ORBIT MAPVIEW

CUSTOMIZABLE SCREEN CONTROL AND MONITORING

User Manual

Issue 1 Revision 3

18 May 2020

www.grassvalley.com

Patent Information

This product may be protected by one or more patents.

For further information, please visit: www.grassvalley.com/patents/

Copyright and Trademark Notice

Copyright © 2020, Grass Valley Canada. All rights reserved.

Belden, Belden Sending All The Right Signals, and the Belden logo are trademarks or registered trademarks of Belden Inc. or its affiliated companies in the United States and other jurisdictions. Grass Valley, Orbit and Orbit MapView are trademarks or registered trademarks of Grass Valley Canada. Belden Inc., Grass Valley Canada, and other parties may also have trademark rights in other terms used herein.

Terms and Conditions

Please read the following terms and conditions carefully. By using Orbit MapView documentation, you agree to the following terms and conditions.

Grass Valley hereby grants permission and license to owners of Orbit MapViews to use their product manuals for their own internal business use. Manuals for Grass Valley products may not be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose unless specifically authorized in writing by Grass Valley.

A Grass Valley manual may have been revised to reflect changes made to the product during its manufacturing life. Thus, different versions of a manual may exist for any given product. Care should be taken to ensure that one obtains the proper manual version for a specific product serial number.

Information in this document is subject to change without notice and does not represent a commitment on the part of Grass Valley.

Warranty information is available from the Legal Terms and Conditions section of Grass Valley's website (www.grassvalley.com).

About this Manual

Title	Orbit MapView User Manual	Issue 1 Revision 3
Revision	2020-05-18, 12:29	

This Orbit MapView User Manual describes how to use the Grass Valley Orbit software for Grass Valley MapView control and monitoring applications.

General information about Orbit is found in the'Orbit - Introduction User Manual', and general information about Orbit graphical tools is in the 'Orbit for Multiviewers User Manual'.

Table 1 Related Documents

Type	Title	Description
User Manual:	Orbit - Introduction	General introduction to the Orbit software and description of common main menu items etc.
User Manual:	Orbit for Multiviewers	Multiviewer-specific features of Orbit.
User Manual:	Orbit Services	Describes Orbit services.
User Manual:	Orbit Router Control Panel	Describes the Bulk Routing Panel openable from Orbit.

Table of Contents

1 Product Overview

Orbit is a configuration, control and monitoring system from Grass Valley, a Belden brand.

Orbit MapView provides customizable, fully-featured, scalable control and monitoring screens for Grass Valley and for third-party devices. This allows multiple channels with large amounts of status information to be observed, including monitoring by exception. Control, monitoring and status screens can be viewed on standard client computer display monitors and also on Grass Valley multiviewer video walls. Graphical custom tiles can be created to filter and monitor status and alarms from devices. Custom screens can be displayed, or actions can be triggered, by the status of one or more devices and device status parameters.

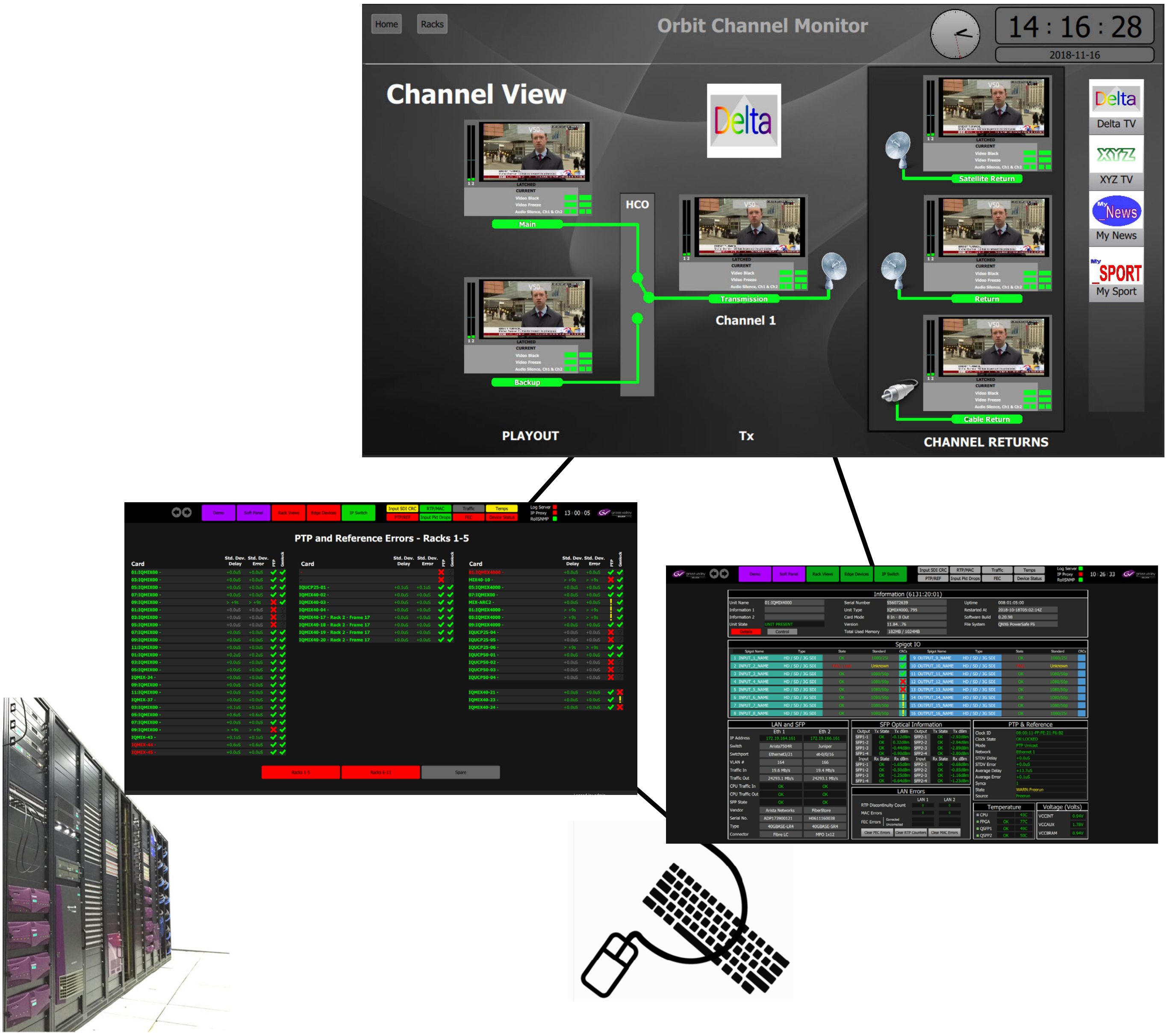


Fig. 1-1: Orbit MapView

Order Codes

Table 1-1 Orbit MapView Order Codes

Order Code	Description
Orbit	Orbit application.
Licenses:	(For license installation instructions, see the Orbit Introduction User Manual.)
FGAN MAPVIEW	License for using Orbit for MapView applications in run mode.

Software Version

The Orbit MapView functionality described in this document is for Orbit MapView using the following version of the Orbit client application and Orbit Services:

Orbit 3.1 or later

This Document

This user manual presents some Orbit MapView information and examples of using Orbit MapView.

Content Summary

- Further Orbit MapView product information,
  see System Overview, on page 4, onwards.
- Orbit MapView home screen,
  see Chapter 2 MapView Home Screen, on page 9, onwards.
- Elements of Orbit MapView screens,
  see Chapter 3 Working with Widgets, on page 41; and
  see Chapter 4 Components and Variables, on page 83.
- Introduction to some Behaviours and Bindings used in examples in this document,
  see Chapter 5 Bindings, on page 123; and
  see Chapter 6 Behaviours, on page 147.
- Some examples using Behaviours and Bindings,
  see Chapter 7 Examples with Bindings, on page 183; and
  see Chapter 8 Examples with Behaviours, on page 229.
- Some examples using Server-side processing,
  see Chapter 9 Server-side Processing Examples, on page 281.
- Custom logic definition,
  see Chapter 10 Custom Logic, on page 315.
- Presentation of a channel monitoring example,
  see Chapter 11 Channel Monitoring Example, on page 331.
- And finally, appendices:
  see Design Tips and Shortcuts, on page 345; and
  see List of Behaviours and Bindings, on page 355; and
  see Troubleshooting, on page 359;

Useful Bookmarks

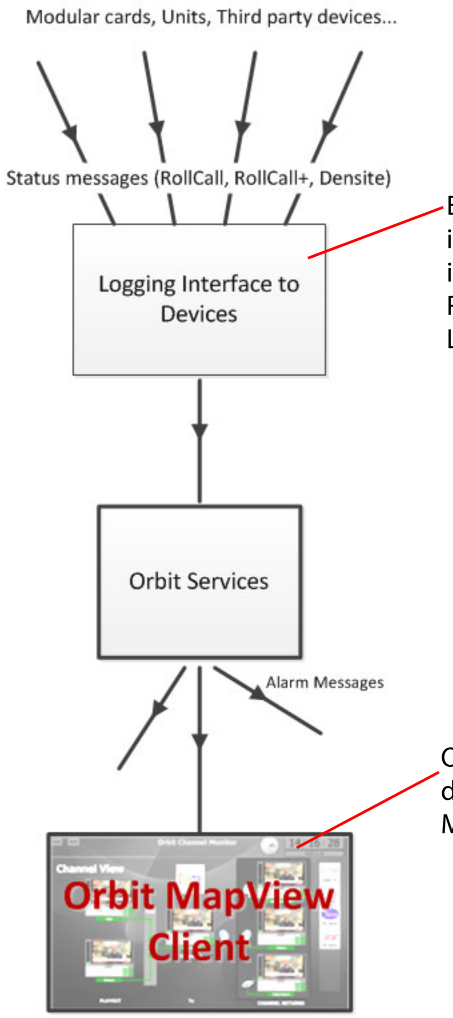
Other useful bookmarks within the document:

- A Behaviour, on page 6, and A Binding, on page 6.
- Accessing Device Control Screens, on page 25.
- Widget Properties and Value, on page 42.
- Setting Up a Test Bench, on page 46.
- Monitoring by Exception, on page 62.
- Format Syntax of a String Value, on page 71.
- Viewing Log Messages (Alarm List Widget), on page 75.
- Variables in Orbit, on page 86.
- Connect Widget-Level Bindings to Component Variables, on page 93.
- Behaviour Arguments, on page 149.
- Alarm Behaviour, on page 150.
- Log Field Behaviour, on page 159.
- #Hash Field# Syntax, on page 164, and <Angle-Bracket> Syntax, on page 166.
- Link Behaviour, on page 174.
- Binding Execution Order, on page 225.
- Example - Screen Link States and Screen Re-Use with Variable Files, on page 243.
- Orbit Global Files for Server-Side Processing, on page 282.

Note: For general information on graphical entry in Orbit, refer to the 'Orbit for Multiviewers' user manual, where Orbit schematic editor, Theme editor, Behaviours/Bindings, and managing users are described.

System Overview

Orbit MapView runs on a client computer. MapView screens can form a soft user panel for monitoring various status items of many devices. A MapView screen can monitor one or more Grass Valley, or third party, devices, cards, modules or units: Additionally, it can connect to server-based services which provide overall system status information, services and virtual alarms. Figure 1-2 shows an outline of a general system view for Orbit MapView.


Modular cards, Units, Third party devices...
Status messages (RollCall, RollCall+, Densite)
Logging Interface to Devices
Orbit Services
Alarm Messages
Orbit MapView Client

E.g.
iControl Densité Manager,
iControl GSM,
RollCall Middleware Services.
Log Server.

Orbit MapView displays alarm state and device status to the user with Orbit MapView screens.

Fig. 1-2: System Diagram with Orbit MapView

Devices in a system send log messages. (For example, RollCall-enabled devices send log messages with a 'LOG_HEADER=Value' form, known as Log Fields.) Log information may be informative, or may carry alarm information which needs to be highlighted on a monitoring screen.

'Raw' log information is sent to a device logging interface which is usually a service running on a server (for example, an iControl Densité Manager). In turn, status messages are sent to Orbit Services (for example, Orbit Monitoring Service, OMS). Aggregate status and alarm messages are published, which an Orbit MapView client can subscribe to (via an Orbit **Alarm** Behaviour).

Note: In a system, the RollCall domain should match across:

- Logging Service
- Orbit Service(s)
- Orbit MapView Client.

And each item must have a unique RollCall address.

Orbit MapView Project

Screen Hierarchy

An Orbit MapView project defines a soft control and monitoring panel comprising various linked graphical screens for control and monitoring applications. A project may contain a hierarchy of screens. See the example hierarchy of Figure 1-3.

