either from individual values, or by merging existing lists. (See List on page 91)
Query	A mechanism to calls a named Query to retrieve bound data, and can select parts of the Query's results using a path and other arguments (see Query on page 85).
Rich Text Template	A text template. Like the String Template, only it permits a restricted set of user-entered XHTML. It allows parameters to be set, while Rich Text does not (see Rich Text and Rich Text Template on page 88).
String Template	A simple static text string (such as a String value), but can also perform more complex tasks using parameters (see String Template on page 88).
Theme	A theme based on a Component, Style and Value. For more information, see Theme on page 87.
Writable Data Object	A specific data object, chosen from the available data source types. (For more information, see Writable Data Object on page 83.)

Details of each Binding

The following types of bindings are available to generate data for views:

- Context
- Writable Data Object
- Data Object Property (and Property)
- Icon
- List
- Localized String
- Query
- Rich Text and Rich Text Template
- String Template
- Theme
- Return Types

Context

Context bindings in a configuration can access any value in the context.

NOTE: Context cannot have parameters.

The following table describes the properties of a Context:

Items shown when **Show Advanced** is true are marked with an asterisk (*).

Table 18. Advanced context binding properties

Property	Description
Key	The name by which this item in the context may be referenced. When context objects are defined, a name key is part of the definition, so normally that value is chosen for the key. Access to the object is managed through its name key.
Show Advanced	When enabled, the following propertiesare avilable: Treat as Type, Return Type, Unit Property Name, and Time Range To Use.
Treat as Type	In the *Treat as Type* field, you can perform a cast to any known subtype of a data element. This is useful when you know that an object is of a particular sub-type but it is passed through an object structure that specifies a more generic type. This happens in cases where the less specific type does not have the properties that you want to access. You can choose to specify one of the allowable sub-types of the object's base type. For example, if the context selection is a host, you might choose to restrict it to an AIX_Host.
Path	Context inputs are declared to be of a definite data type. Path is the location of the desired property within the specific data object hierarchy. The **Context Entry** dialog box displays the actual names of the data-object properties, and not the localized names.
Return First Object in List	If you check Return First Object in List, the first element of the first list that is encountered is set to be a single item. For example, if the context element itself is a list then it uses the first element of the list, instead of the whole list. The optional Path is then applied to that first element. If, on the other hand, the last element is a list and there are no other lists in the path before it, then the first element of that last list is returned. If there are two or more lists in the path, only the first one is reduced to its first element.

Table 18. Advanced context binding properties

Property	Description
Renderer	See Renderers on page 106.
Return Type	Context can return ten possible types of data: • Localized Value • Value • CountPerItem • Count All • Unit • Units • Localized Property Name • Localized Property Names • Localized Type Name • Localized Type Names For more information, see Return Types on page 91.
Unit Property Name	If the Return Type is Unit or Units, then this selects and returns the specified property out of the unit, rather than returning the entire unit object. If the Return Type is neither Unit nor Units, this is not applicable.
Time Range To Use	You can choose the default time range, or all time, or any other time range that has been defined.
Parameter block	Allows you to choose a binding for On Null, or any other parameter that has been declared in the query.

Writable Data Object

If there is a need to build a custom complex user interface in Foglight there may be a need to create temporary objects to manage the new user interface, especially when there is a need to capture input from users to convert later into more persistent objects or to remember choices that a user has made in the past in order to streamline the user's workflow in the future.

The Writable Data Object mechanism allows for the creation of types in the Web Component Framework that are specific to the UI mechanism and associated with the modules that contain the rest of the custom UI artifacts.

NOTE: Writable Data Object instances can be accessed from multiple threads, for example, when using data objects in functions that run in the background (in a different thread). As such, these objects are not thread-safe.

UI Type entities allow you to define types and their properties that can be used in various ways in your UI.

Creating a Data Type in a Module

You can define your own types if necessary.

To create a new user interface type:

1. Choose **Types** in the Definitions > Module Contents pane.
2. Click **Add** and the **Add Type** dialog box appears.
3. In the **Name** field, type in the name for your new type. The name must be capitalized.
4. Choose the super type for your new type by selecting an entry in the list of available super types, and then click **OK**.

You can choose the generic type **Data Object** from the *common* data source if your new type is not based on a particular data source type. Note that you can use the search box to filter the list of available types.

5 In the edit pane, fill in the **Display Name** and **Description** fields.

6 Define the properties of your new type by filling in the *Properties* table.

- **Name**: A property name must start with a lower case letter and contain only A-Z, a-z, 0-9.
- **Display Name**: Any string value is allowed
- **Data Type**: The property's data type.
- **List**: Specifies whether the property is a list.
- **Reference**: Specifies whether the property is a reference. For instance, properties that are enumerated types are always references because the values in enumerated types never change and there is no need to copy the enumeration into the data object. Static types such as strings and numbers cannot be references, but many object types can be. The Definitions editor will be enabled for the property if it can be either contained or a reference. See Saving Data Objects on page 113 for information on how contained and reference types are saved.
- **Default Value**: If desired, a fallback value for the property here. A newly-created object has all its properties set to their fallback values.
- **Annotations**: For information about annotations, see Properties on page 111.

Creating a Data Object based on the New Type

Instances of the new type are created with the Context Binding mechanism available when specifying additional context.

To set the parameters in the Writable Object Binding dialog box:

1 In the Context tab of the definitions editor, click the **Add Context Entry** ( ) button in the *Additional* group.

2 In the **Key** field, give the data object a name by which it can be referenced.

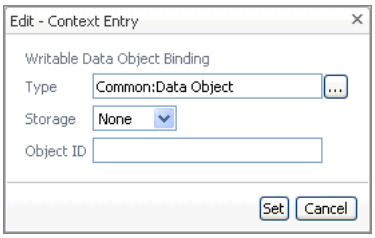
3 If desired, give the data object a display **Name**.

4 Select the **Evaluate Once** check box to create a single data object that may contain other variable properties such as lists.

5 Click the Edit ( ) button and choose **Writable Data Object**.

6 The **Edit - Context Entry** dialog box appears.

Figure 36. Edit - Context Entry



a Choose your new type from the **Select Type** dialog box by clicking the ... button at the right-hand side of the **Type** field.

b Choose a **Storage** persistence. The create method attempts to load the data object from storage or it creates a new instance if the object does not exist. The choices are:

- **None**: The object cannot be stored and is meant to be transitory (used on the page or passed through the context).

- **Site**: Enables the object to be stored in a global location that is accessible to all users. If multiple users are changing the object at once from different sessions, the last copy of the object saved will be persisted.
- **User**: The object is stored so that each user has a different copy of this object. This is useful to remember a user's individual choices.
- **Session**: The object is stored in the user's HTTP session. Calling `save()` is not necessary to persist changes because the single live instance is stored in a session-owned cache. The object is released when the user logs out or when the object's `remove()` method is called.

c Type an identifier for the new object in **Object ID**, which must be formatted as *xxxxx-yyyyy*, where:

- *xxxxx* is your domain identifier (Examples: *java, oracle, ops, admin*)
- *yyyyy* is an identifier for the data object. (Example: *socFilter*)

IMPORTANT: Data Objects created using this approach are considered equal if their Object IDs are equal.

7 Click **Save**.

IMPORTANT: Objects created this way are not time range sensitive.

Saving a Data Object

Changes to an object are persisted by calling the object's `save()` method.

To be saved, an object must be created with a storage mechanism *Site*, *User*, or *Session*.

Saving Contained versus Referenced Properties:

- A contained property is saved along with the parent.
- A reference property (if it is a reloadable object) is saved as a reference that is reloaded and set when the parent is recreated.
- A non-reloadable reference property is reloaded with its fallback value or null when the parent is re-created.
- Simple types (Strings, Numbers, and so on) cannot be references. Enum types are always references.

Removing a Data Object from Storage

An object using *Site*, *User*, or *Session* storage can be removed from those storage mechanisms by calling the `remove()` method on the object.

Query

A Query Binding evaluates to a specified value in the data object or list of data objects returned by a query. For example, you can display a list of hosts from a selected data object.

A Query has the following properties:

Table 19. Query properties

Property	Description
Query	The query that is run to determine the set of values to be used. Open the drop-down menu and select from a tree of queries.
Path	A path within the results from the query. This displays the actual names of the data-object properties, and not the localized names. For more information, see the Data Type Reference.

Table 19. Query properties

Property	Description
Iterate Over 1st Parameter	For details, see Configuring the Query Dialog Box on page 86.
Time Range To Use	If this is checked, the TimeRange currently in force is ignored, and data for *all time* is selected.
Return First Object in List	This lets you select the first item out of the results of the Query property, which is always a list. The optional Path is then applied to that specified element.
Renderer	See Renderers on page 106.
Parameter block	Allows you to choose a binding for On Null, or any other parameter that has been declared in the query.

Unlike String parameters, Query parameters are not used directly by the Query. Rather, they are evaluated by the Query, and then passed on to the underlying query, which is executed by the Query. For more information, see Parameters in Queries on page 69.

Parameters in a Query are often used to specify the root path of a query. If the root is not a parameter, it is an absolute path from the root of the data source. If a query's root is specified with a parameter reference, then the corresponding parameter from the Query is assumed to have evaluated to a data object or a list of data objects. That value is used as the root object for the query.

The behavior becomes more complex if the root is a parameter that evaluates to a list of data objects, and the query specifies aggregation, such as *Max*. In this case you have a choice:

a Obtain just one aggregate value for the result of the query against all of the elements in the list of data objects that is the value of the root object's parameter.

b Obtain one aggregate value for the result of the query against each element in the list that is the value of the root object's parameter. The result is a list of aggregated values.

NOTE: For more information about aggregation, see Aggregations on page 60.

For example, you have a view which is a row-oriented table that displays a list of hosts down the rows. You want to see the maximum severity of events for each host in another column of the table, which requires using option *b*. If the root parameter is the list of hosts, and the aggregate Query is selecting the maximum severity of events, then the results are a list of the maximum severities of the events for each of the hosts (one maximum per host).

Configuring the Query Dialog Box

To configure the query:

1 Select the **Values** property of the desired column of the row-oriented table view.

2 Set that property to be a **Query** that accesses the correct query.

3 Check the **Iterate Over 1st Parameter** check box in the Query.

CAUTION: Failure to follow these steps results in the query showing just one value for all the elements in the list (option a).

If Iterate Over 1st Parameter is selected, then the query executes once for each row. This mechanism is available only for the first parameter in a query with multiple parameters. It executes for each element from the list that is the value of the first parameter. The results of each of those query executions are amalgamated into a list, which becomes the final value of the Query.

If a Query uses Iterate Over 1st Parameter and has an On Null binding, the On Null may need to access the current value being iterated over during the Query evaluation of the list that is set as the first parameter. To make this possible, the current value from the first parameter is put into the context with the key *currentParameter* prior to evaluating the On Null.

CAUTION: Never create a user-defined context key with the name currentParameter.

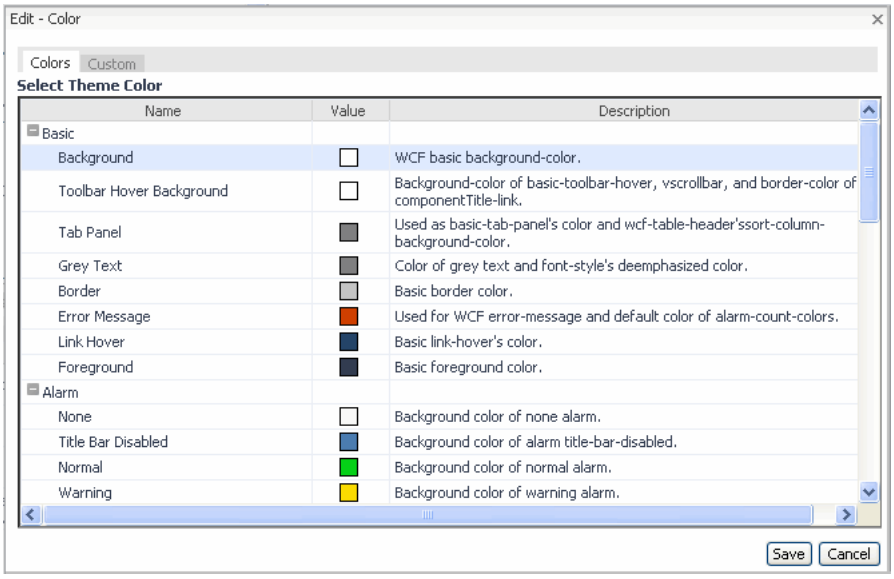
Queries always return a list of results even if there is just one element in the list, as is the case with aggregation queries. Therefore, if a Query uses Iterate Over 1st Parameter and has a list of hosts as the parameter being iterated over, then the result returned is a list containing one result element for the evaluation of the query for each host. In most cases, each result element is a one-element list containing the result data object.