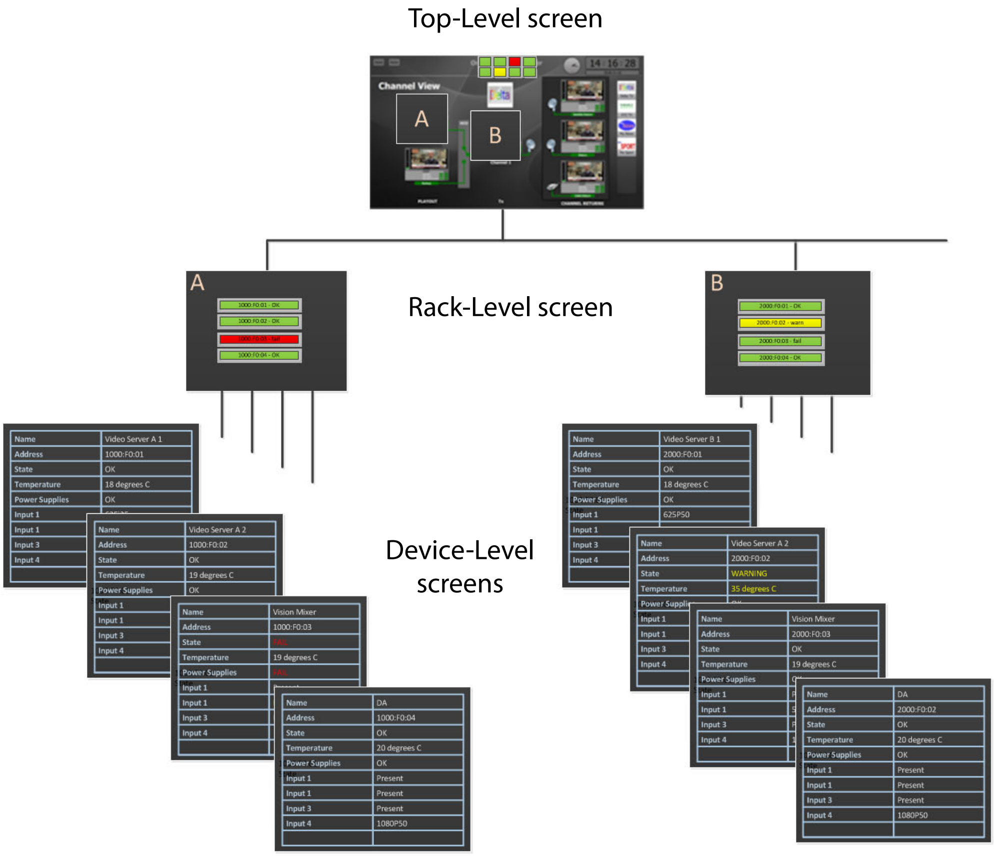


Fig. 1-3: Example Orbit MapView Screen Hierarchy

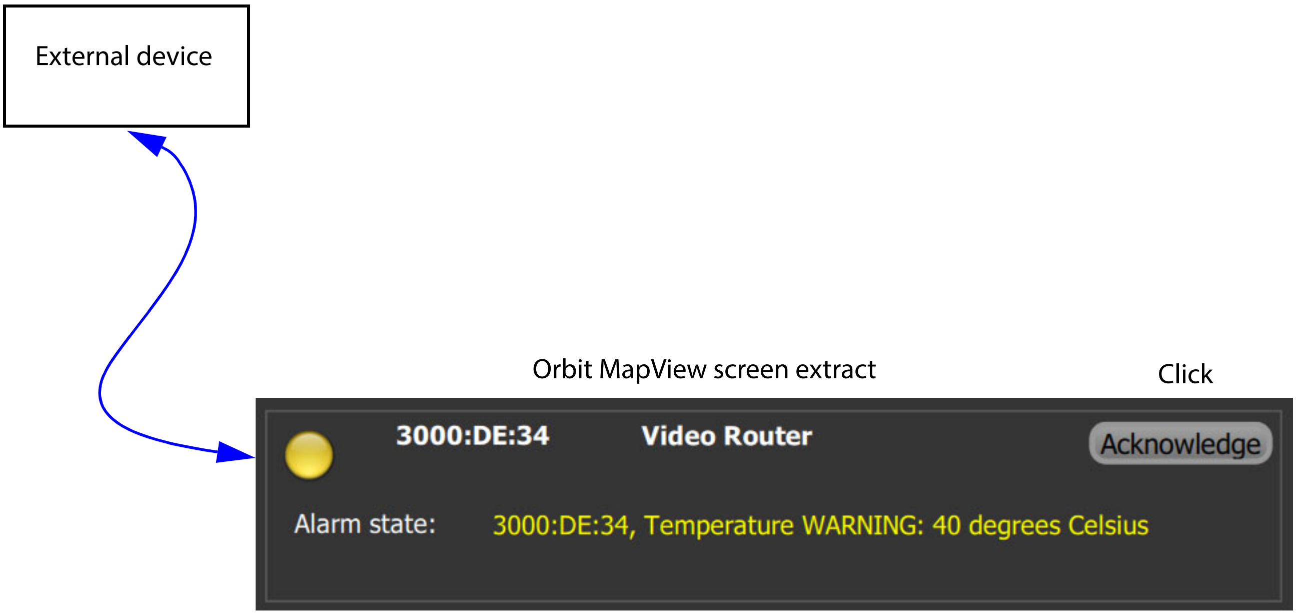
Widgets are graphical items on screens with properties that change the widget's appearance, for example, font size, flashing border, text label, color or transparency. The behavior of a widget is how its appearance changes according to various conditions; this is defined with Orbit 'Behaviours' and 'Bindings'.

Screen Logic

Screens contain graphical widgets such as video tiles, audio bars, tally lamps, status indicators, buttons, check boxes, slider bars, clocks, text labels etc. Control/indicator widgets typically interact with device status information and device control interfaces for their control/monitoring function.

A user may define widget interactions and behavior.
For example, a button click to acknowledge an alarm message and cancel an on-screen warning or error indicator.

The interaction logic (between the widget, external device(s) and any other widgets etc.) happens 'behind-the-scenes' and is defined via the screen schematic. (See Figure 1-4.) The required 'behind-the-scenes' logic can be defined in Orbit on a screen using interconnected Orbit 'Behaviours' and 'Bindings'.



3000:DE:34, Temperature WARNING: 40 degrees Celsius

Example 'Behind-the-scenes' interaction logic.
The status of some external device is monitored and,
if the external device indicates a warning or failure, then:

1. a lamp illuminates yellow (or red) on-screen, to indicate a warning (or failure); and following nay remedial action by the user,
2. a user clicks a button to acknowledge the alarm, changing the lamp color green.
3. The external device subsequently reports an OK status after any warning/failure condition is cleared.
4. The OK condition is reflected on-screen.

Fig. 1-4: Simple 'Behind-the-Scenes' Interaction Logic Example

A Behaviour

An Orbit 'Behaviour' gets, puts or holds data. It can get information from or pass information to the outside world or to a widget property via an Orbit 'Binding'.

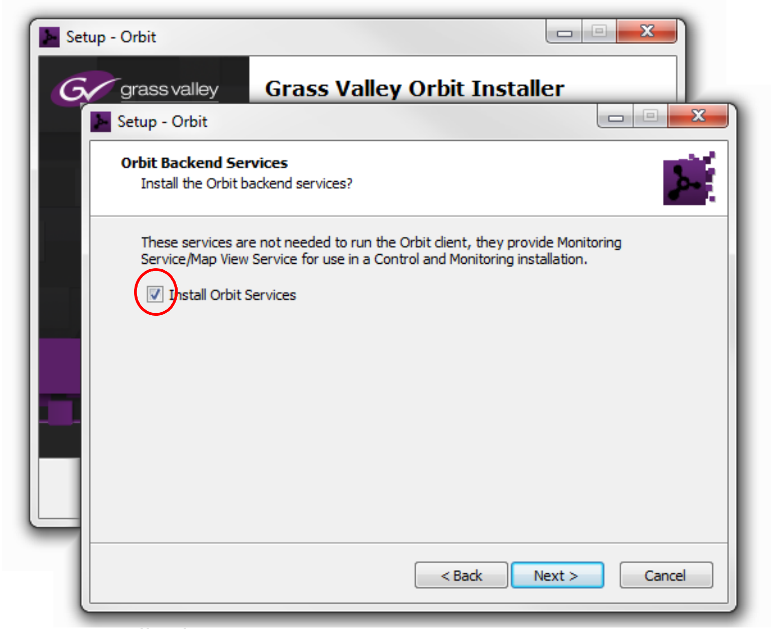
A Binding

A 'Binding' processes data. It connects a Behaviour to a widget or to another Behaviour. Typically, it binds an output of a Behaviour to a widget property.

Orbit MapView Behaviours and Bindings are listed in Appendix B List of Behaviours and Bindings, on page 355.

Orbit Services

Orbit Services run on a server computer as (Windows) services. They are installed on a server computer as an option when installing Orbit, see Figure 1-5.



Grass Valley Orbit Installer

Orbit Backend Services

Install the Orbit backend services?

These services are not needed to run the Orbit client, they provide Monitoring Service/Map View Service for use in a Control and Monitoring installation.

☑ Install Orbit Services

Fig. 1-5: Install Orbit Services

Orbit monitoring services installed include:

- Monitoring Service
- MapView Service.
- Routing Service.
- Email Service.
- Recording Service.

Orbit Monitoring Service

The primary purpose for the Orbit Monitoring Service (OMS) is to calculate key state information from Log Fields generated from log information from cards or units in a system. OMS uses Log Field data from a Grass Valley Log Server (a Windows service and part of the Grass Valley RollCall Middleware Services) and OMS may also process raw log data from cards/units (i.e. any device in its 'control and monitoring' RollCall domain). OMS then publishes the calculated state data in Alarm Messages to subscribers. Orbit MapView can subscribe to these Alarm Messages and Alarm state information can be used on a MapView screen with Alarm Behaviours.

Orbit MapView may also write log data to the 'control and monitoring' RollCall domain. This data will be processed by OMS.

MapView Service

The MapView service runs on a server and executes the same Orbit MapView project in parallel with the client.

The service evaluates the alarm state of *all* MapView screens and then publishes overall project 'state' information to the client. This enables any MapView screen Link icons to actively indicate the overall state of the screen they link to (for example, a Link icon may have a flashing red border to indicate an error state in the linked-to screen).

The Orbit MapView service handles any GSM alarms.

The Orbit MapView service also executes (server-side) logic contained in any special files within the project.

A GIT repository may be used to hold Orbit projects. This can ensure the Orbit MapView service uses the same Orbit MapView project as is running on an Orbit MapView client.

Routing Service

The Routing service interfaces between Orbit soft control panels and a Router Controller device using RollCall Multi-Matrix or SWP-P-08 protocols.

Densité Control Service

The Densité Control service allows Orbit to set/view multicast parameters of Densité cards. Note that a restart will be required for changes to take effect.

Email Service

The Email service sends emails on the behalf of Orbit via a configured SMTP server.

Recording Service

The Recording service captures and stores a copy of all log message or alarm changes within a system, allowing historical events to be viewed and investigated. The service listens to system messages. A main log file is used to record everything. Additionally, smaller, filtered log files can record a smaller subset of data within the system.

Source Control

Orbit projects may be stored in normal file folders or under source control, which is better. Reasons for using source control:

- maintain a backup of changes;
- ability to rollback to a known good working copy; and
- easier sharing of Orbit projects between multiple system engineers.

Git

Orbit can use any Git server to store and share Orbit projects, and this functionality is integrated directly into the Orbit client software via the ‘Push’ and ‘Pull’ tool bar buttons.

The Orbit MapView Service can be configured to automatically pull changes from Git. This avoids copying around different versions of the project.

Other Source Control Systems

Other source control systems, for example SVN, are not integrated with Orbit but may still be used to hold projects and versions of a project. In this case, exclude the project’s ‘.git’ folder from the source control system. (The ‘.git’ folder is found at the top level of the project and will be re-created each time Orbit opens/re-opens the project if it is missing.)

2 MapView Home Screen

Summary

MapView Home Screen

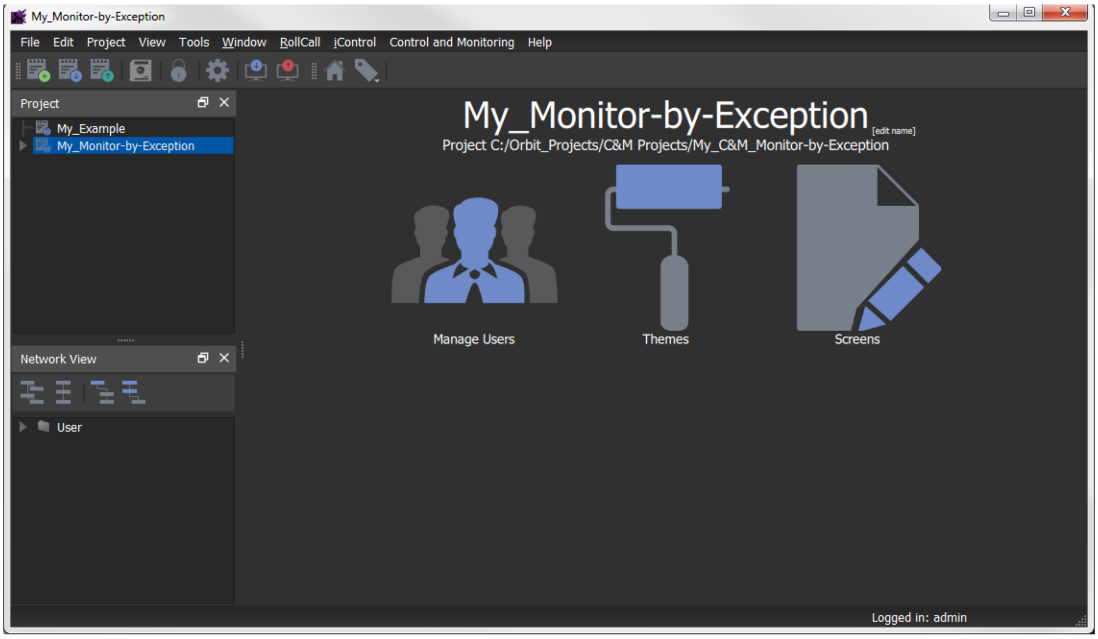

My_Monitor-by-Exception
File Edit Project View Tools Window RollCall iControl Control and Monitoring Help
Project
My_Example
My_Monitor-by-Exception
Network View
User
My_Monitor-by-Exception [edit name]
Project C:/Orbit_Projects/C&M Projects/My_C&M_Monitor-by-Exception
Manage Users
Themes
Screens
Logged in: admin

Fig. 2-1: Orbit MapView Project Home Screen

Initial Orbit Home Screen

Orbit MapView requires the Orbit Client software to be run with a MapView license installed on a license server. It uses an Orbit control-and-monitoring type project (a C&M project). The license is used when a project is run.

Orbit C&M Projects

To create a new project:

1 Click the 'New project' icon. See Figure 2-2.

2 Select 'C&M project'.

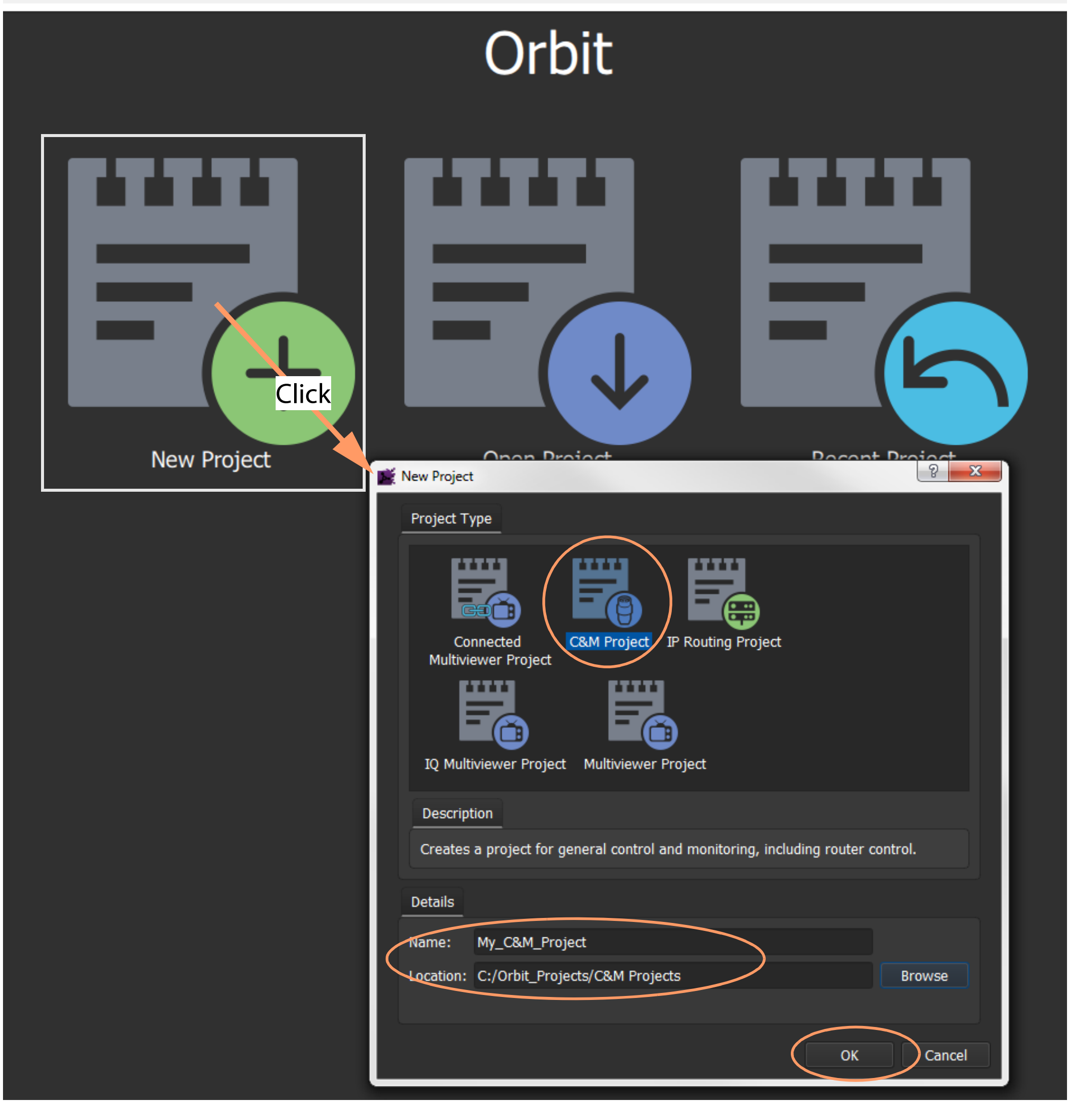
3 Enter a project 'Name'.

4 Browse to the folder where the new project is to be created. Click 'Choose'.

5 Click OK.

A new project is created in the folder.

Note: Creating new and opening existing Orbit projects is described in the 'Orbit Introduction' user manual.



Creates a project for general control and monitoring, including router control.

Name: My_C&M_Project

Location: C:/Orbit_Projects/C&M Projects

Fig. 2-2: Creating a New Orbit 'Control and Monitoring' Project

C&M Project Home Screen

The project home screen of an Orbit C&M project is shown in Figure 2-3.

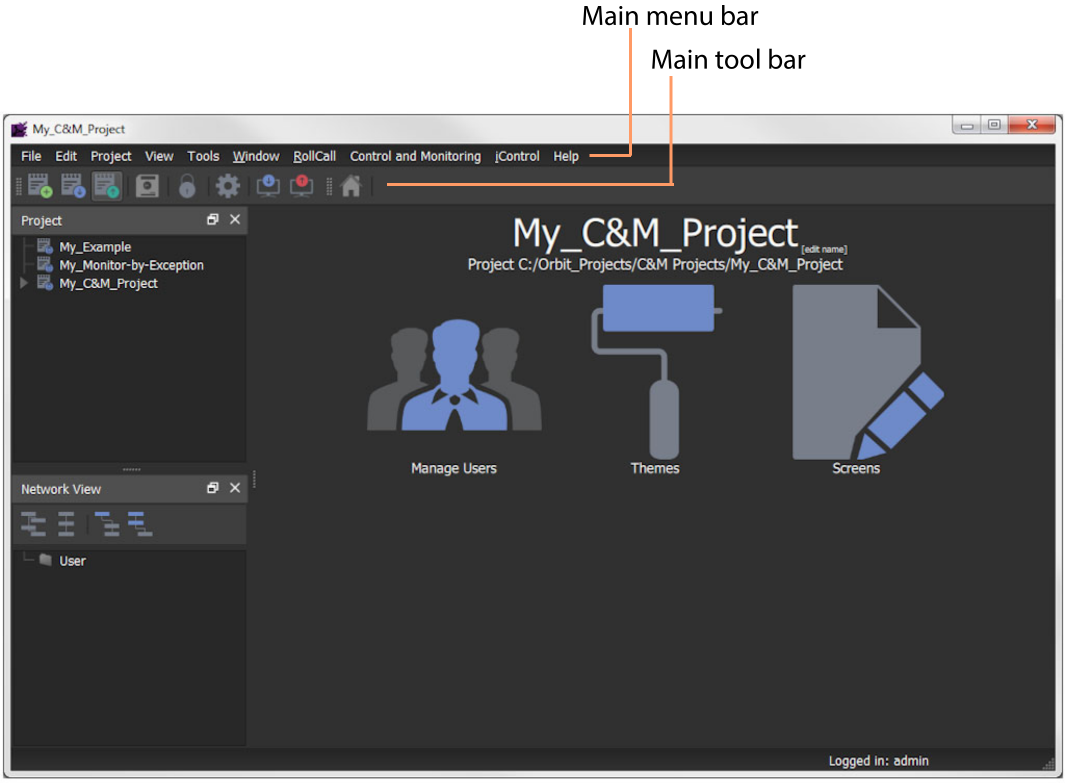

Main menu bar
Main tool bar
My_C&M_Project
File Edit Project View Tools Window RollCall Control and Monitoring iControl Help
Project
My_Example
My_Monitor-by-Exception
My_C&M_Project
Network View
User
My_C&M_Project [edit name]
Project C:/Orbit_Projects/C&M Projects/My_C&M_Project
Manage Users
Themes
Screens
Logged in: admin

Fig. 2-3: Orbit C&M Project Home Screen

The project home screen contains three large icons:

- **Manage Users** - Click to create new users and user roles with permissions.
  By default, a new project has one user set up:
  user name = 'admin'; and
  password = 'admin'.

  See Users, Roles and Permission, on page 35, and refer to the 'Orbit Introduction' user manual for more information.
- **Themes** - Click to show the **Theme Editor**.
  See Themes, on page 38.

> Note: Refer to the 'Orbit for Multiviewers' user manual for more information about on-screen Themes.

- **Screens** - Click to open a screen in the screen **Schematic Editor**.
  This document uses Orbit MapView screen in examples.

> Note: Refer to the 'Orbit for Multiviewers' user manual for basic instructions on using an Orbit schematic editor, widgets, and Behaviours and Bindings.

Main Menu Bar Items

Note: Refer to the 'Orbit for Multiviewers' user manual for full information about other main menu/tool bar items.

There are some specific Orbit MapView main menu bar items:

- **Project > Set as Home**- see Project > Set as Home, on page 13.
- **Tools > Options > Monitoring** - see Tools > Options > Monitoring - Masking tab, on page 14.
- **Tools > Options > MapView** - see Tools > Options > MapView, on page 16.
- **iControl** - see iControl Menu Bar Item, on page 18.
- **Control and Monitoring** - see Control and Monitoring > Properties (Main Menu Bar Item), on page 20.

See Figure 2-4.

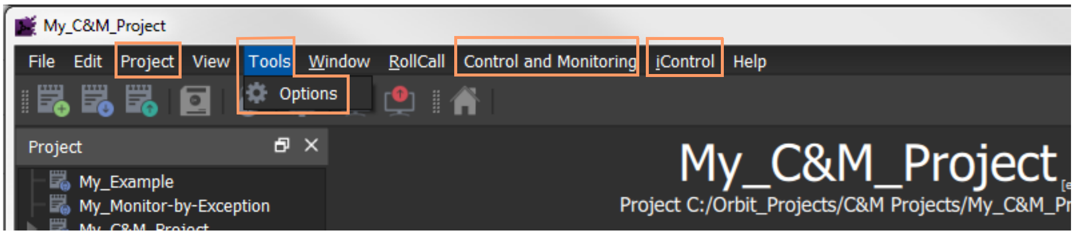

My_C&M_Project
File Edit Project View Tools Window RollCall Control and Monitoring iControl Help
Options
Project
My_Example
My_Monitor-by-Exception
My_C&M_Project
My_C&M_Project
Project C:/Orbit_Projects/C&M Projects/My_C&M_Pr

Fig. 2-4: Orbit C&M Project - Menu Bar

Project > Set as Home

With a top-level screen open in the schematic editor:

- Click 'Project > Set xxx as home' in the main menu.

This sets the top level schematic as the 'Home' screen.

Note: This is required for Orbit Services to correctly parse a project's hierarchical tree structure and evaluate a project's 'Link State'.