To easily access this sole result element, check Return First Object in List. This selects the first element out of each list of query results. When Return First Object in List is used in combination with Iterate Over 1st Parameter, it is applied to each of the query results. The final result is a list that includes one element per host where each list element is the data object containing the result data object.

Theme

Themes can be defined so that views can have a different appearance when printed. Settings for different colors or theme palettes are available for those properties that support these choices.

Figure 37. Themes



Icon

Table 20. Theme properties

Property	Description
Color	A chooser allows you to accept colors from a palette.
Custom	Allows you to input component, style and value strings.

Icon selection is how you specify an Icon binding. The Icon selection allows a user to select a specific icon (as defined in the Icons tab of one of the modules), and attach an Icon Renderer. The Icon Renderer controls the size of icon that is drawn.

An Icon has the following properties:

Table 21. Icon properties

Property	Description
Icon	A drop-down list allows you to choose an icon from the existing group of System icons, or ones that have been added to user modules. For more information, see Type Mappings on page 108.
Renderer	A drop-down list allows you to choose a renderer from the existing group of System renderers, or ones that have been added to user modules. See Renderers on page 106.

String Template

A String Template binding evaluates to a string or list of strings. Unlike a simple string, this binding accepts parameters that can partly determine its value.

A String Template has the following properties:

Table 22. String Template properties

Property	Description
Value	A text box allows you to type a value for the string. Parameters are permitted.
Renderer	A drop-down list allows you to choose a renderer the existing group of System renderers, or ones that have been added to user modules. See Renderers on page 106.
Parameter block	Allows you to choose a binding for On Null, or any other parameter that has been declared in the Value box.

Parameters are added to the String Template through standard 0-based arguments common in programming. The first parameter is substituted for the text {0}, the second for {1}, continuing for each sequential number. For example, the *String Name Machine{0}, Application{1}* with parameters that evaluate to *One* and *Two*, returns the value Name, *MachineOne, ApplicationTwo*.

If the parameter evaluates to a list, then the String also evaluates to a list, repeating the string for each value in the list.

For example, suppose the String Name: {0} has a parameter that evaluates to a list of server names:

- MachineOne
- MachineTwo
- MachineThree

Then the String binding evaluates to the following list:

- Name: MachineOne
- Name: MachineTwo
- Name: MachineThree

A renderer can also be specified for a String binding.

Rich Text and Rich Text Template

The Rich Text data value is the same String Template, but permits some XHTML in its content. There are no block-level constructors. For more information see String Template. A renderer cannot be chosen. Any formatting must be controlled by the use of XHTML tags.

The Rich Text Template allows parameters to be set as well. A Rich Text Template has the following properties:

Table 23. Rich Text Template properties

Property	Description
Value	A text box allows you to type a value for the string. Parameters are permitted as well as XHTML.
Parameter block	Allows you to choose a binding for On Null, or any other parameter that has been declared in the Value box.

CAUTION: Any unexpected interactions between custom HTML used in Rich Text and the Web Component Framework are the responsibility of the Rich Text creator. They must be tested in all supported browsers and may need changing from release to release.

Image Reference

A reference to an image that is specified by its URL.

Function

You can see the listing for pre-defined functions in the Definitions Editor by selecting Functions and then choosing any module in the Module List Pane. You can define your own functions as well. The list of function names appears in the **Edit - Context Entry** dialog box when you configure the Function binding.

Localized String

NOTE: This binding is available only for components in system modules.

A Localized String binding evaluates to a string whose representation may change depending on the locale. This can be a simple string, or it can accept parameters that evaluate to a string. The parameters are similar to a String binding. When entering a Localized String, there are two input values:

- Localized String (required)
- Renderer (optional).

The desired Localized String is edited differently depending on whether you are editing a view within a System Module or a User Module. In a System Module, the localized string is typed in, whereas in a User Module it is selected from a drop-down list.

The localizations are stored in a properties file associated with the *wcf.xml* file that defines the module. However, depending on the application, these files may be stored in the file system, or in a database. The English text files are named `strings.properties`.

Other languages have their own set of files. For example, the corresponding French files are named `strings_fr.properties`. If the files required for your language do not currently exist, contact your system administrator about how to copy and edit the existing strings.properties files into the desired filename for your language, and install those new files.

The values in the properties files are in the form *key=value*. The key is an internal code used by the system. If you use a key that contains spaces, precede them with a backslash. For example, the key *cpu usage* is entered as *cpu\ usage*. The localized value of the selected string is displayed in the binding drop-down list tree.

Data

Generally, Query bindings are used to retrieve data from a data source. However, you can use a Data binding in limited cases to directly select one data object or a list of data objects.

A Data binding has the following properties:

Table 24. Data binding properties

Property	Description
Data Source Type	The data source type. It is only for your information and cannot be edited.
Data Source ID	A drop-down menu of the available IDs. You must select a specific data source instance to force the data object to be selected from that instance.
Data Object	A tree of all of the data available for the given Data Source ID is displayed. You select the desired data object by expanding the tree as required, and then clicking on a data element. This could represent a list of data objects, a single data object, or a simple property of a data object, such as a string property. This is somewhat similar to the Path parameter on a Query, or Context, or the Root Path property of a Query.
Renderer	A drop-down list allows you to choose a renderer from the existing group of System renderers, or ones that have been added to user modules. See Renderers on page 106.

A Data binding cannot have parameters, but it can have On Null.

There are some significant differences between the usage of a Query binding and a Data binding. You do not have to specify the Data Source ID for queries to enable them to select from the default Data Source. By contrast, Data bindings do have to have their Data Source ID specified. As a result, they are generally only used for testing or in User Modules, never in System Modules because the Data Source ID in System Modules is not known in advance. Queries are much more flexible and powerful than Data bindings.

Data Object Property

Use the Data Object Property to set a context key for a particular property of an available data type. The fields available for edit in the dialog box are:

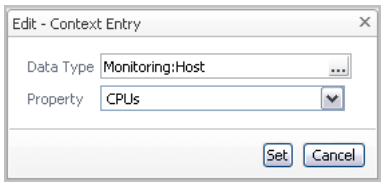
Table 25. Data Object properties

Property	Description
Data Source Type	Any of the available data sources are displayed in a drop-down list.
Data Type	The available data types are displayed in a drop-down list.
Property	Properties of the data type are displayed in a drop-down list.

Data Object Property (and Property)

A data object property is chosen in the **Data Object Property** dialog box.

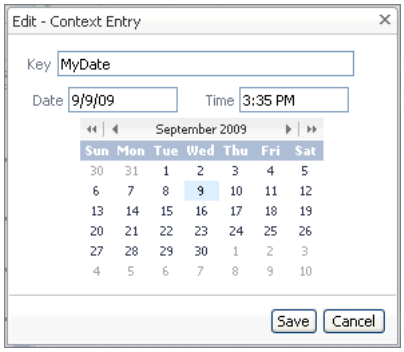
Figure 38. Data Object Property



Date

A Date object is chosen in the **Edit - Context Entry** dialog box.

Figure 39. Edit - Context Entry



List

The List binding must always have parameters. It allows you to either make a list containing the values of each of the parameters, or to make a list that merges all elements in the values of the parameters into one list.

A List binding has the following properties:

Table 26. List binding properties

Property	Description
Merge Lists	If unchecked, the resulting list contains one element for each parameter, with the element containing that parameter's value. If checked, all elements in the list values of the parameters are merged into one big list. In addition, if the parameter values are lists of lists, only the bottommost non-list elements are put into the final list, so that no element of the final list is itself a list. Any data object that is seen as a duplicate of an existing data object is eliminated. However, if the objects in the list are not data objects, duplicates are not eliminated.
Remove Duplicates	Only applies when Merge Lists is true. If true, then duplicates are removed from the merged list, otherwise they are not. A special option, *Intelligent* is available for List bindings created in older versions of the *Web Component Framework*, in which nulls were unpredictably removed or not.
Remove Nulls	If true, nulls are removed, otherwise they are not. A special option, *Intelligent* is available for List bindings created in older versions of the *Web Component Framework*, in which nulls were unpredictably removed or not.
Renderer	A drop-down list allows you to choose a renderer from the existing group of System renderers, or ones that have been added to user modules. See Renderers on page 106.
Parameter block	Allows you to choose a binding for On Null, or any other parameter that you add using *Add Parameter*.

Return Types

There are ten possible types of data that you can extract from the context and the context paths:

- Localized Value
- Value
- CountPerItem

- Count All
- Unit
- Units
- Localized Property Name
- Localized Property Names
- Localized Type Name
- Localized Type Names

Localized Value

This is the default return type. It returns the basic type of data of the context, but localizes it if a localization is available. If the context is a host server and the path is name, then it returns a simple string. If the context is a list of servers and the path is name, then it returns a list of names.

Value

Value returns the basic type of the data in the context, not its localized value. If the context is a host server and the path is name, then it returns a simple string. If the context is a list of servers and the path is name, it returns a list of names.

CountPerItem

This is used when you want to show the count of a list. For example, use this property to display the number of file systems on a host. Count acts on the result of evaluating the Context binding normally. There are several cases:

- If the result is not a list, then the number of objects is “1,” and therefore the count is “1.” This is not a common use of this property. If you set up the context to be a single value, then you do not need to derive the count.
- If the result is a list of objects, then this returns the number of items in the list. For example, a Host data object has the events property, which is a list. If this property is used as the context value, then Count returns the number of events.
- If the result is a list of lists, then it returns a list composed of the results of evaluating the *CountPerItem* on each value in the top level. For example, if the context is set to hosts, and the path is set to events, then *CountPerItem* returns a list of the number of events for each host.
- If the result is a list of lists of lists (or even deeper), the rule is the same as in the previous case, but it can return lists of lists (and so on...) of counts.

Count All

Unlike *CountPerItem*, this return type just counts up all of the non-list elements of a list. For example, if context is set to Hosts and the path is set to Events, then Count All returns the total number of events for all of the Host data objects. *CountPerItem* would return a list, with each item a count of events under a given host.

Unit

This returns the unit associated with the path. If the path is set to *cpuUsage*, then this returns the unit object for percentage (%). If the path is set to *diskUsage*, then this returns the unit object for size (in MB). Unit returns null if no unit is associated with the path.

Units

Unit only returns one unit, even if the path is set to a list property. If you have a view that contains a list of different metrics for one object, then you can use Units. This returns a list containing the unit for each item. For example, the view shows the CPU usage and the disk usage for a host in a row-oriented table. Set the Return Type to Units for a column, and the entries shows % and MB for CPU usage and disk usage, respectively.

Localized Property Name

Each property of a data object has a localized name. This Return Type returns the localized name of the property set in the Path. For example, if the context is a Host, and the path is set to *cpuUsage*, then Localized Property Name returns CPU Usage.

Localized Property Names

Similar to Units, this returns a list containing the localized name for each element of a list. This differs from Localized Property Name, which returns one value for the first element in a list.

Localized Type Name

This returns the localized name of the type of the property. For example, if the item taken from the context is a Host, and the path is set to *cpuUsage*, then Localized Property Name returns Metric. If the path is set to name, then Localized Type Name returns String.

Localized Type Names

Similar to Localized Property Names, this returns a list containing the localized name of the type of the property for each element of a list.

7

Additional Components

A number of helper components round out the Web Component Framework. Renderers are available to provide special formatting. The Task mechanism allows you to launch actions from your view. You can file arbitrary Icons and use them in your views. Units are available for use with metrics. Theme and Module Resources let you specify different appearances, such as the Application theme or the Monitoring theme. The Printing mechanism allows you to print reports.

Files

Developers can upload files such as image files to each module's public directory, by using the Files tab in the Definition Editor. Files can be uploaded to the root directory (under public), the images directory, or other subdirectories.

To manage files within a module:

1 Select the module, and then click the **Files** tab.

The list of files (with their paths) within that module is displayed. Clicking on a filename displays the filename along with the special Web Component Framework URL that refers to that file, for instance in the Background Image field of a View, the URL field of an Image Renderer.

You can copy that URL to the clipboard for later use.

2 Select the URL, then click **Edit > Copy** into the command line of your web browser.

To delete a file:

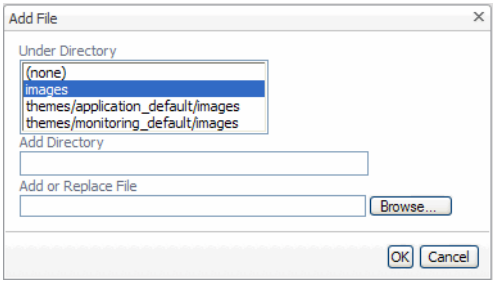
1 Select the file, then click the **Delete** toolbar icon.

2 Select **OK** to confirm the deletion of the file.

To add or update a file:

1 Click the **Add** toolbar icon. The following dialog box appears.

Figure 40. Add File dialog box



You can select which existing directory (under public) the file is uploaded to by selecting from the Under Directory list box. You also move it into another subdirectory of the selected directory.

2 Type the new subdirectory name in the **Add Directory** field.

3 Type in the full path name on your machine to the file to be uploaded in the **Add or Replace File** field.

   Alternatively, click the **Browse** button to select the file using a standard file open dialog box.

4 Click **OK**.

   A confirmation message appears confirming the upload that is going to happen. When you click OK on that message, the upload occurs. Unless an error message is shown, the new file displays in the list of files, along with the Web Component Framework URL for accessing it.

Icons

The Icons choice allows you to configure how graphics render in the Web Component Framework. In a view definition, an icon can be referenced directly by using an Icon binding and then setting an Icon Renderer to specify the desired size. In the Types tab, an icon can be mapped to a data type. In a view definition, selecting data of that type and specifying an Icon Renderer causes the data to display as the mapped icon.

Icons collect differently-sized (but similar in appearance) images under a single name. Images can be specified for the following sizes:

- Extra small (8x8 pixels, or 9x9, depending on the image)
- Small (11x11 pixels)
- Medium (16x16 pixels)
- Large (32x32 pixels)
- Extra large (64x64 pixels)
- Huge (128x128 pixels)
- Scalable (any size)

NOTE: Each size should have its own image that has been created with the appropriate number of pixels. The icons are then uploaded from the Files section.

Associations

Associations provide a mechanism to organize references into entities. A typical association contains a label (a renderable artifact), one or more tags for organization, and a reference to an entity, frequently a view. Different types of associations include properties that are specific to their purpose.

The following types of associations are available:

- Tab Associations
- Question Associations
- Category Associations
- Domain Associations
- Message Associations
- Customizable Associations
- Hierarchy Associations
- Alarm Associations
- Activity Associations
- Action Associations
- Node Associations

- Edge Associations

Tab Associations

Tabs supply views to the Tab Manager component. The reference contained by a tab association is a view, displayed by the Tab Manager component when the tab is selected.

Tab Associations are configured and managed like view components, except that they have no flow (and therefore no generated context). The purposes of a tab association include:

- Providing a label (title) for the referenced view that can be displayed without the view being loaded
- Permitting more control over the selection of dynamically-loaded tabs
- Allowing context re-mapping between the Tab Manager and the referenced view (which is normally not possible for views which are dynamically added to a container)

A tab association contains the following elements:

- Label and description, used to display the tab
- View, displayed when the tab is selected
- A set of tags, used by the Tab Manager component to find the applicable tabs to display
- Priority property, used to order the tabs
- Display property, used by the Tab Manager to determine if the tab is visible or not.

Question Associations

Questions are used by the question viewer and mini-viewer components. The label-type property is `Question` which contains the text of the question to display, and the Answer View contains the answer to that question.

Questions belong to a domain (see Domain Associations), that broadly filters the set of all questions to those appropriate for a particular viewer instance with a corresponding `Domain` property set. Each question has one or more categories, that control grouping and are used for navigation. In addition, a question can have an associated Report View, used if the question is added to a report.

Category Associations

Category Associations do not directly reference another entity, and are used primarily to organize questions. Categories have a `Display Name` property, seen by the user, as well as a `Canonical Name` property. The canonical name is used to match and merge category instances. In some cases, multiple developers may need to independently define certain semi-common categories, but the questions should be sorted together on a running system.

Domain Associations

Domains are like high-level categories. A single domain definition corresponds to a specific domain (for example, DB2), and it logically associates all questions and question viewers in that domain. It is also the basis by which activities are discovered by the Domain Viewer component.

In addition to the properties included with categories, a domain has an icon associated with it, that visually identifies the domain, and an Overview view which provides a default view of the domain. The definition also includes the `Parents` property which defines how the domain relates to other domains.

Message Associations

Messages inform the users that are currently logged into the browser interface of any activities that may affect them. This way you can quickly inform the end user when a background process that they started is finished. For example, when a process that prints a report in the background finishes, you can send a message to notify the user.

A message includes the message text and message details. It also contains a link to a relevant view, if its `View` property is set. For example, if you need to show a message that references a view displaying more information, if the `View` property of the Message Association is set, the view name appears as a link in the message. Clicking the link displays the specified view.

Customizable Associations

Customizable Associations form a special class of free-form associations that can express relationships between elements of their solution domain. Customizable Associations that are interchangeable have a common `Group Tag` property, that essentially identifies the purpose that a particular association fulfills. A Customizable Association has an icon that can be used to visually represent the association or its target, and a free-form `Value` property that is a list of the type `Any`. The association can refer to a single view, for instance, or a list of domain associations, or whatever entities or data you need to be associate with this instance.

Hierarchy Associations

Hierarchy domains group and categorize objects by their type and function. A common example of hierarchy domains is a combination of various types of monitored operating systems, such as OS, Windows®, Linux®, UNIX®, AIX®, HP-UX, Oracle Solaris®.

Hierarchy domains are used by the Domain Viewer to sort and group objects for counting. Higher-level hierarchy domains can contain sub-domains.

Alarm Associations

Alarm Associations link alarms (using its rule ID) to default dwell and drill-down views, and optionally to a set of named or tagged diagnostic and remediation workflows, when such workflows are defined.

Views that display alarms can use alarm associations. An Alarm Association for a given rule ID contains the following elements:

- Detailed views
- Drill-down view
- List of tags or names that refer to diagnostic views or workflows
- List of elements that refer to remediation workflows

Activity Associations

A domain activity associates an activity with a hierarchy domain, and defines how objects in the domain are counted.

Domain activities are primarily used by hierarchy domains within the context of the Domain Viewer.

Each domain activity specifies an activity and a hierarchy domain, meaning that the given activity is available to that domain. It also specifies a function that is used to categorize the state of objects belonging to the domain, which generates counts in the Domain Viewer.

Action Associations

An Action Association defines an action for use in toolbars and action lists. A minimal action has label text and a reaction that is fired when the action is triggered.

Action Associations are used by toolbars and components that provide sets of actions internally.

Actions can be tagged and discovered by matching tags which means that a toolbar can dynamically have actions added to it, as functionality is added to the system.

The `Disabled` and `Display` properties may be bound to RVs which can control the visibility and accessibility of the action based on current data/state.

When an icon is configured for an action, it appears in toolbars and in action lists.

Node Associations

Node Associations bind a tag and a type to configuration for an object node in a topology component.

Node Associations have a tag (or set of tags) which are used to match the nodes against topologies which make use of them, and a `Type` and `isList` properties, that match an object type displayed in a topology.

Given a matching Node Association, the topology can retrieve the properties necessary to render the node.

Edge Associations

Edge Associations bind a tag and a pair of `Type` and `isList` properties that define the kind of nodes that the edge links.

An Edge Association contains:

- One or more tags that are used to match the edges against the topology that uses them.
- A pair of `Type` and `isList` properties defining the origin of the edge.
- A pair of `Type` and `isList` properties for the termination of the edge. The combination of these properties matches a specific kind of connection within the appropriately tagged topology.

Given a matching Edge association, the topology can retrieve the properties necessary to render the edge.

Creating Associations

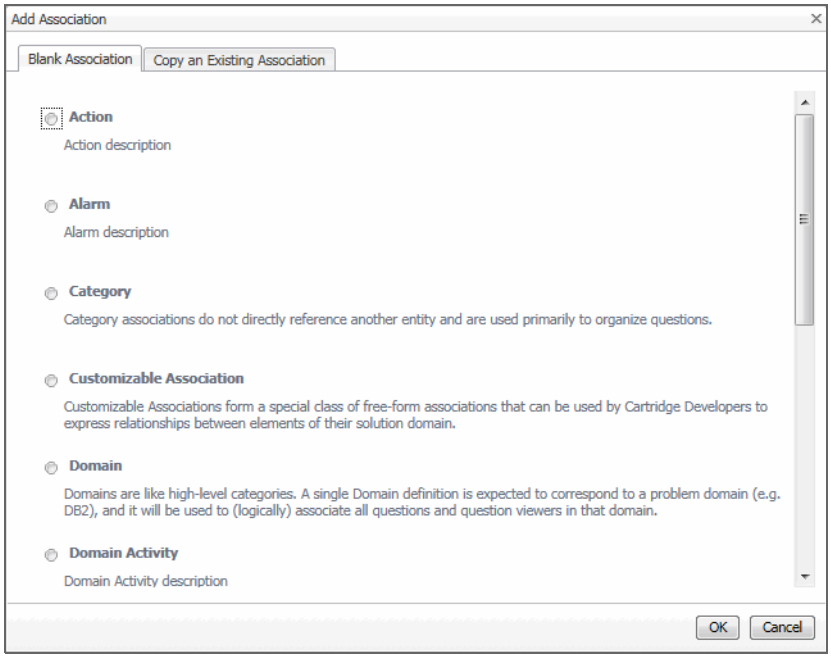
An association is used to house and organize references to entities. Create a new association either by starting with a blank association, or by copying an existing association.

To create an association:

1. From the navigation panel, under **Dashboards**, click **Configuration > Definitions**.
2. In the Module List pane, select the module where you want to create the association.
3. In the Module Contents pane, select **Associations** from the drop-down list.
4. Click .

   The **Add Association** dialog box appears.

Figure 41. Add Association dialog box



5 **Starting with a blank association only.** Create a new, blank association by selecting the association type.

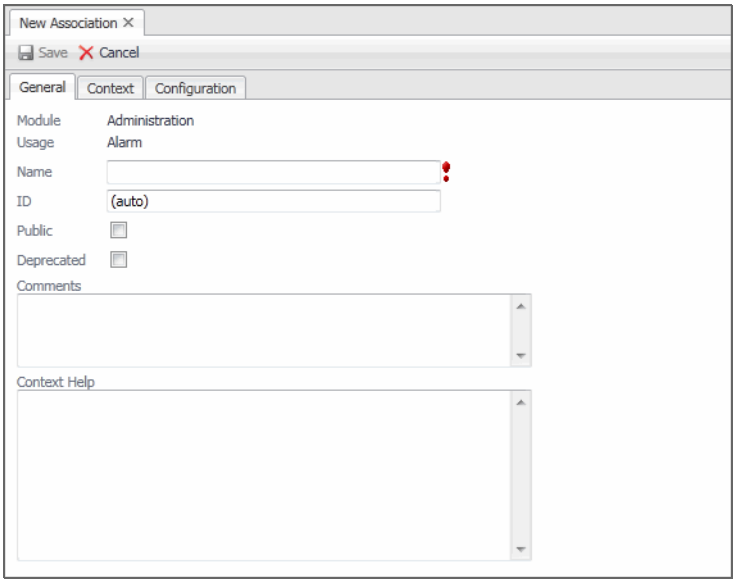
a In the **Add Association** dialog box, ensure that the **Blank Association** tab is open.

b Select the desired association type.

c Click **OK**.