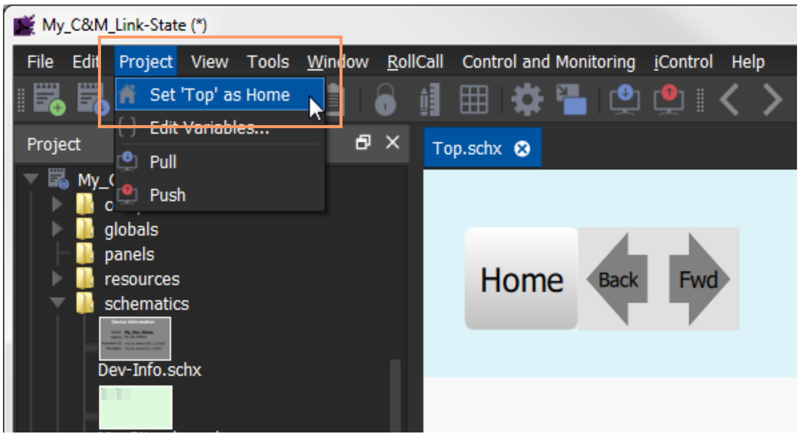
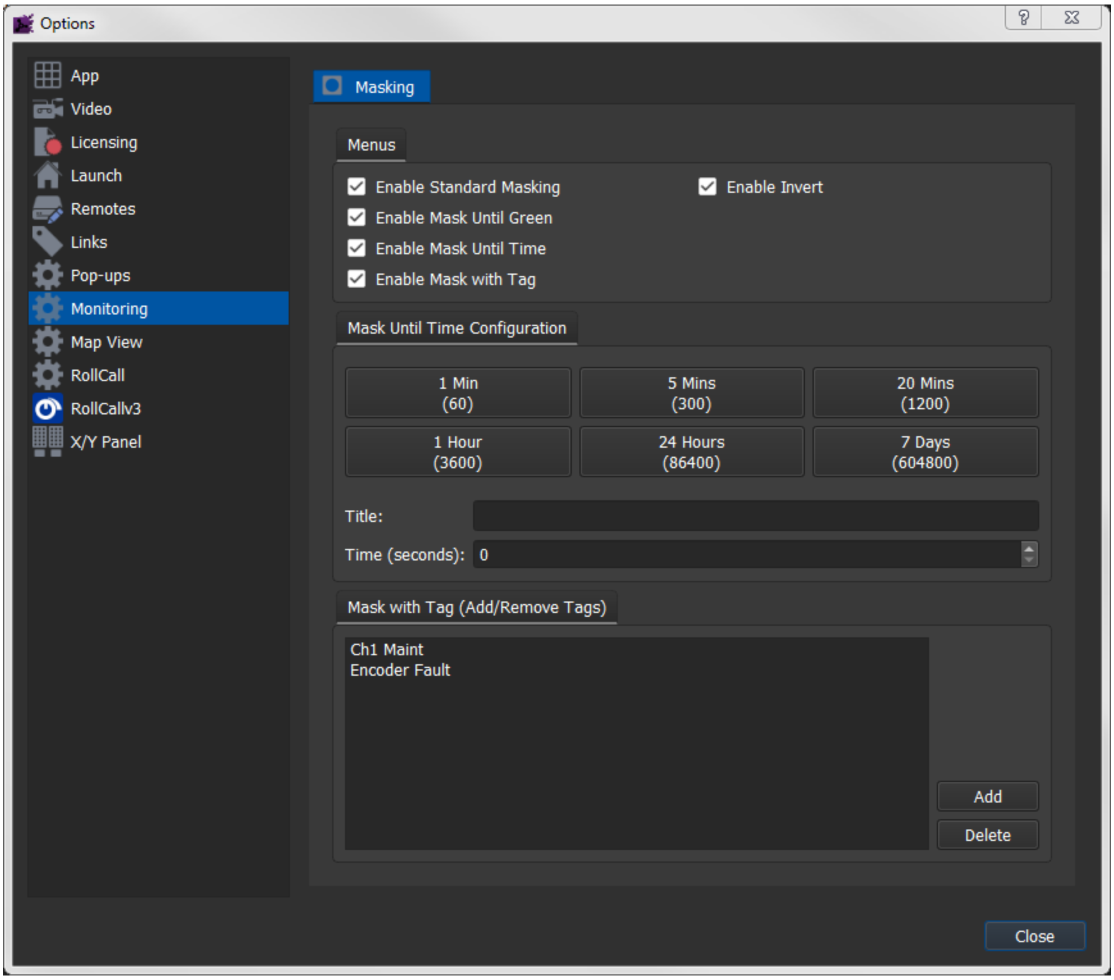


Fig. 2-5: Tools > Options > MapView

Tools > Options > Monitoring - Masking tab

The **Masking** tab of the 'Tools > Options > Monitoring' main menu contains various setting associated with masking alarms.



☑ Enable Standard Masking
☑ Enable Mask Until Green
☑ Enable Mask Until Time
☑ Enable Mask with Tag
☑ Enable Invert

Title:
Time (seconds): 0

Ch1 Maint
Encoder Fault

Fig. 2-6: Tools > Options > Monitoring - Masking Tab

Table 2-1: Tools > Options > Monitoring - Masking Items

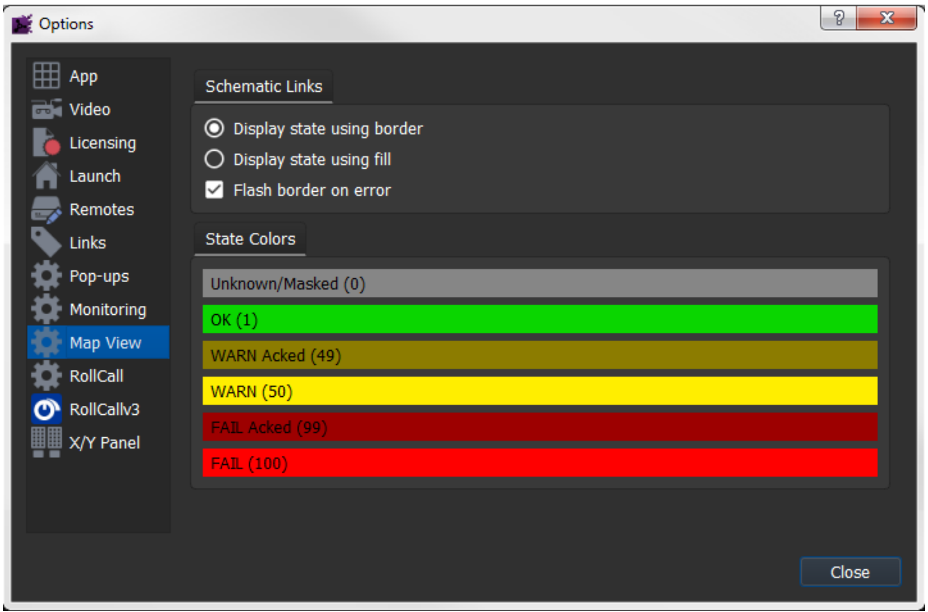
Item	Description
Menus	Check boxes. Enable various masking facilities in the Orbit MapView project.
Enable Standard Masking	
	Select to enable the standard masking mode for units and Log Field headers. Alarms from a unit/Log Field header will be suppressed in the system. Masking is applied to an alarm until it is removed by the user (unmasked).
Enable Mask Until Green	
	Select to enable the 'Mask Until Green' masking mode. Alarm/status value(s) from a selected unit will be suppressed (masked) until the alarm state is "Green", i.e. "OK". After this, the alarm is unmasked, i.e. an alarm condition will be raised by the alarm/status value(s).

Table 2-1: Tools > Options > Monitoring - Masking Items (continued)

Item	Description
Enable Mask Until Time	
	Select to enable the 'Mask Until Time' masking mode. Alarms from a selected unit will be suppressed (masked) for a period of time. After this, the alarm is unmasked, i.e. an alarm condition will be raised by the alarm/status value(s). There are six configurable 'Mask Until Time' rules which can be configured on buttons in a panel in the dialog. See the **Mask Until Time Configuration** table item below.
Enable Mask With Tag	
	Select to enable the 'Mask With Tag' masking mode. Mask tags can be defined. (See the Mask With Tag table item below.) This enables standard masking to be done but in a tagged way, enabling a unit to be masked for more than one reason. See Tagged Masking, on page 29.
Enable Invert	
	Select to enable 'Invert' masking mode, where all 'Warnings' and 'Failures' are set to show an 'OK' status. Applying this 'invert' can be done via an **Alarm Mask** Behaviour.
Mask Until Time Configuration	
	Six 'Mask Until Time' rule buttons via a panel. The rules are set by default to durations ranging from 1 minute to 7 days and are user configurable:
1 Min (60) **5 Mins (300)** **20 Mins (1200)** **1 Hour (3600)** **24 Hours (86400)** **7 Days (604800)**	To configure a rule: • Click on a button in the panel. • Enter a **Title** (in the text box below it). • Enter a **Time (seconds)** duration.
Title	Text box. Enter a name for the selected 'Mask Until Time' rule button.
Time (seconds)	Time value (seconds). Adjust this time value required for the selected 'Mask Until Time' rule - use the up/down arrow controls.
Mask With Tag (Add/Remove Tags)	
	A list of mask tags. See Tagged Masking, on page 29. To add a tag item: • Click **Add**. • Edit the tag name. To delete an item: • Select an item. • Click **Delete**.

Tools > Options > MapView

Button widgets in Orbit MapView screens may be used to link to other screens with Orbit **Link** Behaviours. The buttons can show the overall 'state' of the lower-level, 'linked-to' screen. The set of overall states for a project is called the **Link State**. This 'Tools > Options> MapView' configuration item provides **Link State** display options for such **Button** widgets.



Schematic Links

- ● Display state using border
- ○ Display state using fill
- ☑ Flash border on error

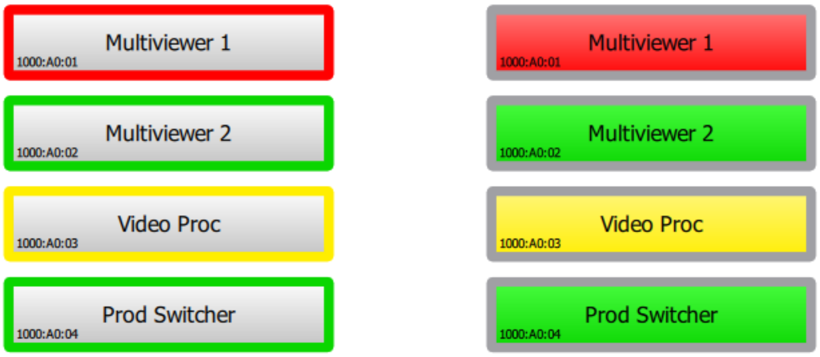
State Colors

- Unknown/Masked (0)
- OK (1)
- WARN Acked (49)
- WARN (50)
- FAIL Acked (99)
- FAIL (100)

Fig. 2-7: Tools > Options > MapView

Table 2-2: Tools > Options > MapView Items

Item	Description
Schematic Links	
Radio buttons:	
Display State Using Border	
	Select to show 'state' via a colored border. See Figure 2-8a.
Display State Using Fill	
	Select to show 'state' with a colored fill. See Figure 2-8b.
Flash Border on Error	
	Check box. Select to flash the colored border in case of a warning or an error state. This is done when **Display State Using Border** is selected.
State Colors	List of colors to be used to show the state. Click on a horizontal colored bar to edit the colors used.



a) Border Color

b) Fill Color

Fig. 2-8: State Shown: a) Border Color; b) Fill Color

State Value 0 to 100

In Orbit, state is represented with an integer value from 0 to 100, inclusive, described in Figure 2-3

Table 2-3: State Values

State Value	State Description	Color Representation
0	Unknown state or a masked state	Gray
1	OK	Green
2 to 48	Not used	
49	Warning, acknowledged	Dark Yellow
50	Warning	Yellow
51 to 98	Not used	
99	Fail, acknowledged	**Dark Red**
100	Fail	**Red**

iControl Menu Bar Item

Orbit can be configured to connect directly to a Grass Valley iControl server (General Services Manager, GSM). Orbit can obtain alarms (virtual or real) which can be used on the screen like other alarms, as required.

To enable GSM alarms to be seen in the **Network View**, configure Orbit with one or more iControl servers. In the Orbit main menu bar.

- Click **iControl > Configuration** in the main menu bar. (See Figure 2-9.)

  The 'iControl/Densité Configuration' dialog is shown, see Figure 2-10.

In this dialog, the IP addresses of multiple iControl Servers can be entered. The iControl servers can optionally provide the following functionality to Orbit: a Lookup service, a GSM REST API, and a Densité REST API. These can be enabled/disabled independently for each iControl Server.

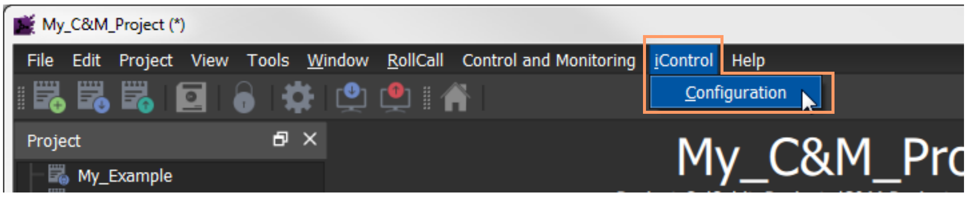
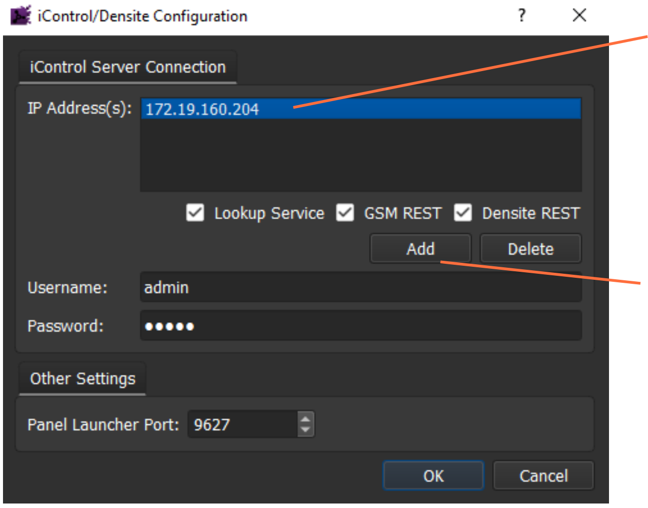


Fig. 2-9: iControl > Configuration



IP Address(s): 172.19.160.204

☑ Lookup Service ☑ GSM REST ☑ Densite REST

Username: admin

Password: •••••

Panel Launcher Port: 9627

First IP address listed will be used for GSM alarms.