The new association appears in the editor pane.

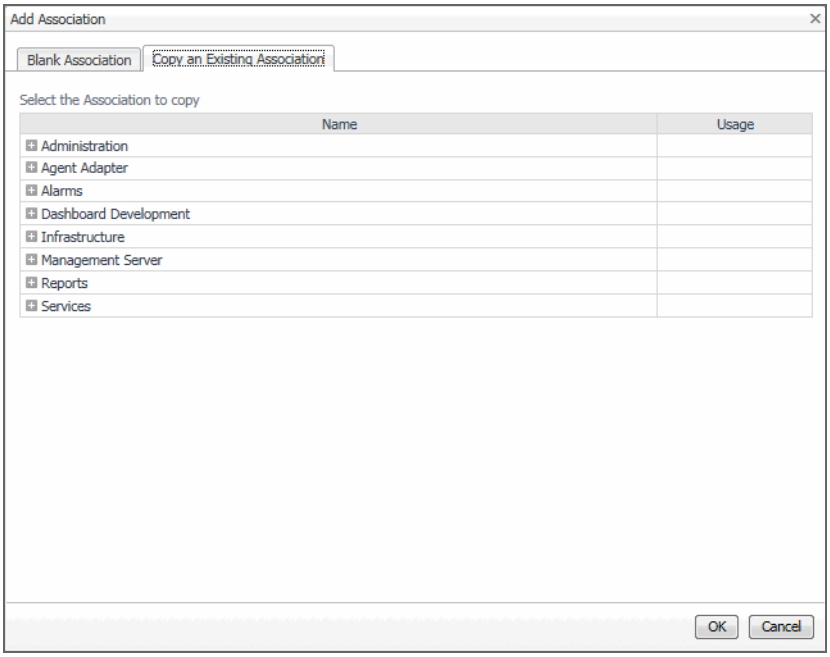
Figure 42. New Association tab



6 **Copying an existing association only.** Create a new association by copying an existing association.

a In the **Add Association** dialog box, open the **Copy an Existing Association** tab.

Figure 43. Copy an Existing Association tab



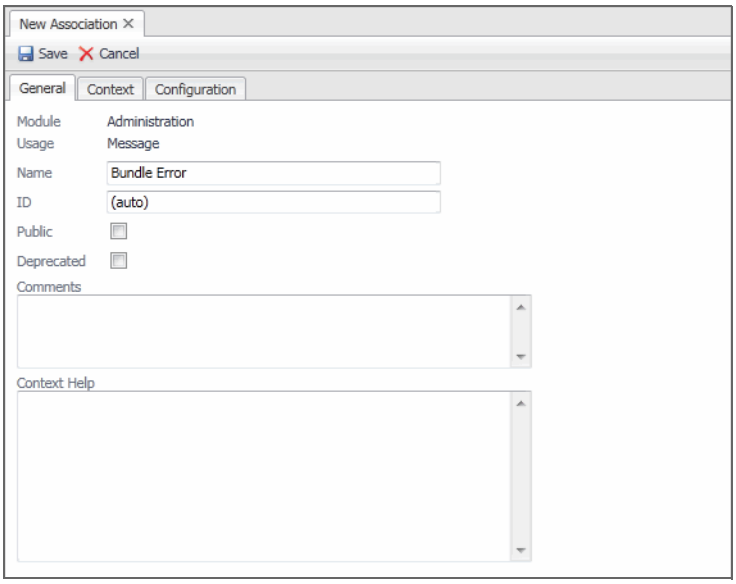
This tab displays a navigation tree containing a list of the modules that exist in your system. Expanding the tree structure shows the available associations and their usage.

b Navigate to the desired association and select it in the tree.

c Click **OK**.

The new association appears in the editor pane.

Figure 44. New Association tab



7 Configure the association by filling in the required fields in the editor pane. The fields for the different association types are described in the *Web Component Reference* help pages.

Creating Alarm Associations: Example

Alarm associations are used to link alarms to view components. Views that display alarms can use alarm associations. If you want alarms to use the **Alarm** dialog box, create an alarm association for each alarm type you want to display in this dialog box. If you want the **Alarms** table and **Alarm** dialog box to display agent information, the agent should be set as the monitoring agent for each entity it monitors.

Configuring the alarm association requires setting the following elements:

- To define agent details to display, specify the time range and agent as context inputs.
- Configure the `Small Detail View` property to point to a view to display on the **Recent History** view on the **Summary** tab.

  IMPORTANT: This is a small view. It should be approximately 130 px high.

- Configure the `Large Detail View` property to point to a view to display on the **Recent History** view on the **History/Notes** tab.

  IMPORTANT: This view should be approximately 130 px high, but can be wider than the Small Detail View.

- Set the `Rule ID` property to the ID of the rule that generates the alarm.

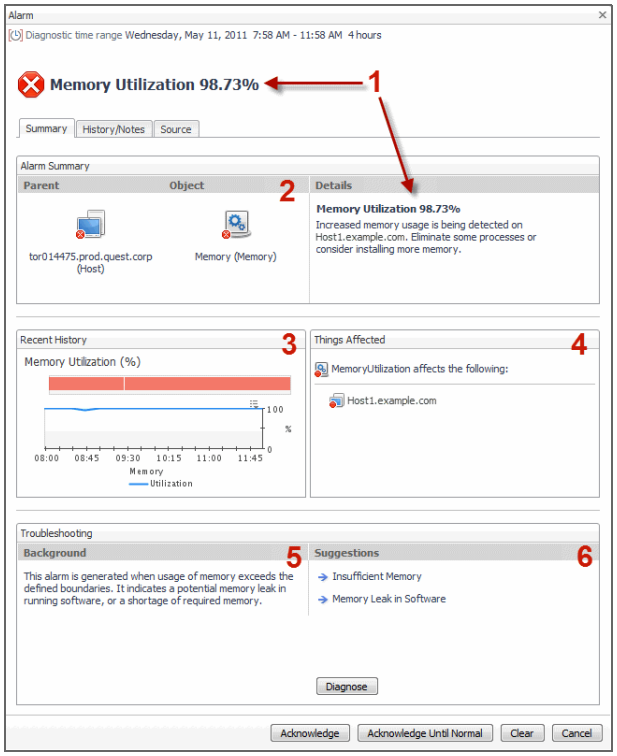
  IMPORTANT: There should be only one alarm association per rule. Otherwise, Foglight uses the first association it finds.

- The rule that generates the alarm must have its **Alarm Description** property set.

Configuration details

The following diagrams indicate the components of the **Alarm** dialog box that need to be configured in the alarm association. Use the guidelines provided below when configuring these components in an alarm association.

Figure 45. Alarm dialog box



1 This text comes from the first line of the alarm description. The description is parsed up until the first space following a period. The second line is the text that comes after the first space following a period.

> **IMPORTANT:** Keep this in mind when writing alarm descriptions. An alarm description should follow this format and have at least two sentences.

2 The object in this image is the `topologyObject` property of the alarm. If the object does not have a parent (if it is the Domain Root, which is the parent that appears), then only that object appears in the centre of the space.

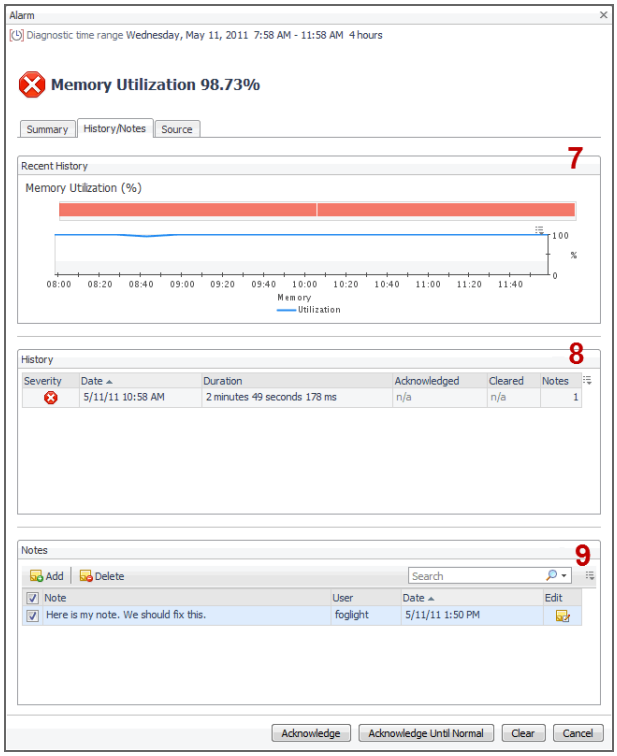
3 The **Recent History** view is the view to which the `Small Detail View` property is mapped. There is no default if this is not specified. The space allows for a small detail view whose height should be no more than 130 px.

4 The elements affected here are determined by the parents and services to which the `topologyObject` belongs.

5 The alarm association description. There is no default if this is not specified.

6 These diagnostics are customizable associations. You must provide a name and description for each. It is best to provide a few diagnostics, but too many can result in an overhead and as such be counter-productive. The **Diagnose** button takes its view from the `Diagnosis` property. If none is provided, the button does not appear. This property is optional.

Figure 46. Recent History view, History table, and Notes table

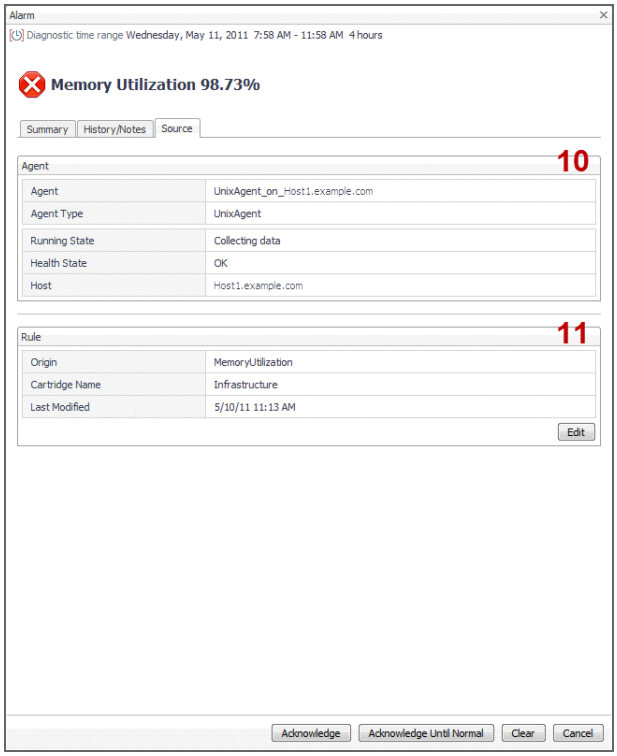


7 The **Recent History** view on the **History/Notes** tab is the view to which the `Large Detail View` property is mapped. It has similar height restrictions to the `Small Detail View` (130 px, see Step 3), but can be wider. This can be the same view as the `Small Detail View`, or a different view containing alarm metrics. The contents of the selected view should be compatible with the title (Recent History). In cases where the small view is flexible then the view to which the small view is mapped can be mapped to the large detail view as well. The large view should also be compatible with the title (Recent History).

8 The **History** table is the standard list containing alarm instances, and appears by default.

9 The **Notes** table is the standard list containing user-defined notes, and appears by default.

Figure 47. History and Notes tables



10 On the **Source** tab, the **Agent** and **Agent Type** entries in the **Agent** view contain the agent name and its type, and appear by default. The section containing the **Running State**, **Health State**, and **Host** entries also appears by default, and can be replaced by setting the `Agent Detail View` property. To populate this view, ensure that the agent is set as the monitoring agent for each entity it monitors.

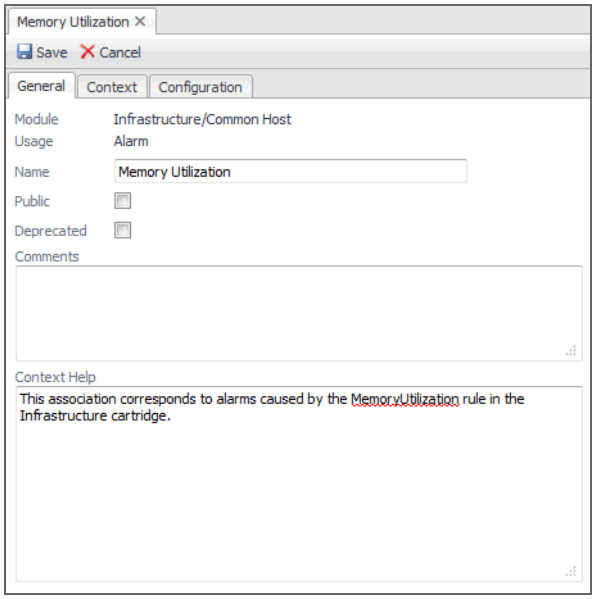
> **TIP:** The **Alarm** dialog box looks for the agent under *<alarm>/topologyObject/monitoringAgent*.

11 The **Rule** view appears by default.

Configuration example

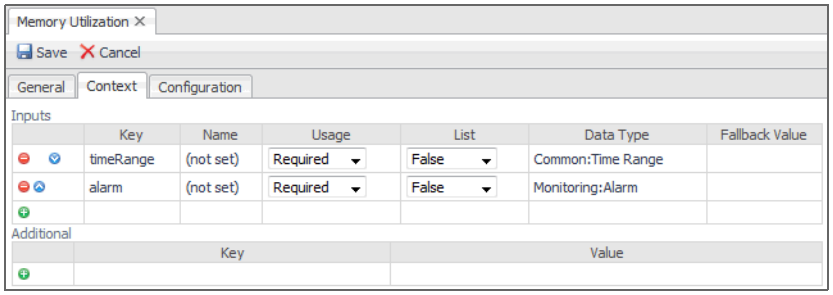
The following images show a sample configuration of an alarm association.

Figure 48. Alarm association example, General tab



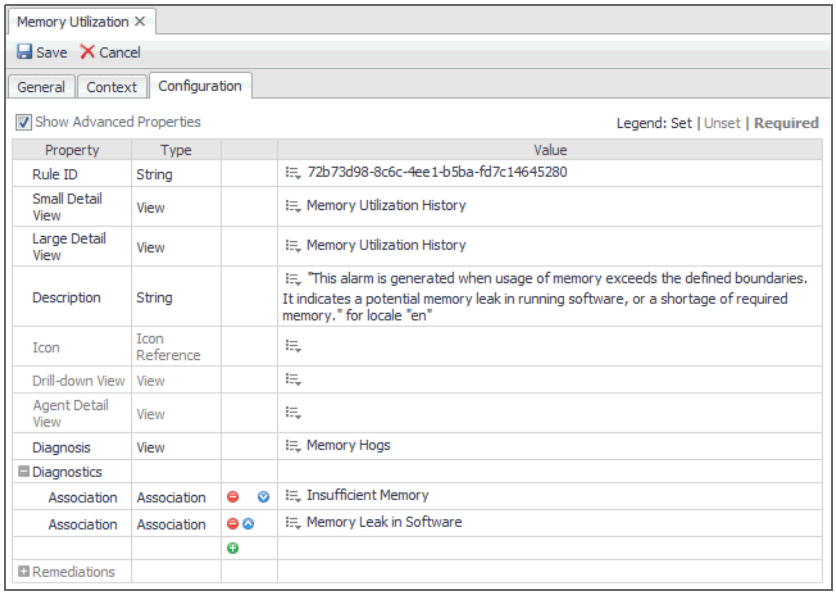
TIP: Context help is optional, but often very useful.

Figure 49. Alarm association example, Context tab



TIP: Passing the alarm is useful in most cases, especially when you link the alarm to specified views.

Figure 50. Alarm association example, Configuration tab



TIP: The Rule ID is mandatory, otherwise the browser interface fails to match up the association to its alarm.

NOTE: The `Description` property is useful for verbose background information.

The `Small Detail View` maps to the **Recent History** view appearing on the **Summary** tab.

The `Large Detail View` maps to the **Recent History** view appearing on the **History/Notes** tab.

This example contains a mix of required and optional settings. Not all of the optional settings are set. For example, the `Agent Detail View` property is not set, which results in the **Agent** view showing the default entries.

Renderers

Configuring how data is displayed in a dashboard is determined in the view definition, but relies on the existence of renderers, icons, and mappings that are configured separately from the view.

Renderers are created and configured in the Renderers tab in the Module Contents pane. Renderers determine the display of data. For example, the number of decimal places shown by a value inside a table cell is decided by a Number Renderer instance, the display of data as an icon is determined by an Icon Renderer instance, and the display of a metric as a time plot chart is configured using a Sparkline Renderer instance.

At runtime, how data in a Foglight® dashboard displays depends on which renderers are set in the view definition and which renderer has been mapped to a data value's type, property, or unit. Renderers can be mapped to types and properties in the Types tab or to units in the Units tab. Mapped renderers are looked up automatically at runtime if no specific renderer has been set in the view definition.

Default Renderer can be Overridden

All the units and also some types supported by Foglight Core are associated with a default renderer. These associations can be found in the Module Definitions pane modules under the Units and Types tabs. For example, a percent unit is associated with the Percent Renderer.

To define a new renderer:

1. In the Module List pane, select the module that should contain the renderer.
2. Select the **Renderers** tab in the Module Contents pane.
3. Click the **Add** button and choose either a **Blank renderer** or a **Copy of** an existing renderer.

   If you choose a blank renderer, select one from the **Type** drop-down list.

   A **New Renderer** tab appears in the Definitions pane.
4. In the **General** tab, give the new renderer a name and supply an appropriate description.
5. Select the **Configuration** tab and set the appropriate properties for the new renderer.
6. Click **Save**.

   The new renderer is added to the selected module.

   IMPORTANT: You can define a new renderer to override the default settings an existing renderer. For example, the default text renderer has the Support Newlines property set to false. If you need newlines, set the property to true. The cost is a slight slowdown in rendering.

To associate the renderer with a data type:

1. In the Module List pane, select the module that should contain the renderer.
2. Select the **Types** tab in the Module Contents pane.
3. Click the **Add** button.

   The **New Type Mapping** dialog box appears.
4. Select the desired data type in the drop-down list, and then click **OK**.

   A **New Type Mapping** tab appears in the Definitions pane.
5. Select its **Renderers** tab.
6. In the left pane underneath this tab, select the same renderer type that you chose in the previous procedure.
7. In the **Renderer** drop-down list, select from the **Available Renderers** dialog box the renderer that you defined previously. Click **Save** in the dialog box.
8. Click **Apply** in the editor pane.
9. Click **Save**.

Binding a property that takes a color or an image to data

Threshold Renderers are configured to provide a color or image (icon) associated with a particular data item. When they are passed a value they will specify a particular color or image, so you bind the property to the data and specify the appropriate renderer, which obtains the correct value for you.

NOTE: If you have a requirement to use different renderers for different sorts of labels, say in table columns, you can use the table's nested grouping feature. Since columns are independent of groups, you can assign a renderer to one column and a different renderer in another column. This avoids the complexity of having to use one renderer and parameterizing it, which may cause alignment problems in the table's columns.

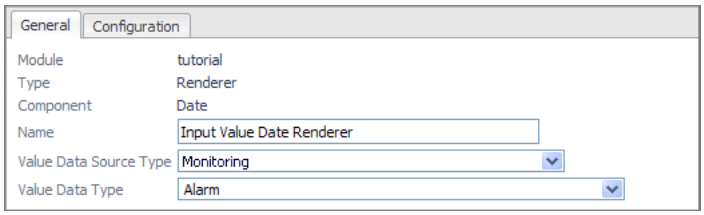
Input Value

The Web Component Framework supports the concept of an Input Value. This is the value to be rendered and is made available in the context under the key *value*.

If you edit a renderer that supports having an Input Value, on the General tab, you can specify the Value Data Source Type, Value Data Type, and Value List. These specify the expected data source type of the value, data

object type, and whether the value will be a list. You can then refer to the Input Value via a Context Selection binding by setting the key of such to *value* and you can also use the context entry as the root of a path in order to access subproperties of that type.

Figure 51. Input values



For example, for a Date Renderer, if you set the Value Data Source Type to Monitoring and the Value Data Type to Alarm, then you can set the Date property to a Context Selection Binding with key=*value*, path=*Created Time* in order to render the Created Time of an alarm as a date.

The default value of Value Data Source Type and Value Data Type depend on the renderer component. The default value of Value List is false.

Determining the Appropriate Renderer for a Binding

Determining which renderer to use for a given binding involves several possibilities. The following list specifies the current possibilities, listed in priority order, which means if the first one does not apply, the second one is used, and so on.

- If a specific renderer is configured for this binding (in the Renderer attribute of the binding editor), use that renderer.
- If the value is a Web Component Framework Data Object, and a renderer is associated with its parent Type and containment Property (using the Child Property Mapping tab of the Type Mapping editor), use that renderer.
- If the value is derived from a child property of a Metric Data Object, and is a Number, and a renderer is associated with the parent Type and containment Property of the Metric (using the Child Property Mapping tab of the Type Mapping editor), use that renderer.
- If the value is a Web Component Framework Data Object, and a renderer is associated with its Type (using the Type Mapping editor), use that renderer.
- If a renderer is associated with the unit of this object (via the Unit Mapping editor), use that renderer.
- If none of the preceding rules apply, use the Default Renderer.

Type Mappings

Type mappings associate entities such as renderers, icons, and tagged flows to a specific data type or data type property. At runtime, Web Component Framework looks up the entities mapped to these types and properties to use as defaults if more specific entities have not been set.

If you configure a view's flow with a flow type of **Select type flow**, the Web Component Framework looks in its type mappings at runtime for a flow associated with the type in question.

You can register different kinds of default flows by specifying tags. You can provide your own by typing its name into the tag name combo box, or choose one of the predefined tags:

- **drilldown**: Used for a flow that takes you to a new page.

- **summary**: Used to retrieve a view appropriate for a dwell.
- **menu**: Used to retrieve a view that gives the user a list of options with a temporary popup.
- **dialog**: Used to flow to a view that is sized to appear overtop of current data, either as a temporary or permanent popup as appropriate.
- **edit**: Used to flow to the appropriate editor for the specified object.

At run-time, if no specific renderer has been specified on a binding, the Web Component Framework will look in the type mappings to see if a renderer has been mapped to the type of the data or the property the data was retrieved from. In the event of a conflict, renderers mapped to properties take precedence over renderers mapped to types.

When an Icon Renderer is explicitly specified as the renderer on a binding, the renderer looks in the type mappings to find the icon that has been mapped to the type of the data evaluated from the binding.

NOTE: You can register a renderer for any child property. This renderer takes priority over a renderer that is registered to the particular type.

Types Tab in the Module List Pane

Mappings of icons to data types and mappings of renderers to data types and properties are configured on this tab. When an icon is mapped to a data type, the icon is displayed if an Icon Renderer is specified on any View Property referencing data of that type. When a renderer is mapped to a data type or property, any view property bound to data of the type or property is displayed using the mapped renderer unless a specific renderer is set on the view property.

The Web Component Framework has a concept of renderers for bindings, which can be used to format the output from the binding in various ways. Definitions for all of the renderers are held in a one-file-per-*wcf.xml* file and in a global file that is common to the whole application. The files are named *renderers.xml*.

The ID attribute is a unique name that is associated with the corresponding renderer. This is the value that is to be specified in the renderer ID of the binding. See Data on page 90.

The optional `<associations>` section specifies associations between the renderer and *sdo* types, metric, classes or units, depending on the tags it contains. Each of the tags in the associations section is optional and more than one association of each type may be specified. These associations are used to resolve the renderer when a binding is rendered.

The `<configuration>` section can be any of the types of configuration mentioned previously. However, each specific class of renderer expects a specific type of configuration.