Other IP addresses can be used for Densité connections.

Click Add to add an iControl Server IP address

Fig. 2-10: iControl/Densité Configuration Dialog

Table 2-4: iControl/Densité Menu Bar Items

Item	Description
iControl Server Connection:	
IP Address(s)	Add iControl Server IP address(es). **Note**: First IP address listed will be used for GSM alarms. Other IP addresses can be used for Densité connections.
Lookup Service	Check box. This is automatically selected for each added IP address. This is a Query service to enable discovery of other iControl services. Deselect service if required.
GSM REST	Check box. This is automatically selected for each added IP address. This provides a RESTful application programing interface (API) to a GSM alarm service. Deselect service if required.
Densité REST	Check box. This is automatically selected for each added IP address. This RESTful API to an iControl server. Deselect service if required.
Add	Button. Click to add an IP address to the list.
Delete	Button. Click to delete the selected IP address in the list.
Username	Text box. Enter a username for the iControl server. **Note:** The Username and Password applies to all listed iControl servers.
Password	Text box. Enter a password for the iControl server. **Note:** The Username and Password applies to all listed iControl servers.
Other Settings:	
Panel Launcher Port	Text box. Enter the IP port number to use to communicate with the Densité Panel companion application. **Note:** The **Panel Launcher Port** applies to the Densité Panel application running alongside Orbit on the client PC. **Note:** The Densité Panel is used to view device control screens of iControl/Densité devices (see iControl/Densité Device Control Screen Panel, on page 26).

Control and Monitoring > Properties (Main Menu Bar Item)

A main menu bar item.

- Click **Control and Monitoring > Properties** in the main menu bar.

  A 'properties dialog is shown, see Figure 2-12.

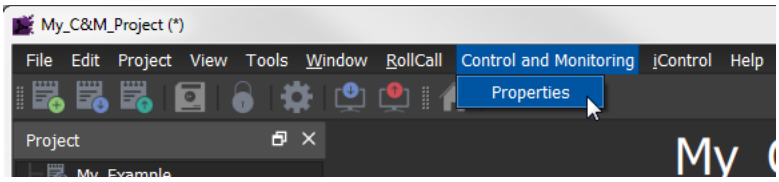
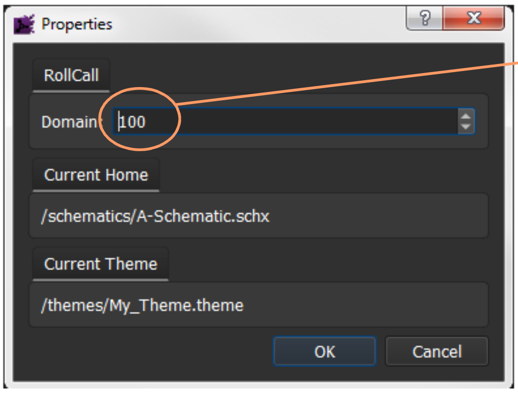


Fig. 2-11: Control and Monitoring > Properties



Note: The RollCall domain should match the client domain configured in the Orbit Services.
The Orbit client then sees device and service items in the same domain; and items appear in the **Network View** pane of the Orbit client application.

Fig. 2-12: Control and Monitoring > Properties Dialog

Table 2-5: Control and Monitoring > Properties Dialog Items

Item	Description
Domain	Enter the RollCall domain for the Orbit MapView client to use. See **Note 1**.
Current Home	Shows the screen currently set to be the home screen. To set a screen as the home screen: • open the screen schematic and click **Project > Set as Home** in the main menu bar.
Current Theme	Shows the currently set theme. To set a theme to be the current theme: • Right-click on the theme in the **Project View** and select **Set as Current Theme**.
Note 1:	The RollCall domain should match the client domain configured in the Orbit Services. Items will appear in the Network View pane of the Orbit client application.

Network View

The **Network View** pane in Orbit shows system devices in a hierarchical list which forms an expandable tree folder structure.

System devices can be various pieces of Grass Valley equipment.
For example:

- iControl General Service Manager (GSM) folders and alarms.
- Densité modules or frames.
- IQ modules or frame (via RollCall).
- IQ module frame and module logging data from a Log Server.
- Virtual alarms.
- Routing devices.

Figure 2-13 shows an example **Network View** pane. Devices and groups of devices in the system are shown and their live (non-latched) alarm state is shown (OK, Warning or Fail/Error).

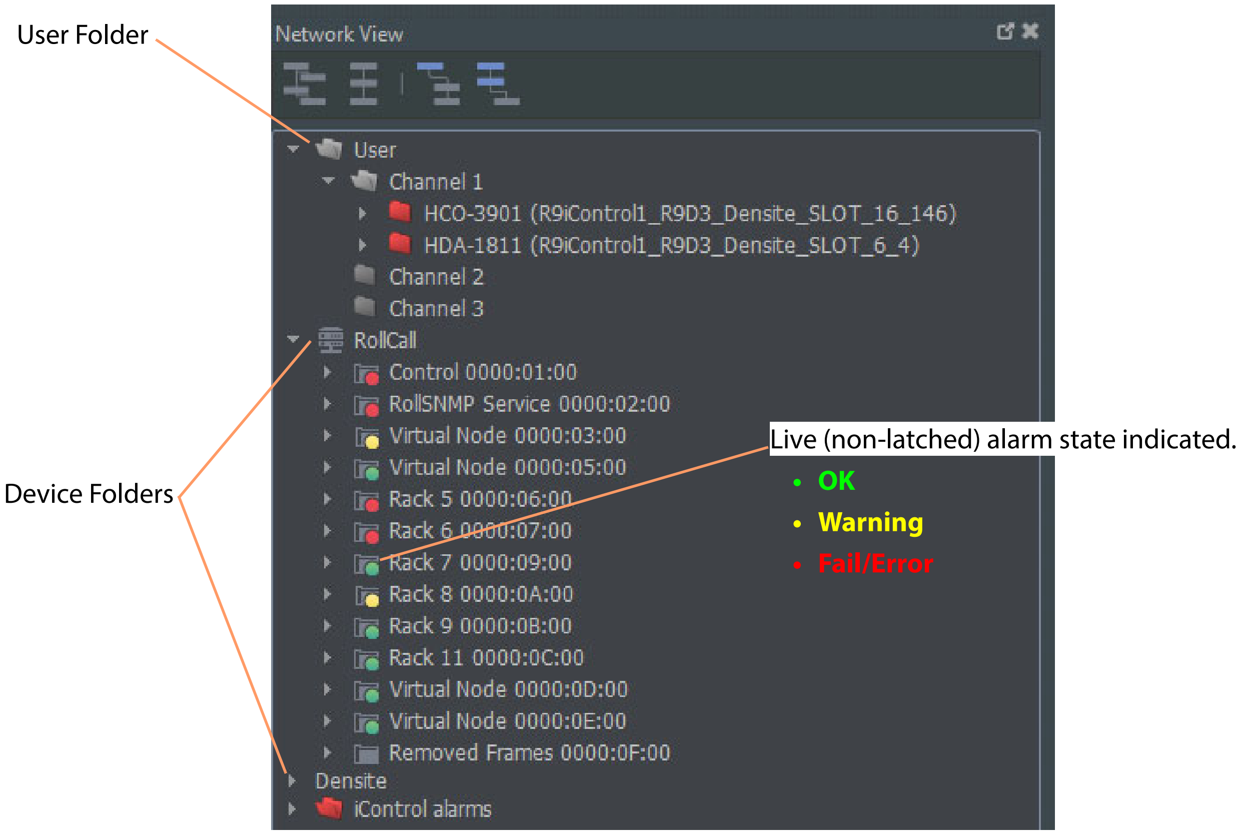


Fig. 2-13: Example Network View Pane

Folders

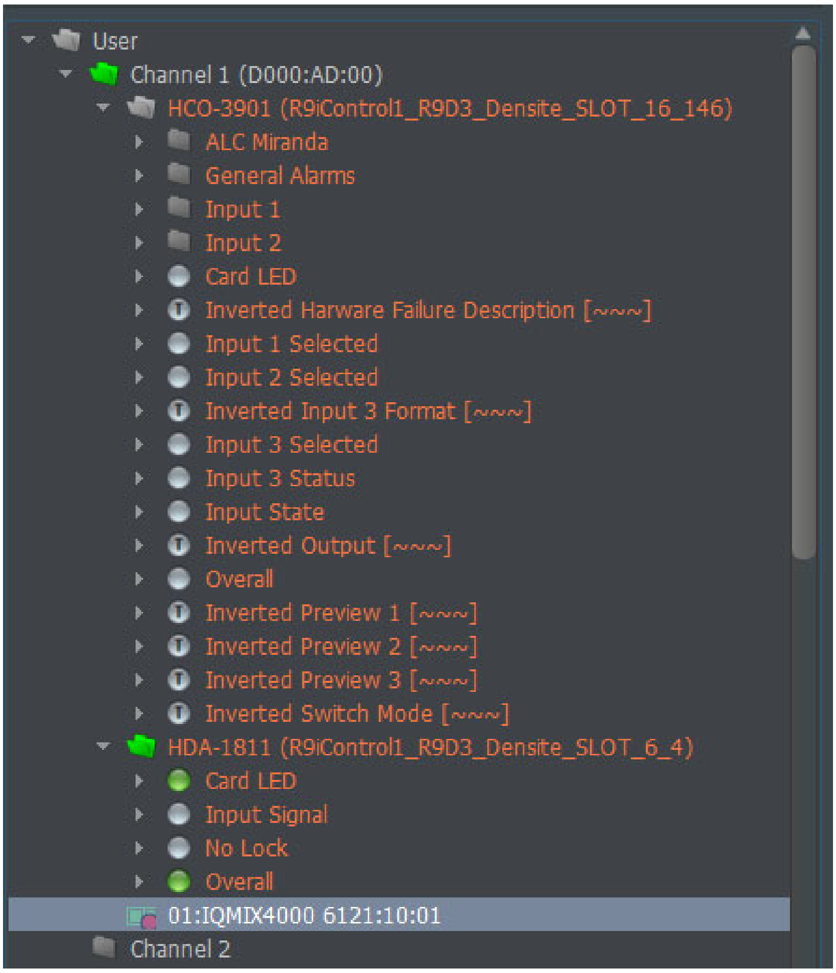
Items in the **Network View** are grouped under specific top-level folders (root elements):

- **RollCall** – Contains all RollCall/IQ devices and log data.
- **Densité** – Contains all devices from the iControl Densité manager.
- **iControl Alarms** – Contains all the alarm from iControl GSM manager.
- **User** – Allows custom groupings of devices to be created. Devices may be from any of the other top-level folders.

User Folder

User folders can form a list of frequently-accessed devices, or they may be used for monitoring purposes (by creating a virtual alarm for all devices in a User folder.

Folders can be created within the **User** folder and any devices and/or other user sub-folders can be added.



User
Channel 1 (D000:AD:00)
HCO-3901 (R9iControl1_R9D3_Densite_SLOT_16_146)
ALC Miranda
General Alarms
Input 1
Input 2
Card LED
Inverted Harware Failure Description [~~~]
Input 1 Selected
Input 2 Selected
Inverted Input 3 Format [~~~]
Input 3 Selected
Input 3 Status
Input State
Inverted Output [~~~]
Overall
Inverted Preview 1 [~~~]
Inverted Preview 2 [~~~]
Inverted Preview 3 [~~~]
Inverted Switch Mode [~~~]
HDA-1811 (R9iControl1_R9D3_Densite_SLOT_6_4)
Card LED
Input Signal
No Lock
Overall
01:IQMIX4000 6121:10:01
Channel 2

Fig. 2-14: Example User Folder View

To Add (Create) a Sub-folder

1 Right-click on the User folder (or on any sub-folder) and select 'Create Folder'.

2 Enter a folder name and click OK.

To Delete a Sub-folder

1 Right-click on the sub-folder item and select 'Delete Folder'.

2 Confirm by clicking 'Yes'.

To Rename a Sub-folder

1 Right-click on the sub-folder item and select 'Rename Folder'.

2 Enter a new name and click OK.

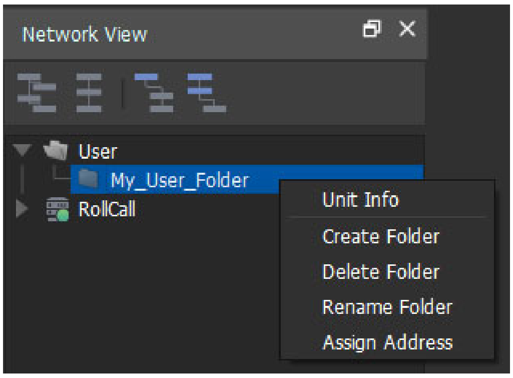


Fig. 2-15: Right-click Context menu

To Assign a RollCall Address to an Item

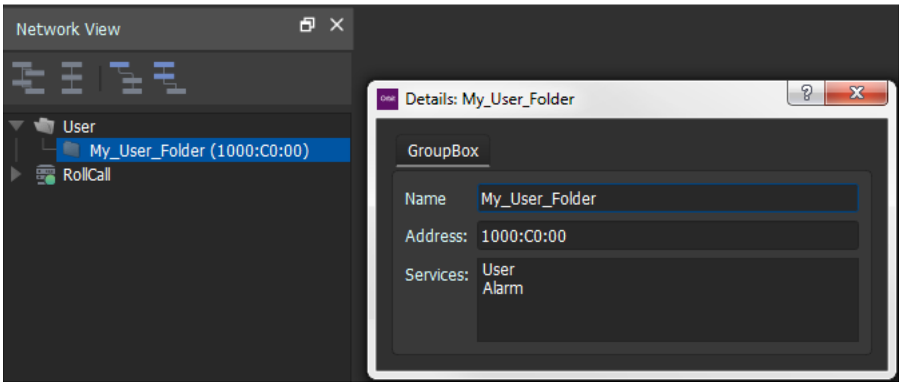
1 Right-click on the sub-folder item and select ‘Assign Address’.