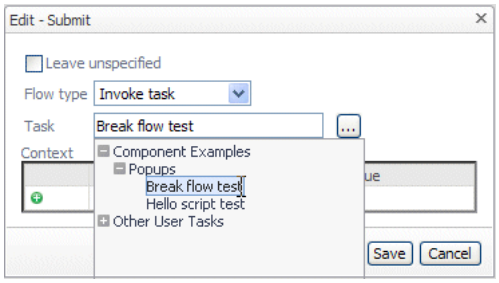
Unit Mappings

Unit mappings associate renderers to a specific data unit. At runtime, if no specific renderer has been specified on a binding and no renderer has been found in the type mappings, the Web Component Framework looks in the unit mappings to see if a renderer has been mapped to the unit of the data.

Mappings of renderers to units are configured in the Units tab. When a renderer is mapped to a unit any view property bound to data of the unit displays using the mapped renderer. This is unless a specific renderer is set on the property or another renderer is mapped to the data's type or property.

Tasks

Tasks are logical actions external to the Web Component Framework. Examples in the Foglight realm are tasks for clearing and acknowledging alarms, building a service, scheduling a report, or launching PerformaSure in context on a particular request. The figure demonstrates the selection of a task as the target of a flow action.



Otherwise tasks are configured to behave in the flow in the same manner as views.

The workflow generally looks like:

- Action on a View
- Invoke component
- Update page

Some tasks may break the flow – that is, not return. An example of this is launching PerformaSure. In this case the task does not have a flow action that can be configured to update the page.

A task consists of its code and three text files, and is usually packaged in a JAR file.

Table 27. Components of a Web Component Framework Task

descriptor.xml	Location of the code for the task and its characteristics.
types.xml	Configuration schema: the task's properties.
strings.properties	The English text for the component. This provides a default localization. Other localization files have names like *strings_fr.properties*.

Tasks are located in *META-INF/wcf-metadata/task/X.Y,* where X.Y is the Web Component Framework internally generated name for the component.

NOTE: You can use the Execute Groovy Task component to implement a simple task. See the Web Component Tutorial for an example.

Types

If you are building your own complex user interface for some special purpose, you may want to create temporary objects to manage it. You can create your own types in the Web Component Framework.

When building a complex user interface it is common to discover the need to create temporary objects to manage the UI you are creating. You may want to capture input from users to convert later into a more persistent object or to remember choices that a user has made in the past in order to streamline the user's workflow in the future.

There is a mechanism to allow the creation of types in the Web Component Framework that are specific to the UI mechanism and associated with the modules that contain the rest of your UI artifacts.

Type entities allow you to define types and their properties which can be used in various ways in your UI.

Creating a Data Type in a Web Component Framework Module

NOTE: You can only create a type in a cartridge module. This option is not available in any user module.

To create a Module Type:

1 Choose **Types** in the Definitions editor and select **Add** to create a new type.

The **Add Type** dialog box appears.

2 Select a super type for your new data object in the dialog box You can type a partial name in the *Search Type* text box and press **Enter** to filter the list of super types.

Two choices are listed, Data Object types and Enumeration types. The editor you see depends on which type is chosen as the super type.

3 Name the new type.

Data Object Types

Properties and Annotations can be specified for data object types.

Properties

At least one property must be defined. The following attributes may be set.

- **Name**: The name of the property
- **Display Name**: The localizable display name of the property. The value specified in the editor is used as the value for the base locale.
- **Data Type**: The type of the property
- **List**: Indicates whether the value of the property is a list or not.
- **Description**: The localizable description of the property. The value specified in the editor will be used as the value for the base locale.
- **Reference**: (Only applicable to properties of a data object type.) Whether the value of the property is a reference to a Data Object that exists somewhere else. For properties of an enum type, *Reference* is always true.
- **Default Value**: The default value for the property. Only supported for non-list properties of a primitive type that the Web Component Framework can persist. For *enum* property types, this is supported if Type has the *keyProperty* annotation set.
- **Annotations**: For editing the property *Annotations*. When you click on the edit icon, a dialog box is displayed showing the available annotations. The listed annotations depend on the super type of the type. If you hover the cursor over the name of an annotation in the table, the description of the annotation is shown in a tooltip. See Code Annotations below for more information about annotations for data objects of the Writable Data Object type.

Code Annotations

Some of the annotations for the `WritableDataObject` type (for example, `Expression` and `Validator`) are executed as a script, as Java™ code, or as a WCF function depending on the value of the corresponding annotation Type (for example, `Expression` type and `Validator` type).

TIP: Select Java™ or Function as the type in cases where you want WCF to execute the annotation by referencing logic that is stored elsewhere (a reusable portion of code). Select Script when you do not require this type of re-use and it serves your needs to execute the annotation as a single-use script.

Java Code Annotations

You can specify a Java method as the value of an annotation. When an annotation refers to a Java method, use the following syntax to specify it:

```
<Fully Qualified Class Name>.<Method Name>
```

The method you specify must accept a single parameter of the following type: `com.quest.wcf.publicapi.code.WritableDataObjectHelper.`

Script Annotations

If an annotation is a script, the value that you specify for the annotation is the name of the script.

You can use the following special variables with script annotations; they are available for use when the script is run:

- `object`: The `WritableDataObject` instance whose annotation is being evaluated.
- `property`: The property that is acted upon. For example, for `Expression`, the property being evaluated.
- `specificTimeRange`: The current specific time range.
- `helper`: The instance of `WritableDataObjectHelper` that the script can use.

Function Annotations

If an annotation is a function, the value that you specify for the annotation and should be executed is the fully qualified ID of a WCF function. See Functions on page 73 for more information.

Annotations

The annotations of the type may be edited. Mousing over the name of an annotation in the table causes a tooltip to be shown containing the annotation's description. Each annotation value can be edited or set to Inherit, which means that the value of the annotation is inherited from the corresponding annotation of the super type.

Enumeration Types

Enumeration values can be specified for Enumeration types.

At least one enum value must be defined. The following attributes may be set:

- **Value**: The value that represents the enum value
- **Name**: The display name of the enum value. The value specified in the editor is used as the value for the base locale.
- **Description**: The description of the enum value. The value specified in the editor is used as the value for the base locale.

Creating a Data Object of your Type

Instances of your type are created in one of the following ways:

- With the Create Binding mechanism that is available when specifying context or binding to entity properties.
- In Java™ or Groovy WCF via the FunctionHelper's `createDataObject()` method.

Parameters to the creation mechanism are:

- **Type**: The type you want to create. You can only select from Data Object types (not including Enum types) from the Common Data Source or other UI Types.
- **Storage**: This is the persistence mechanism to use to "save" this object. The create method will try to load this object from the storage mechanism specified and create a new instance of it if it does not exist.
    - **None**: This object cannot be stored and is meant to be transitory (used on the page or passed through the context).

    - **Site**: Enables the object to be stored in a global location that is accessible to all users. If multiple users are changing the object at once from different sessions then the last copy of the object saved will be persisted.
    - **User**: This object is stored per user so that each user has a different copy of this object. This is useful to remember a user's particular choices.
    - **Session**: This object is stored in the user's *http* session. Calling *save* is not necessary to persist changes, since the single live instance is stored in a session-owned cache. The object is released when the user logs out or the object's *remove* method is called.
- **Object ID**: It should use the form *XXXXX-YYYYY*, where:
    - *XXXXX* is your domain identifier. For example, *java*, *oracle*, *ops*, *admin*
    - *YYYYY* is the identifier for the object. For example preferences, *socFilter*, etc.

An object that is created from scratch has all of its properties set to their default values.

Objects created using this mechanism are not time range sensitive.

Objects loaded from the *Site* and *User* Storage mechanisms are always loaded from the store and not persisted in any cache other than the Web Component Framework page context if they are used within it. Note that these object are not in Web Component Framework pages in the same way that Topology objects are.

Equality

Data Objects created through this mechanism are considered equal if their uniqueIds are equal.

Saving Data Objects

Changes to an object are persisted when the object is acted upon by calling the object's `save()` method or by using the *WCF/Common/Save Data Object* function to do this for you.

An object that has not been created with a storage mechanism of either *Site*, *User*, or *Session* cannot be saved.

Removing a Data Object from a Storage Mechanism

An object using *Site*, *User*, or *Session* storage can be removed from those storage mechanisms by calling the `remove()` method on the object or calling the *WCF/Common/Remove Data Object* Web Component Framework function.

Saving Contained versus Referenced Properties

Note these points:

- A contained property is saved along with the parent.
- A reference property (if it is a reloadable object) is saved as a reference that is retrieved and set when the parent is re-created.
- A reference property that is not reloadable will be reloaded as its default value or null when the parent is re-created.
- Simple types (Strings, Numbers, and so on) cannot be references. Enum types are always references.

For more information, see Writable Data Object on page 83.

Setting up RSS Feeds

You can set up a syndication feed for Foglight views, using Really Simple Syndication (RSS) to show frequently updated content. For some Foglight views, such as setting up an RSS feed for alarms, this is an out-of-the-box feature.

In the following example, we create an out-of-the-box RSS feed for the Alarms page.

To create an out-of-the box RSS feed for the Alarms page:

NOTE: To access the Web Component Framework for creating an RSS feed, you need the role of Cartridge Developer or Dashboard Designer assigned to your username.

1. Under **Dashboards**, click **Configuration** > **Definitions**.
2. Set the **Feeds** purpose for the particular Alarms page.
3. Determine the dashboard name by viewing the dashboard definition in **Configuration** > **Definitions**. The dashboard name is the `Reference Id` property.

   Figure 52. Reference Id property

4. To view the feed for the alarms view, type the URL for the feed such as:

   ```
   http://<fms hostname>:8080/console/feed/system:core_alarms.28
   ```

   The RSS feed for the alarms view appears.

Theme and Module Resources

Your modules can contain image files, such as icons, logos, and background images, that can be used in views.

Views can reference images defined in the container module. Some of the referenced images never change, such as product logos. For more information, see Module-Specific Images on page 114.

For other types of views, such as localized versions of the same view, you need to create modified versions of the same image and display different versions at different times. For more information, see Theme-Specific Images in Module Definitions on page 116.

Module-Specific Images

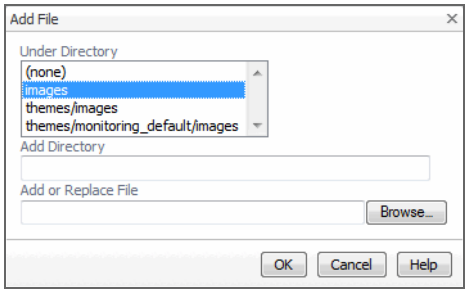
These types of images never change. For example, you can have a set of images that always appear in report headers or footers, regardless of the selected locale or theme. Module-specific images must be stored in the module's *images* directory.

The following list illustrates the directory structure required to create module-specific images.

```
<module>/
   images/
      <imageA>
      <imageB>
```

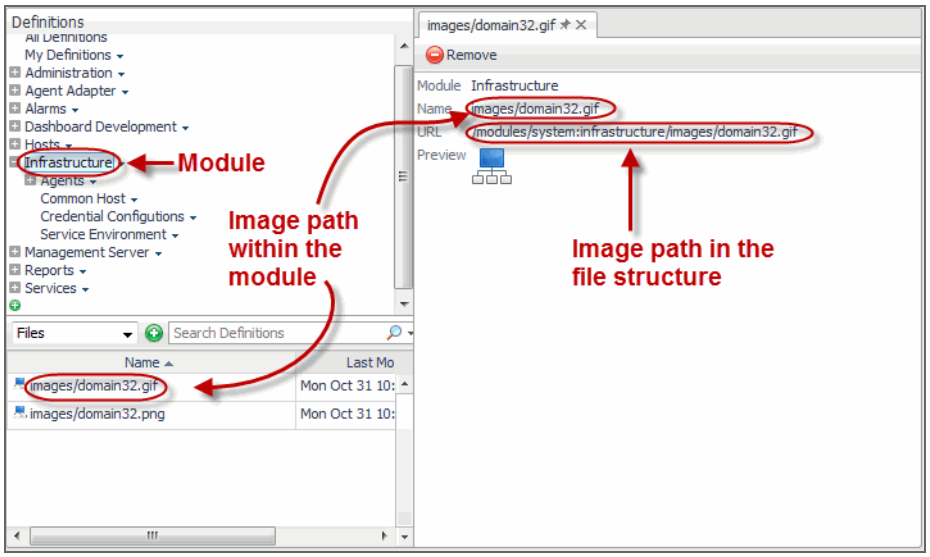
You add images to modules and create the required directory structure by adding files to your module. For more information, see Files on page 94.

Figure 53. Add File dialog box



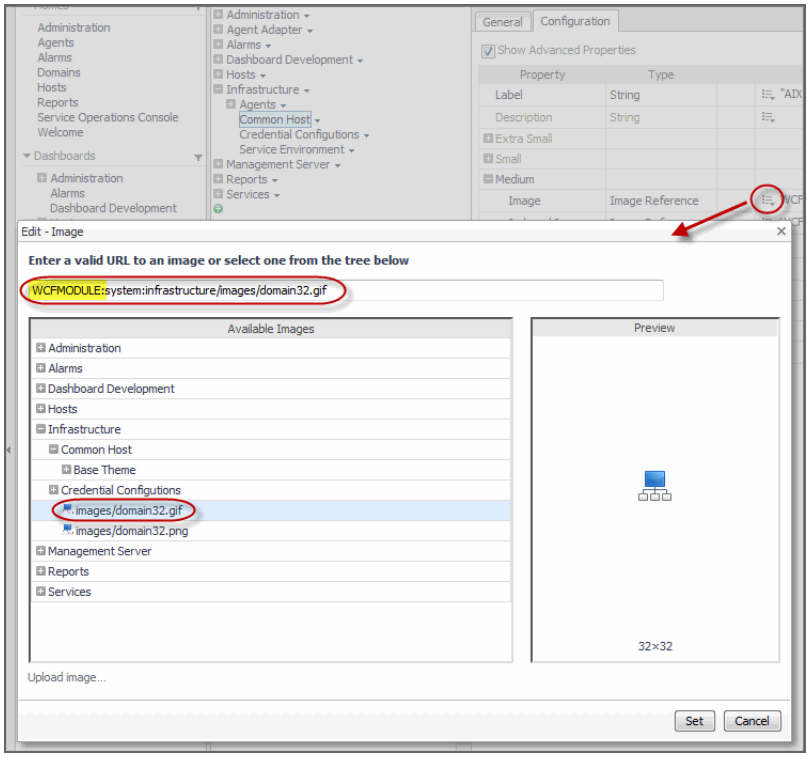
When you add an image file to the module, its path is displayed in the Module Contents pane and in the image file properties:

Figure 54. Image path



When you reference a module-specific image, the `WCFMODULE:` resource tag in the image URL indicates that an image file is defined in the module, and that it does not have any theme-specific variants.

Figure 55. WCFMODULE: resource tag



Theme-Specific Images in Module Definitions

In a default installation, Foglight® browser interface includes two default themes: *Application* and *Monitoring*. These themes allow you to control the background color for dashboards. For more information about these themes and other ways to customize the browser interface, see the *Foglight User Help*.

Theme-related sets of images allows you to change the look and feel of your views, and to personalize them to suit specific end-user needs. Your module can have one or more *themes* that enable the display of different styles, locales, and view sizes.

The implementation of themes relies on the existence of a *base theme*. The base theme contains a set of default images. When you reference a theme-specific image in a view or icon definition, you can only select a reference image from the base theme. When that view or icon appears in the browser interface, the system displays the image variant defined in the currently selected theme (such as Application or Monitoring), or the locale. This means that you only need to provide images for the themes whose images differ from the base theme images. For example, most images included with the Management Server have only the base theme and Monitoring theme variants because the Application theme is the default theme. The image variants for this theme are supplied in the base theme, which eliminates the need for a separate set of images for the Application theme.

To implement themes and enable selective access to images, create a directory structure in your module using the following guidelines:

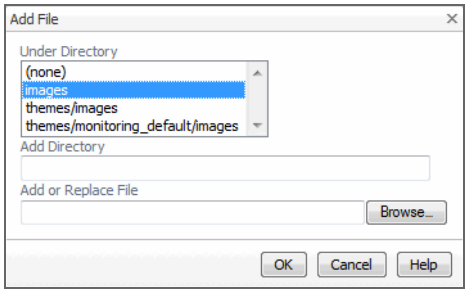
- Themes are stored in the module's *theme* directory. This directory must contain the base theme images in its *images* sub-directory.
- Each theme, except the base theme, must have its own directory. The directory name is not necessarily the same as the theme's localized name. For example, the Monitoring theme uses `monitoring_default` as the directory name.
- Each theme, except the base theme, must have an *images* sub-directory.
- Theme-related images must be stored in the theme's *images* sub-directory.

The following list illustrates the directory structure required to implement themes.

```
<module>/
   themes/
      images/
         <imageA-base>
         <imageB-base>
      <theme1>/
         images/
            <imageA-theme1>
            <imageB-theme1>
      <theme2>/
         images/
            <imageA-theme2>
            <imageB-theme2>
```

You add images to modules and create the required directory structure by adding files to your module.

Figure 56. Adding images to modules

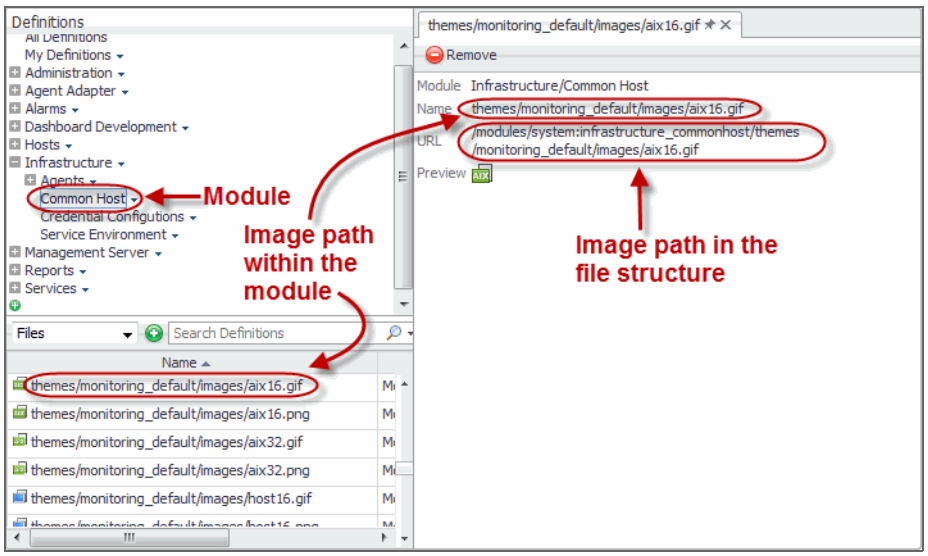


For more information, see Files on page 94.

For example, you can have two different themes for two monitoring modes, Application and Monitoring. This requires a base theme directory, *images* (located under *<module>/themes/images*). If you need a separate set of images for the Application and Monitoring themes, you need to create image directories for those themes: *<module>/themes/monitoring_default/images* and *<module>/themes/application_default/images*. Theme-specific images are optional and only required if a theme is different from the base theme.

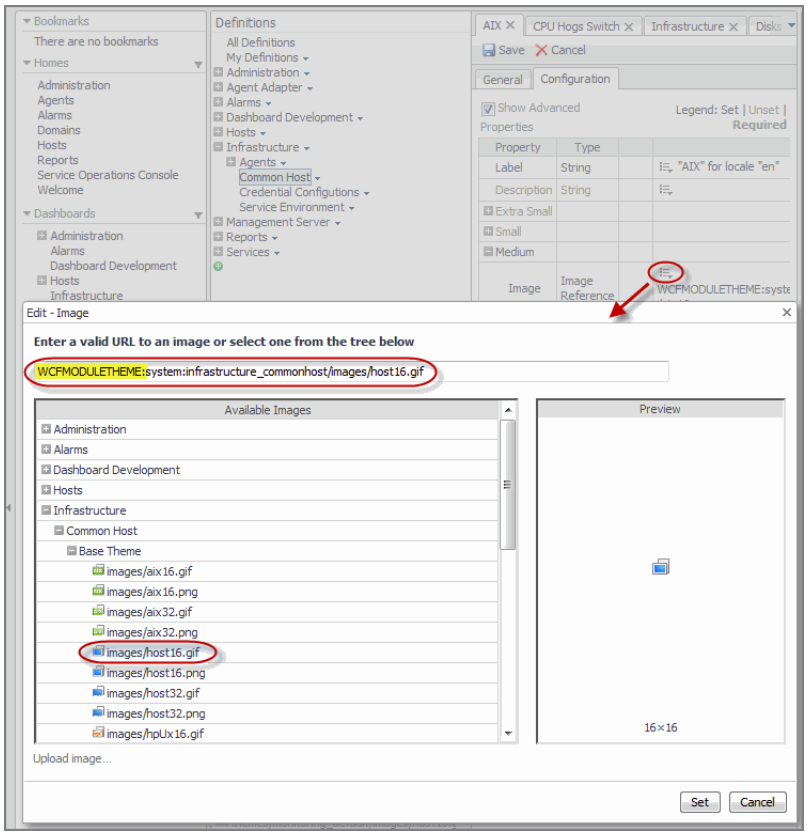
When you add an image file to a theme in a module, the image path is displayed in the Module Contents pane and in the image file properties:

Figure 57. Image path



When you reference a theme-specific image in a view or icon definition, the `WCFMODULETHEME:` resource tag in the image URL indicates that a theme-specific image is selected. Specifying the image in the base theme instructs the system to display the image variant defined in the currently selected theme.

Figure 58. WCFMODULETHEME: resource tag



Printing

There are two different mechanisms available for printing in the Web Component Framework:

- PDF Generation
- Reports

PDF Generation

To produce a printable version of a view in PDF format you need to create a report. Simple reports can be produced by selecting **Create Report** in the action panel. Drag in the views you want and click **Create PDF**. See the *Foglight® User Guide* for additional details. Custom reports can be produced using the PDF Layout component. See the *Web Component Tutorial* for instructions on using this component.

One of the major advantages of using PDF printing over browser printing is that PDF printing renders a chart as lines so that they can be printed with the full resolution of your printer, where browser printing limits the resolution of charts to that of the screen.

Another advantage of PDF printing is that views built with the PDF Layout component have the ability to have page-friendly features such as custom headers and footers, which include features such as page numbers and titles. In addition, it is possible to control the orientation of each page, and where pages break.

Reports

Reports are useful both for informative and archival purposes. You can use the PDF Layout container to structure a group of views that you deem useful for dissemination to interested parties or simply to keep as a historical record.

Reports in PDF format may be generated in color or in black and white (monochrome).

The Page Decoration component is used to define headers and footers in a PDF layout if the intention is to convert the report to PDF format. If you use them to define a header and a footer for a report, both header and footer elements must be placed before the body pages. This design is required so that footers will appear in the case where the body component is a table whose rows may span several pages.

It is possible to arrange for different headers and footers after the first page, and in different sections of a large report. Simply add more Page Decoration components after body views whenever new ones are required. You can emit a page break both before and after any view, which gives added control over the layout, such as permitting you to start a new section on a fresh page.

See the *Web Component Reference* pages on PDF Layout and Page Decoration for detailed descriptions of these components, or refer to the *Web Component Tutorial*.

Report Layout

The PDF Layout component is designed to permit multi-page reports. It does this by checking if there is enough space for the next view, and if there is not, generating a new page. Reports also have special view types for headers and footers which are used to generate running headers and footers on the pages of the reports. A footer must be specified before the view on the first page it is to appear because it must be available when the page is generated in order to allocate the correct amount of space for it. Headers and footers can be changed later on in a report by re-specifying them among the body views of the report.

Once the report determines where on a page to place a view, that view is drawn in the report's graphics, as opposed to the report generating a graphics for the view to use. This means that reports which are nested in a parent report can also span multiple pages, but it also means that the report is effectively transparent, and can not have a background or border. Nested reports can not specify headers or footers—only the top-level report is able to create them.

NOTE: Container views that use a Fixed layout print in the sizes that are strictly specified by those views and do not look substantially different than they do on the page. Most tables, either printed directly as a report or embedded in a report, are able to flow to more than one page. For this to work, the table can only be nested in some combination of reports and iterators. In all other cases (for instance, a table in a fixed or grid layout) the table has a maximum size of a single page, and will clip to that.