2 Enter a RollCall address and click OK.
The RollCall address is associated with the sub-folder item (as shown in Figure 2-16).

To See Information about an Item

1 Right-click on the sub-folder item and select ‘Unit Info’.

2 Enter a new name and click OK.
A Details window is shown. See Figure 2-16.



Name My_User_Folder
Address: 1000:C0:00
Services: User
Alarm

Fig. 2-16: Item Information in Details Window

To Add Devices to a Folder/Sub-folder

1 Drag and drop units from one of the other folders into the user folder/sub-folder.

When devices have been added, they are fully functional with their normal **Network View** folder item behavior: Normal right-click context menu functions are available, i.e. the folder structure can be expanded/collapsed; the Densité parameters editor can be opened; and the IQ control screen editor can be opened.

To Remove an Item

1 Right-click on an item and select ‘Remove Item’ to remove the device/sub-folder.

User Folder RollCall Address and iControl/Densité Devices

Note: iControl/Densité devices are discoverable in the **Network View** pane’s Densité folder when iControl/Densité configuration has been set up (see iControl Menu Bar Item, on page 18).

A **User Folder** can be used to provide one or more iControl/Densité devices with a RollCall address. This is useful in Orbit.

Some Orbit Services (Monitoring, Recording) just handle RollCall addresses; and, in these cases, an Orbit MapView project can be used to provide devices with a RollCall address.

1. Open a MapView project.
2. Configure the iControl Menu Bar Item, on page 18, to connect to an iControl server.
3. Create a new **User Folder** in the **Network View** pane.
4. Assign a RollCall address to the folder.
5. Drag iControl/Densité devices into the **User Folder**.
6. Save project to a location when the Orbit services have access to the project. (Shared drive or Git repository - see the Orbit Services user manual.)

The iControl/Densité devices are now accessible to the MapView project via the RollCall address.

Accessing Device Control Screens

The control screens of discovered devices in the **Network View** pane are accessible by:

1 expanding the list;

2 right-clicking on a device item; and

3 selecting 'Open Control Screen' in the context pop-up menu.

The control screen is shown for the selected device:

- **RollCall-enabled** device - the control screen is opened within the Orbit Client tool. (See RollCall-Enabled Device Control Screen, on page 25.)
- **iControl/Densité** device - the control screen is shown in a launched Densité device panel. (See iControl/Densité Device Control Screen Panel, on page 26.)

RollCall-Enabled Device Control Screen

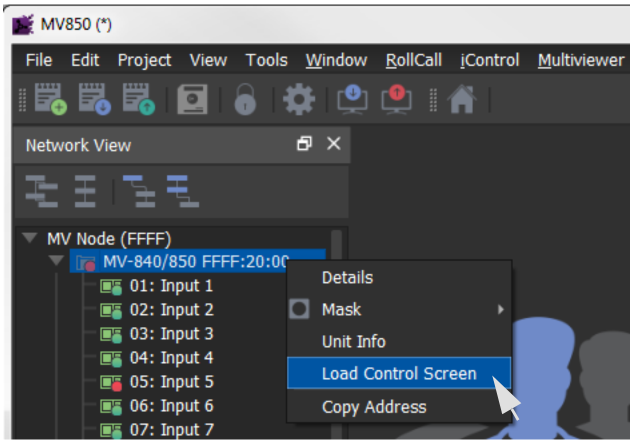
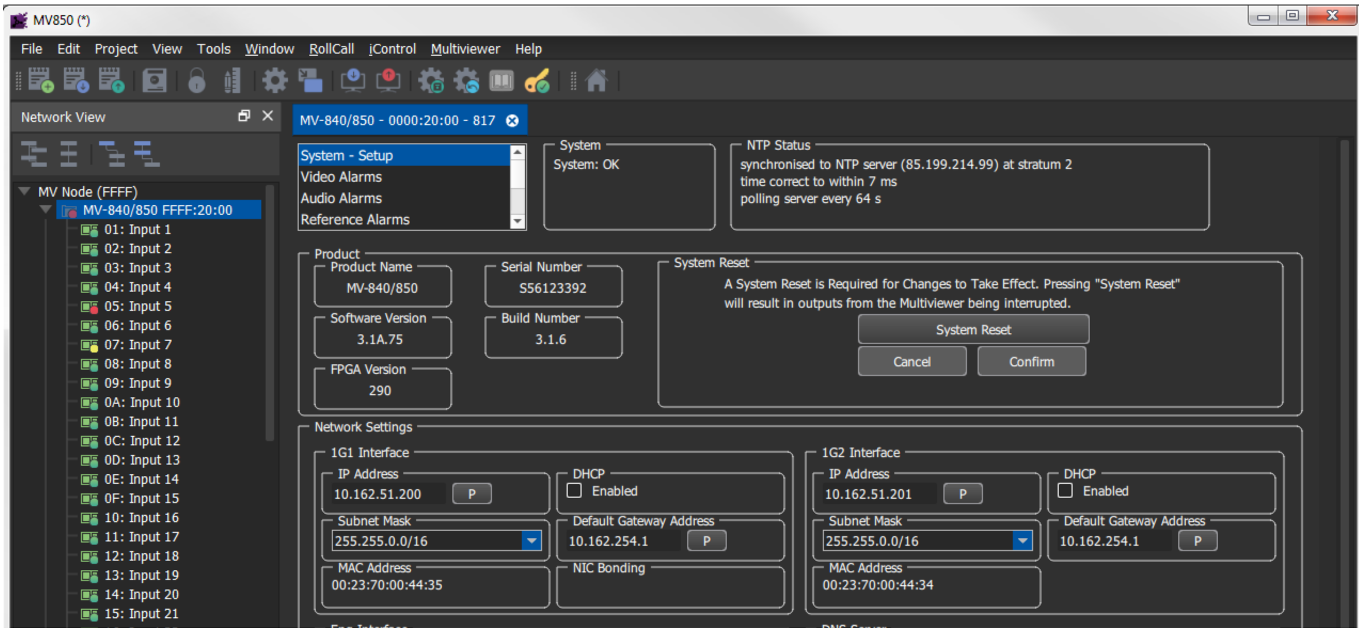


Fig. 2-17: Right-Click on Network View Device Item



System
System: OK

NTP Status
synchronised to NTP server (85.199.214.99) at stratum 2
time correct to within 7 ms
polling server every 64 s

Product
Product Name: MV-840/850
Serial Number: S56123392
Software Version: 3.1A.75
Build Number: 3.1.6
FPGA Version: 290

System Reset
A System Reset is Required for Changes to Take Effect. Pressing "System Reset" will result in outputs from the Multiviewer being interrupted.

Network Settings

1G1 Interface
IP Address: 10.162.51.200
DHCP: □ Enabled
Subnet Mask: 255.255.0.0/16
Default Gateway Address: 10.162.254.1
MAC Address: 00:23:70:00:44:35
NIC Bonding

1G2 Interface
IP Address: 10.162.51.201
DHCP: □ Enabled
Subnet Mask: 255.255.0.0/16
Default Gateway Address: 10.162.254.1
MAC Address: 00:23:70:00:44:34

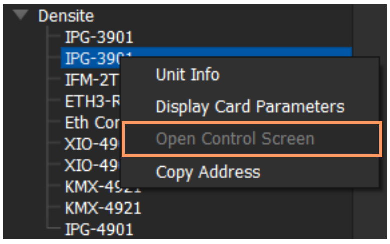
Fig. 2-18: Device Control Screen Opened in Orbit

iControl/Densité Device Control Screen Panel

Control screens for an iControl/Densité device are opened directly from Orbit and are shown in a launched Densité Panel application, a companion application to Orbit.

The Densité Panel application is installed automatically when installing Orbit 3v1 onwards. There is a prerequisite that the Java runtime environment is already installed on the Orbit client computer (Java SE Runtime Environment 8u121, 32bit).

Note: If Java is not available on the Orbit client computer, the 'Open Control Screen' drop-down menu option is grayed-out in the **Network View** pane.



Also, if the Densité Panel is not yet fully-initialized, then the option will be grayed-out. If this is the case, then wait and retry.

Orbit Configuration

In the Orbit client application, the location of the iControl Servers are configured through the Orbit menu iControl->Configuration, see iControl Menu Bar Item, on page 18.

The Densité Panel requires a connection to the iControl server's Lookup Service. The iControl server's GSM RESTful API and Densité RESTful API are required for Orbit to display Densité devices in the **Network View** and the show the Alarm State of those devices.

Usage

Once the iControl server connection is configured in Orbit, the **Network View** pane displays a tree view of the iControl/Densité devices.

Note: If the **Network View** pane is not visible, then click 'View > Windows' in the Orbit main menu and select 'Network Window' to enable the **Network View**.

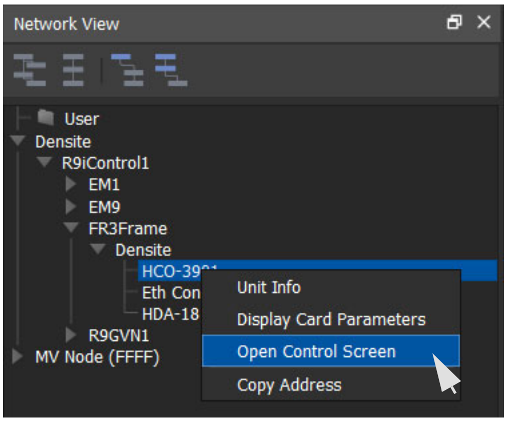
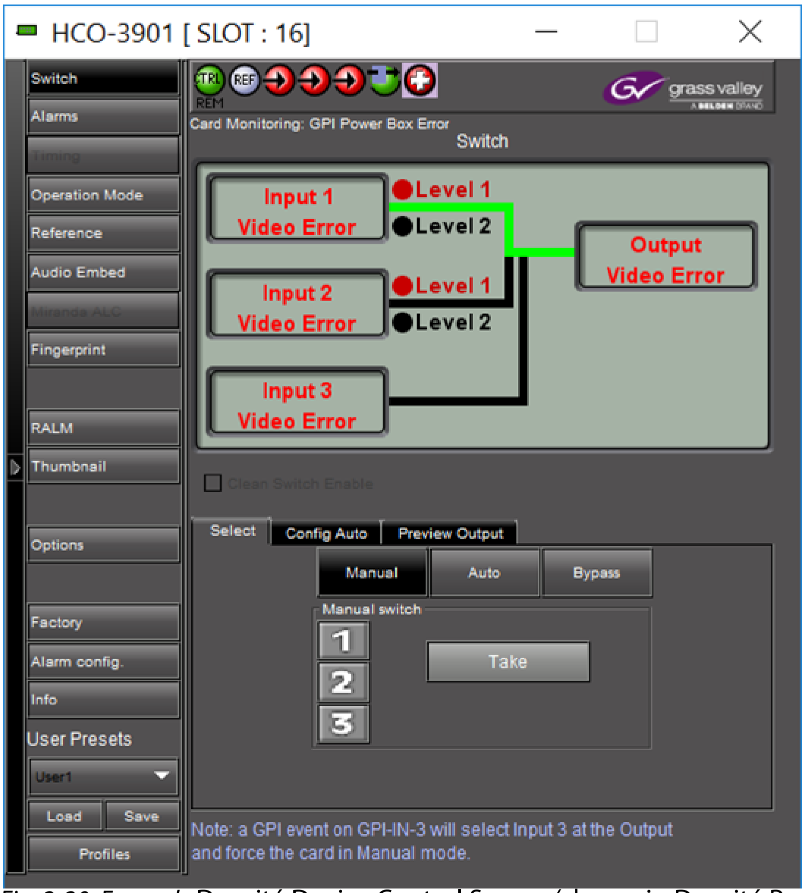


Fig. 2-19: Right-Click Drop-Down Menu

1 Browse to the device of interest and right-click on it. Select 'Open Control Screen'.

The device’s control screen is shown in the Densité Panel which is opened a separate window to Orbit.



Card Monitoring: GPI Power Box Error

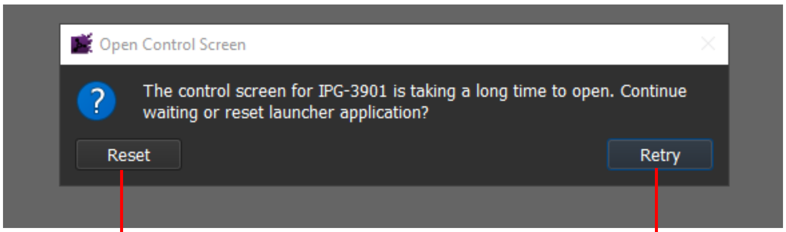
Note: a GPI event on GPI-IN-3 will select Input 3 at the Output and force the card in Manual mode.