NOTE: When you generate a PDF file for a table that exceeds the number of rows that fit on a single page, specify the table as either a top level view or include the table in the PDF Layout container.

Scheduling Reports

It is possible to set up a schedule in the **Administration** module under *Dashboards* in the navigation panel so that reports are generated automatically at regular intervals. See the *Administration and Configuration Guide* for details on setting up a schedule and associating a report generation task with it.

Remote Access to Views

This section describes the mechanisms that are available for embedding a Web Component Framework View in your application. The following are supported:

- Portlet (JSR 168)

- SharePoint Web Part with Windows Single Sign-On

Portlet

This is a JSR168 compliant portlet created for the purpose of allowing remote access to browser interface's views. So far, this portlet has only been successfully tested on these Portal Servers as shown below:

- JBoss® Portal Server 2.6
- Apache Pluto 1.0.1
- WebLogic® Portal 10

Deploy and Run

In order to test your Foglight® portlet, you will need to have a portal server available in which you can place the portlet's *war* file. JBoss Portal, Apache Jetspeed2, and WebLogic Portal are suitable choices for testing a remote portlet.

To test your portlet:

1. Locate the Foglight remote portlet *war* file:
   ${Foglight Server Home}/tools/foglight-remote-portlets.war
2. Deploy the portlet.

   The specific steps depend on which portal server you use. Please follow the guidelines given in your particular portal server for deploying portlets. For instance, to deploy to Pluto and JBoss Portal Server, the simplest way is to simply copy the *war* file into the server's deploy directory.
3. Access the portlet.

   Once the portlet is successfully deployed, be sure to follow the user guide of your portal server for information on how to access its deployed portlets.

Configuration

You need to configure the portlet by setting its **host**, **port**, and **viewId** before attempting to run it.

- **host**: The Management Server host you would like to connect with.
- **port**: The Management Server port you would like to connect with.
- **viewId**: The Web Component Framework fully qualified ID, for example, *system:fsmhome.0*. There are also predefined view mappings available for use.

NOTE: Validations for all these settings are also available.

SharePoint Web Part with Windows Single Sign-On

Users without a Foglight® account cannot have access to Foglight dashboards appearing in SharePoint® Web pages. Configuring Foglight to support Windows® Single Sign-on, grants Active Directory users access to Foglight views embedded in SharePoint Web pages.

When Windows Single Sign-on is configured, if you log in to a Windows machine and then log in to the browser interface, the Management Server uses your Windows account credentials to authenticate you as a Management Server user.

IMPORTANT: When accessing SharePoint Web pages that contain embedded Foglight views, Active Directory users must use Web browsers that support SPNEGO authentication.

For complete information about configuring Foglight with Windows Sign-on, see “Configuring Windows Single Sign-On” in the *Administration and Configuration Help*.

Active Directory® users who need to have access to Foglight dashboards in SharePoint Web pages must have proper permissions granted to their Active Directory user accounts in Foglight. For more information, see “Managing Users and Security” in the *Administration and Configuration Help*.

To embed a Foglight dashboard into a SharePoint Web page:

1 Using your Active Directory account, log in to the Foglight browser interface.

2 Navigate to the dashboard that you want to embed into the SharePoint Web page.

3 Copy and record a link to this dashboard.

a On the action panel, on the **General** tab, under **Actions**, click **Properties**.

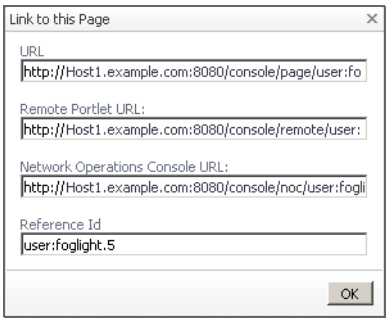
A menu appears.

Figure 59. Properties menu

b Click **Link to this page** in the menu.

The **Link to this Page** dialog box appears.

Figure 60. Link to this Page dialog box



c In the **Link to this Page** dialog box, copy and record the value from the **Remote Portlet URL** box.

d Click **OK** to close the **Link to this Page** dialog box.

4 Copy and record the preferred size for this view.

a On the action panel, open the **Design** tab.

b On the **Design** tab, ensure that the **Definition** tab is open.

c On the **Definition** tab, under **Configuration > Sizing**, copy and record the preferred width and height.

5 In SharePoint, navigate to the SharePoint page where you want to embed the Foglight dashboard.

6 Add a Page Viewer Web Part to the SharePoint Web page.

7 Start editing the Page Viewer Web Part by choosing **Modify My Page > Modify My Web Parts > <Newly Added Page Viewer Web Part>**.

8 In the **Link** box, type the link to the Foglight dashboard recorded in Step 3.

9 Under **Page Viewer**, ensure that **Web Page** is selected

10 Expand the **Appearance** section and specify a title for the Foglight dashboard.

11 Set the dashboard's height and width using the preferred size properties recorded in Step 4.

12 Ensure that **Frame State** is set to **Normal**.

13 Leave other properties at their default values.

14 Click **Apply** to apply your changes.

15 Click **Save**.

Performance Improvements

This section contains information that can help you improve the performance of your dashboards.

Optimizing Data Access with the Batch API

Batch data access results in optimized data access through single back-end round trips. This is highly advisable for dashboards that load more than a handful of metrics.

For example, you may have a dashboard showing thousands of hosts and their metrics in a table. Populating these metric values can results in numerous back-end round-trips, which affects the table performance when any of the following actions are performed:

- Sorting a column that points to a metric
- Viewing the table if an initial sort is defined for one of the columns
- Calculating a maximum column value
- Searching the table

First, you need to write a function that triggers the Batch API mode. The batch data access API is exposed on the Data Service through the following APIs:

```
/**
* Returns a new query object that can be configured to specify the data that is
* to be returned.
* @return the blank query object
*/
ObservationQuery createObservationQuery();
/**
 * Performs the query specified in the query object and returns the resulting data.
 * @param query the query to perform
 * @return the queried data
 */
ObservationQueryResult performQuery(ObservationQuery query);
```

The `ObservationQuery` interface includes methods to set the time range, retrieve observations, and the retrieval type. When used from WCF, the start and end times of the time range are set based on the `SpecificTimeRange` object provided by WCF. If the metrics are used in chart components, a `SpecificTimeRange` object for charts can be obtained by calling the method `functionHelper.getSpecificTimeRangeForCharting()`.

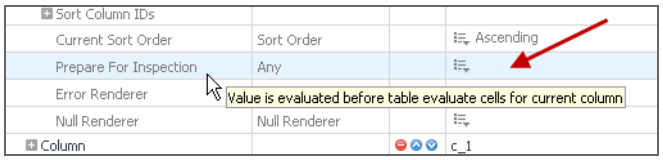
NOTE: Any metrics used by the dashboard implementing the Batch API mode need to be added to the set of observations that are to be retrieved. There are no restrictions on which observation keys can be added to the set: they can be obtained from different topology objects, and can be of different types (such as Metrics, StringObservations, and so on). The retrieval type defines the type of data access. If the view is displaying current and period values, then the retrieval type should be AGGREGATE_AND_LAST (the default). The other options are AGGREGATE, if only the period value is required, LAST_N if only the current value is required, or RAW if raw metric values are required.

The following Groovy script has an example of how you can call the batch API. The results of the batch data access call are not important for the observation pre-loading to work, and can be discarded by the Groovy script. WCF is not aware of batch queries and the dashboards will continue to access the observations in the usual way, but that access will not require any database round trips because the observations will be in the Persistence Cache after the batch loading script completes.

```
import com.quest.nitro.service.sl.interfaces.data.IDataService;
import com.quest.nitro.service.sl.interfaces.data.ObservationQuery;
import com.quest.nitro.model.topology.TopologyObject;
import com.quest.nitro.service.sl.ServiceLocatorFactory;
import org.apache.log4j.Logger;

def LOG = Logger.getLogger("batch.query.test");
topologyObjects = new HashSet(#! RequestType #.topologyObjects);
LOG.info("Querying ${topologyObjects.size()} topology objects...");

endTime = System.currentTimeMillis();
startTime = endTime - (4 * 60 * 60 * 1000L);

IDataService dataSvc = ServiceLocatorFactory.getLocator().getDataService();
ObservationQuery query = dataSvc.createObservationQuery();
query.setStartTime(startTime);
query.setEndTime(endTime);
query.include(topologyObjects, "responseTime");
query.include(topologyObjects, "invocations");
result = dataSvc.performQuery(query);

long duration = System.currentTimeMillis() - endTime;
LOG.info("Query completed in ${duration} ms.");
```

Next, in WCF, to enter the Batch API mode, invoke that function by using the *Prepare For Inspection* property.

Figure 61. *Prepare For Inspection* property



The Tree Table and Row-Oriented Table components include this property. The calling component does not take into consideration the result of this function, so as long as you call the function this way, it starts the Batch API mode.

For example, a dashboard can list thousands of hosts, including their names and different metrics (for example, Metric A, B and C). This data displays in columns of a Row-Oriented Table component. If Metric B is more important than metrics A and C, so the user should be able to sort this column or search for a specific value. The layout cannot display all of the rows, so for example, it only shows 20 rows. If the end-user attempts to sort the column containing Metric B, that sort will not be very fast because it requires a round-trip to server to get that value for each host.

To overcome this problem, the function you write to start the Batch API mode can take a list of topology objects (hosts) and the Metric B name as inputs. For example:

```
import com.quest.nitro.service.sl.interfaces.data.IDataService;
import com.quest.nitro.service.sl.interfaces.data.ObservationQuery;
import com.quest.nitro.model.topology.TopologyObject;
import com.quest.nitro.service.sl.ServiceLocatorFactory;
import org.apache.log4j.Logger;
topologyObjects = args["topologyObjects"]
metricName = args["metricName"]
IDataService dataSvc = ServiceLocatorFactory.getLocator().getDataService();
ObservationQuery query = dataSvc.createObservationQuery();
```

```
query.setStartTime(specificTimeRange.getStart().getTime());
query.setEndTime(specificTimeRange.getEnd().getTime());
query.include(new HashSet<TopologyObject>(topologyObjects), metricName);
dataSvc.performQuery(query);
return true;
```

At run-time when the table is rendered, if the user sort the column that contains Metric B, this function is invoked and the Batching API mode starts. Row objects with all instances of Metric B (one for each host) are cached, which eliminates the need for subsequent round trips to the server to retrieve individual Metric B values. If you want to apply this behavior to Metric A and Metric C, invoke the function for each column.

About Us

We are more than just a name

We are on a quest to make your information technology work harder for you. That is why we build community-driven software solutions that help you spend less time on IT administration and more time on business innovation. We help you modernize your data center, get you to the cloud quicker and provide the expertise, security and accessibility you need to grow your data-driven business. Combined with Quest's invitation to the global community to be a part of its innovation, and our firm commitment to ensuring customer satisfaction, we continue to deliver solutions that have a real impact on our customers today and leave a legacy we are proud of. We are challenging the status quo by transforming into a new software company. And as your partner, we work tirelessly to make sure your information technology is designed for you and by you. This is our mission, and we are in this together. Welcome to a new Quest. You are invited to Join the Innovation™.

Our brand, our vision. Together.

Our logo reflects our story: innovation, community and support. An important part of this story begins with the letter Q. It is a perfect circle, representing our commitment to technological precision and strength. The space in the Q itself symbolizes our need to add the missing piece—you—to the community, to the new Quest.

Contacting Quest

For sales or other inquiries, visit https://www.quest.com/company/contact-us.aspx.

Technical support resources

Technical support is available to Quest customers with a valid maintenance contract and customers who have trial versions. You can access the Quest Support Portal at https://support.quest.com.

The Support Portal provides self-help tools you can use to solve problems quickly and independently, 24 hours a day, 365 days a year. The Support Portal enables you to:

- Submit and manage a Service Request.
- View Knowledge Base articles.
- Sign up for product notifications.
- Download software and technical documentation.
- View how-to-videos.
- Engage in community discussions.
- Chat with support engineers online.
- View services to assist you with your product.