Fig. 2-20: Example Densité Device Control Screen (shown in Densité Panel window)

Additional Information

Orbit maintains a ‘heartbeat’ signal with the Densité Panel which ensures that if Orbit is closed before the Densité Panel, then the panel will also close.

If in some cases the Densité Panel takes an extended length of time to open and show a device’s control screen, Orbit will present an Open Control Screen dialog to the user. The user may then cancel the operation or retry.



The control screen for IPG-3901 is taking a long time to open. Continue waiting or reset launcher application?

Click **Retry** to attempt to open the device’s control screen again

Click **Reset** to cancel

Fig. 2-21: Open Control Screen Dialog (shown if Densité Panel takes a long time to open)

Device Details

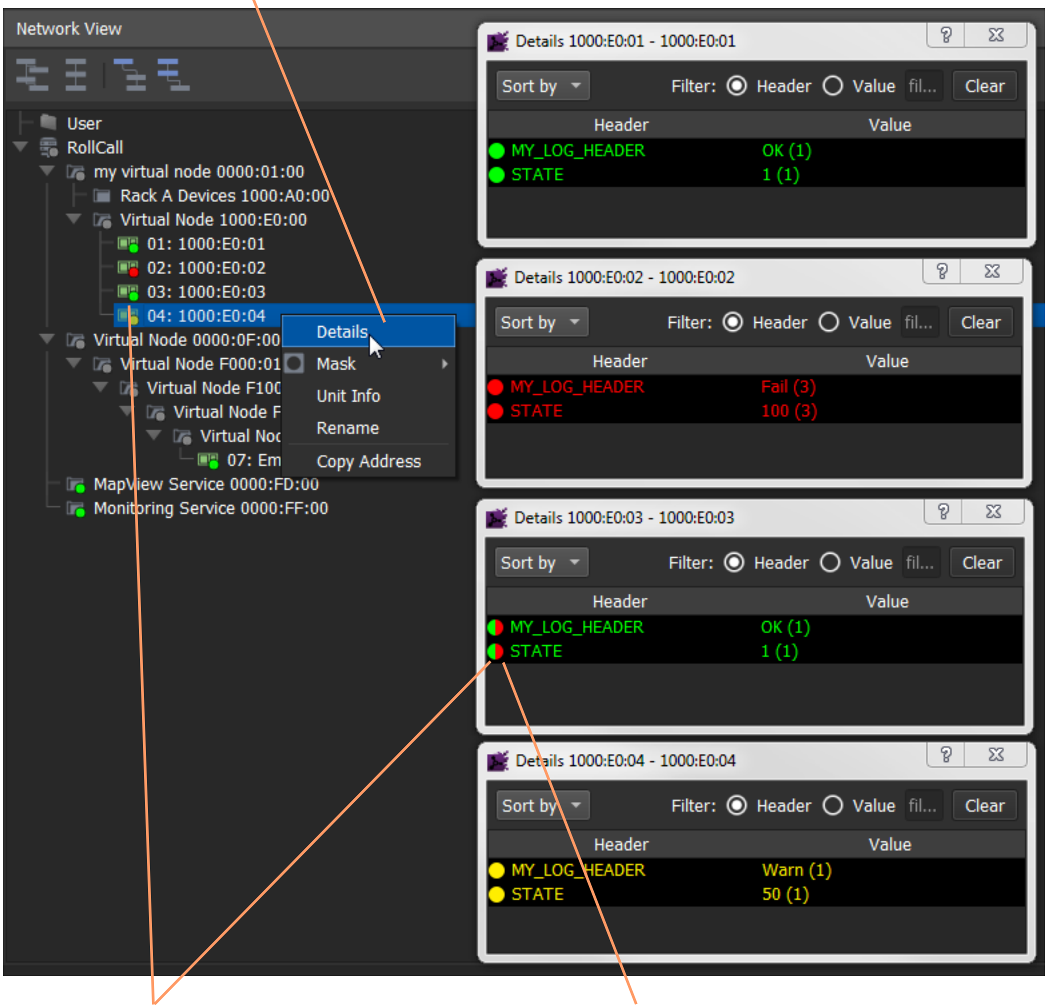
To show a device's 'Details' window from the **Network View** pane:

- Right-click on a device in the **Network View** and select 'Details'.

  See Figure 2-22.

Right-click, select 'Details'

'Details' windows:



Details 1000:E0:01 - 1000:E0:01

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● MY_LOG_HEADER	OK (1)
● STATE	1 (1)

Details 1000:E0:02 - 1000:E0:02

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● MY_LOG_HEADER	Fail (3)
● STATE	100 (3)

Details 1000:E0:03 - 1000:E0:03

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● MY_LOG_HEADER	OK (1)
● STATE	1 (1)

Details 1000:E0:04 - 1000:E0:04

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● MY_LOG_HEADER	Warn (1)
● STATE	50 (1)

Live (non-latched) alarm state indicated.

Latched alarm state indicated.

Fig. 2-22: Details Windows

The live (non-latched) Alarm state is indicated in the **Network View**.

Both the live and the latched Alarm state are indicated in the **Details** window.

Alarm Masking

Masking affects the overall displayed state. If a device's alarm is masked then its state value is 0.

- Right-click on an item in the **Network View** and choose the masking option from the context menu.

Tagged Masking

Alarms from system devices may be masked. Tagged masking is useful where there may be more than one reason for masking a device/unit. For example, a unit may need to be masked for operational maintenance reason and/or for some fault condition reason. Tagged masking enables two or more maskings to be defined and used.

Define tags in the main menu, in the 'Tools > Operation > Monitoring' Masking tab.

If one or more **Tagged Mask** is applied to a device, then the device alarms are masked.

All **Tagged Masks** must be removed before a device's alarms are unmasked.

Masking can be applied/removed via the **Network View** or via an **Alarm Mask** Behaviour.

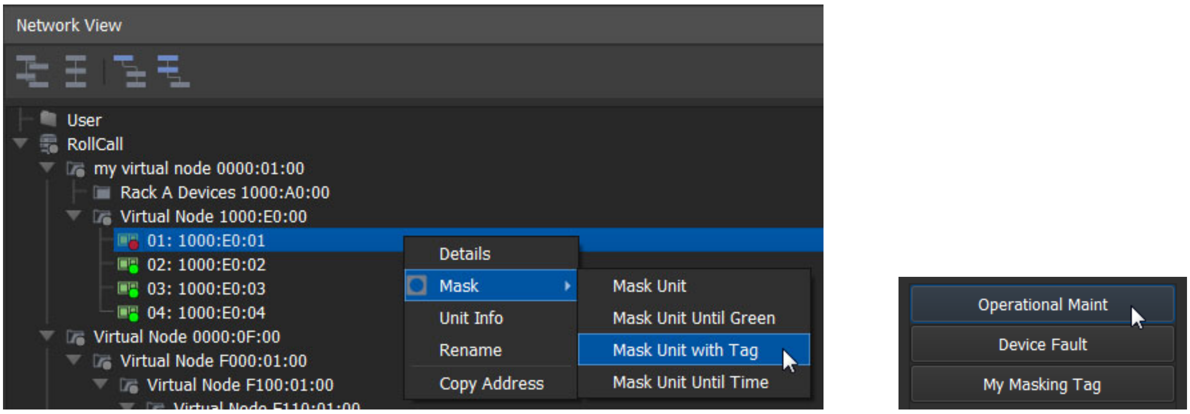
To Mask a Device with a Tagged Mask via Network View

Two **Tagged Masks** are defined in the 'Masking' tab for this example:

- 'Operational Maint'
- 'Device Fault'

To apply a **Tagged Mask**, in the **Network View**:

1 Right-click on a device item and select 'Masking with Tag'. (Figure 2-23a)



a) Right-click, select 'Mask > Mask Unit with Tag'

b) Select a Tag Mask

Fig. 2-23: Tag Mask

2 Select the **Tagged Mask** to apply. (Figure 2-23b)

A mask is applied to the device and this is indicated in the **Network View**. See Figure 2-24.

Grayed-out mask icon indicates alarms are masked

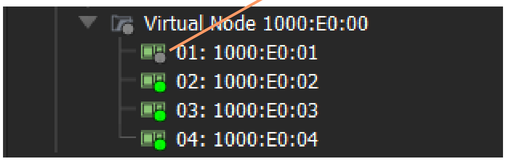


Fig. 2-24: Grayed-out Mask Icon

A further **Tagged Mask** maybe applied:

3 Right-click, select 'Masking with Tag' and then select another **Tagged Mask**. See Figure 2-25.

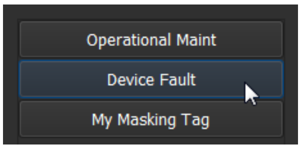


Fig. 2-25: Apply Another Tag Mask

Now there are two **Tagged Masks** applied to the device: 'Operational Maint' and 'Device Fault'. The device/unit is shown as being masked in the **Network View**.

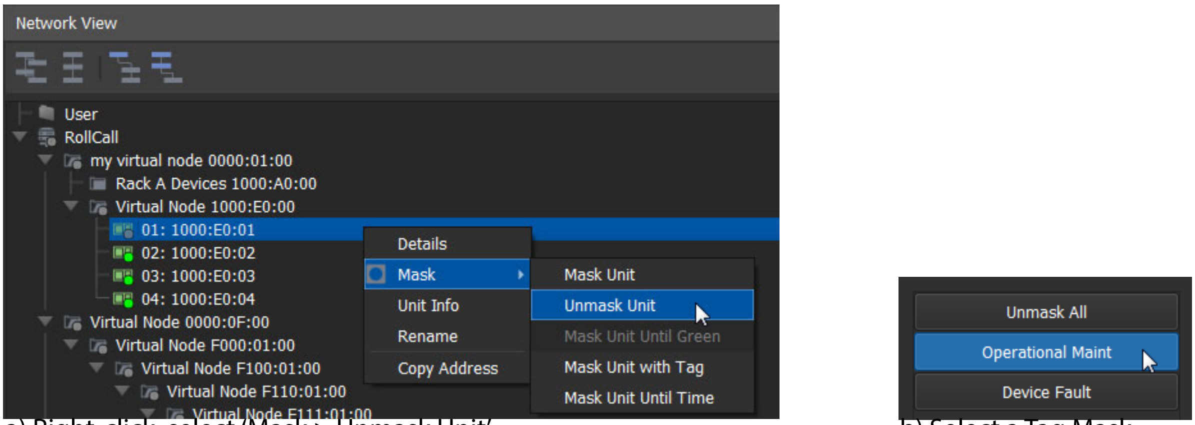
To Unmask a Device via Network View

To remove a **Tagged Mask**, in the **Network View**:

1 Right-click on a device item and select 'Mask > Unmask Unit'. (Figure 2-26a)

If two or more **Tagged Masks** are applied to the device,
then these are presented on screen.
Select the **Tagged Mask** to unmask. (See Figure 2-26b.)

Note: The device will have one **Tagged Mask** removed, but alarms will still be masked by any other remaining **Tagged Mask** operating on the device/unit.



a) Right-click, select 'Mask > Unmask Unit'

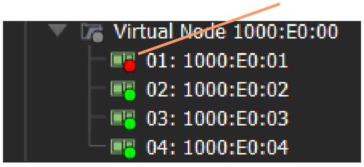
b) Select a Tag Mask

Fig. 2-26: Tag Mask

2 To remove the other **Tagged Mask**, right-click on a device item and select 'Mask > Unmask Unit'.

Note: If this is the only remaining **Tagged Mask** for the device/unit, then the unmasking is done immediately; otherwise, a list of **Tagged Masks** are presented, similar to Figure 2-26b.

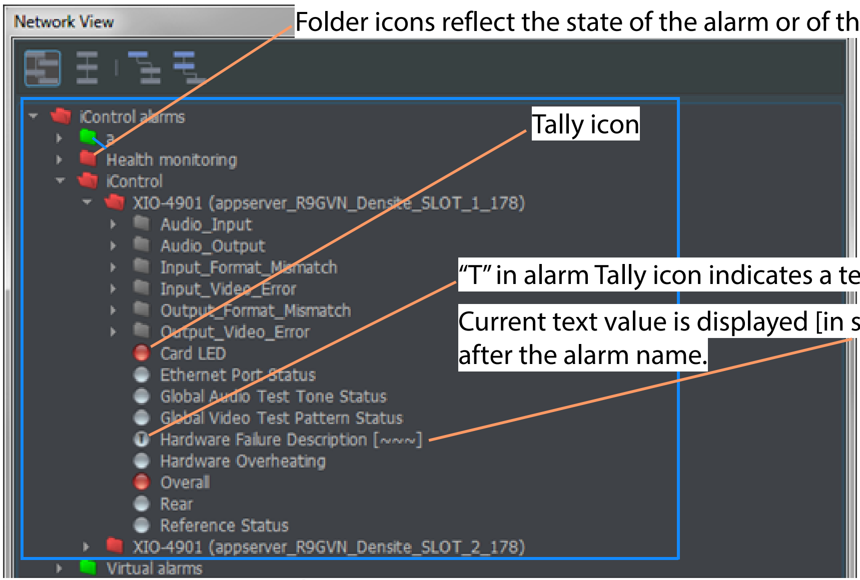
When a device is fully unmasked, its alarm state is re-shown in the **Network View**. See Figure 2-27.



Alarm state is unmasked and shown in icon in **Network View**

Fig. 2-27: Network View Shows Device Alarm State

iControl Alarms in Network View



Folder icons reflect the state of the alarm or of the folder.

Tally icon

“T” in alarm Tally icon indicates a textual alarm.

Current text value is displayed [in square brackets] after the alarm name.

Fig. 2-28: iControl and Densité Alarms in Network View

iControl Alarm States and State Value 0 to 100

The states for folders and alarms are taken *directly* from the GSM manager; no aggregate state calculation is performed by Orbit.

The iControl alarm state is mapped from the native GSM alarm state into the Orbit state scheme (see *State Value 0 to 100*, on page 17*)*:

- **0** – Unknown / Masked
- **1** – OK
- **49** - Warning Acknowledged
- **50** - Warning
- **74** – Major Failure Acknowledged
- **75** – Major Failure
- **99** – Critical Failure Acknowledged
- **100** – Critical Failure

This casts the alarm state values consistent with other, non-iControl devices.

Orbit also maps the “acknowledged” states from GSM into overall state values to provide one overall state value. This permits easier configuration of property Bindings because there is only one state to deal with.

iControl Alarm Acknowledging

Acknowledge an alarm using the Grass Valley iControl navigator tool.

Design, Test and Run

When an Orbit MapView screen is open, it can be:

- edited in Orbit 'Design Mode';
- exercised in Orbit 'Test Mode'; or
- run in Orbit 'Run Mode'.

Orbit Design Mode and Run Mode

To toggle between 'Run Mode' and 'Design Mode':

- Click on the **Run Mode / Design Mode** button in the main menu. See Figure 2-29 and Figure 2-30.

Run Mode / **Design Mode** Button in main tool bar

(In 'Design Mode', click button to enter 'Run Mode', and vice versa.)

Properties Box

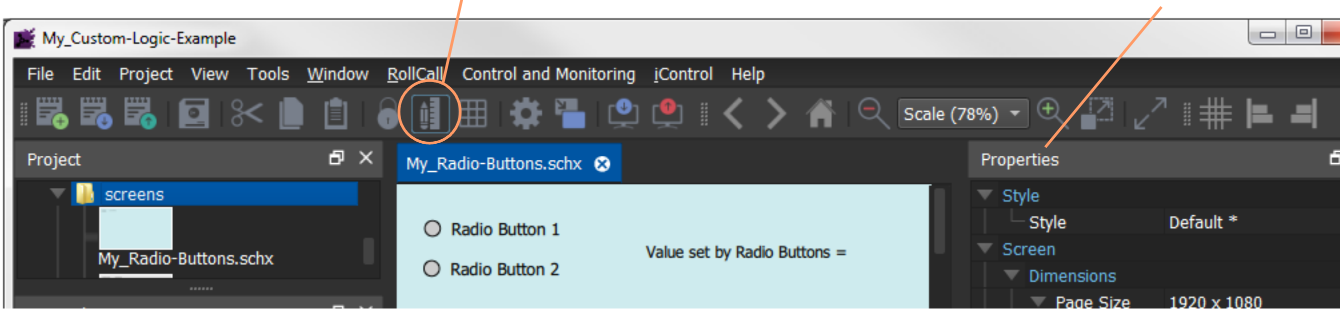


*Fig. 2-29: **Run Mode / Design Mode** Button*

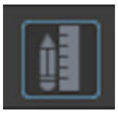
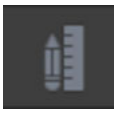
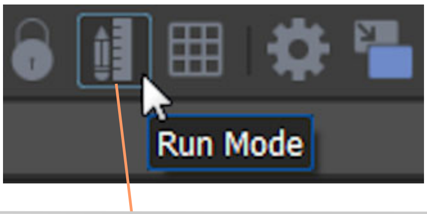
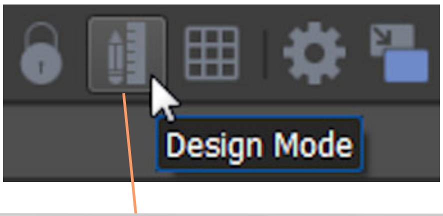
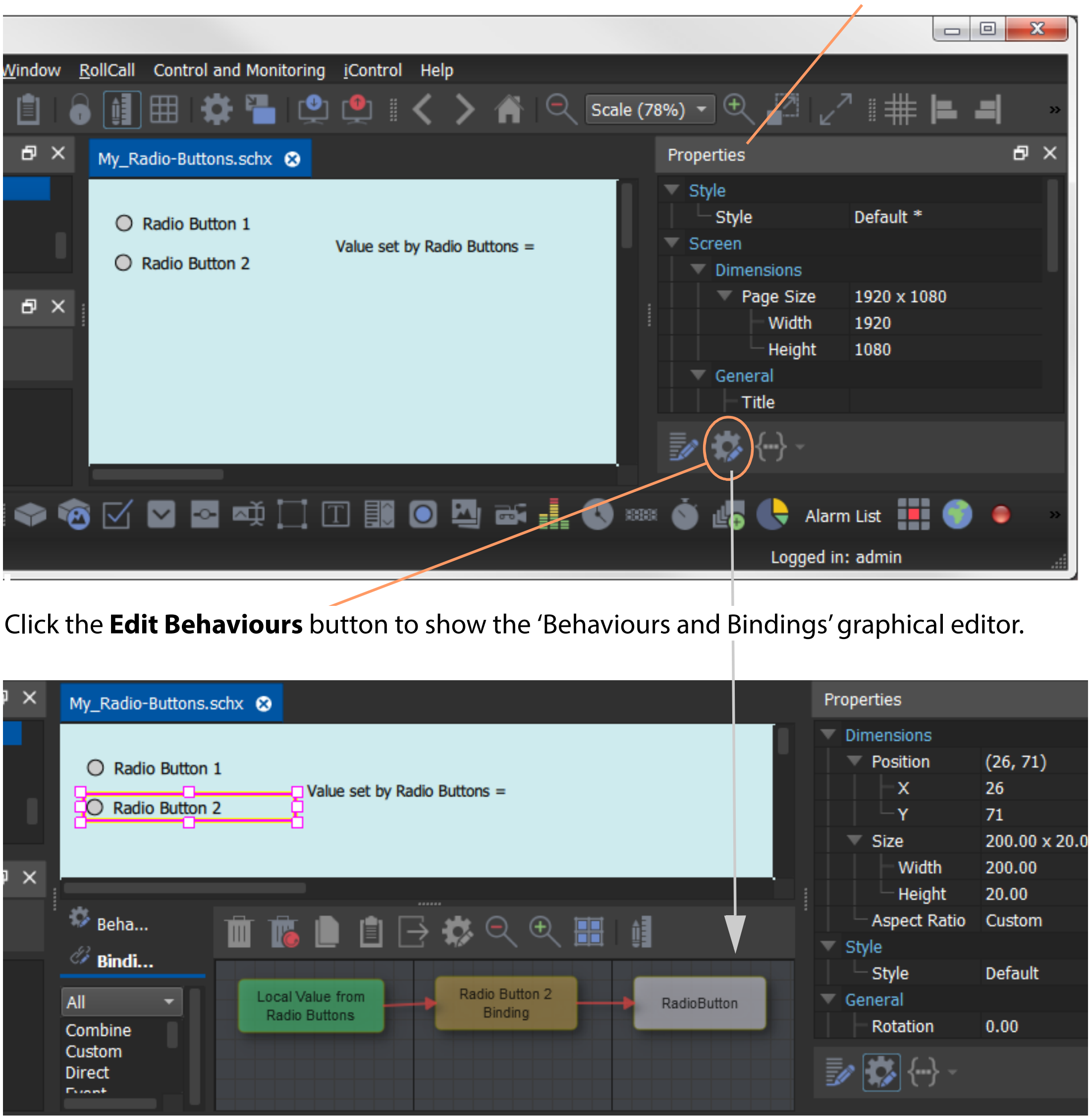
	Orbit in 'Design Mode'	**Orbit in 'Run Mode'**
Icon appearance		
Icon when hovering the cursor over icon		
Click icon to...	Click icon to enter 'Run Mode'	Click icon to enter 'Design Mode'

Fig. 2-30: Design/Run Mode Icon Appearance and Usage

Designing

In ‘Design Mode’, a screen may be edited, widgets can be added and configured, and any Behaviours and/or Bindings can be set up.

In ‘Design Mode’, click the **Edit Behaviours** button to show the ‘Behaviours and Bindings’ graphical editor. See Figure 2-31.



Properties Box

Click the **Edit Behaviours** button to show the ‘Behaviours and Bindings’ graphical editor.

Properties	
Style	
Style	Default *
Screen	
Dimensions	
Page Size	1920 x 1080
Width	1920
Height	1080
General	
Title	

Properties	
Dimensions	
Position	(26, 71)
X	26
Y	71
Size	200.00 x 20.0
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00

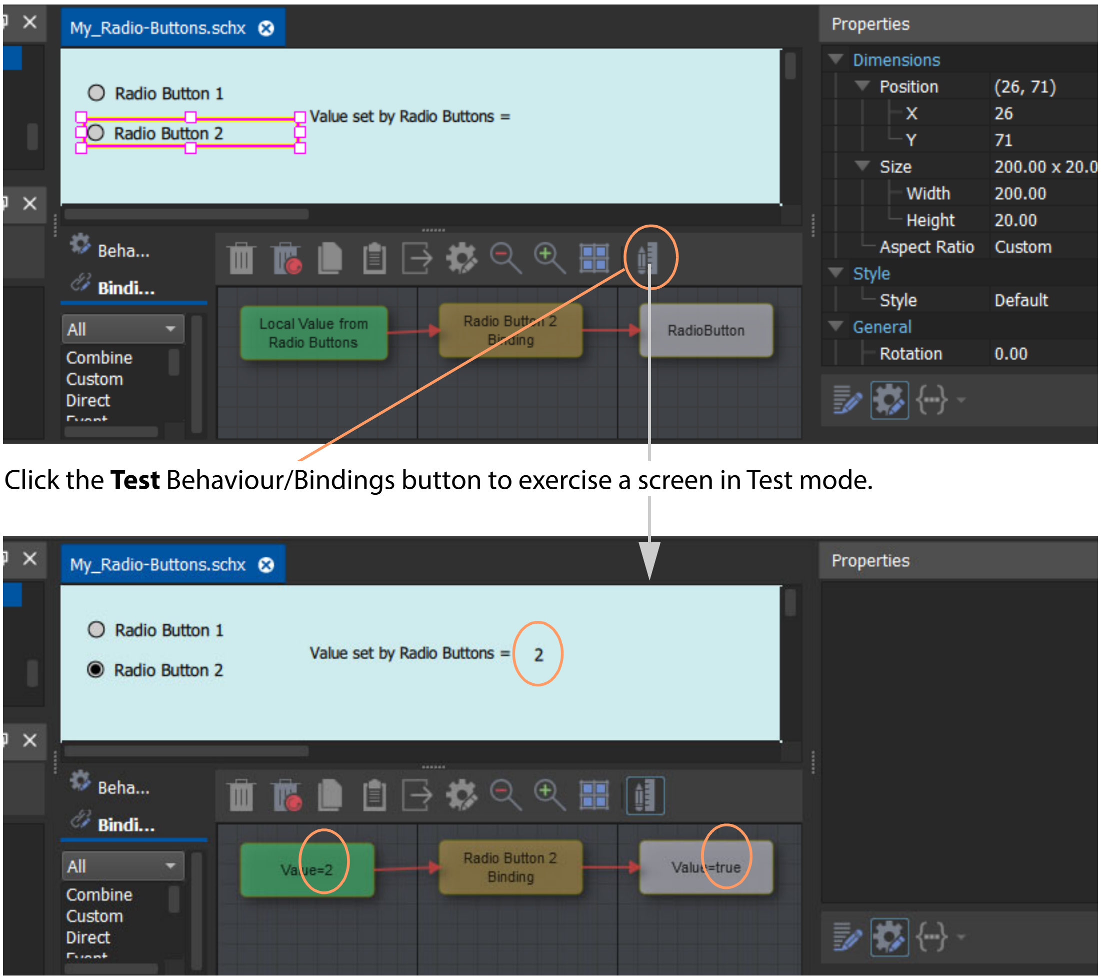
Fig. 2-31: Showing the ‘Behaviours and Bindings’ Graphical Editor in Design Mode

Testing

Behaviours and Bindings on a screen can be exercised/tested from ‘Design Mode’ *without* having to formally enter ‘Run Mode’. Test mode is a sort of ‘Run Mode’ but no license is needed.

Click the **Test** button (see Figure 2-32) to enter ‘Test’ mode. In ‘Test’ mode, Behaviour/widget values can be explicitly seen in the ‘Behaviour and Bindings’ graphical editor while the design is running.

Click the **Test** button again to stop exercising and revert to ‘Design Mode’.



Properties	
Dimensions	
Position	(26, 71)
X	26
Y	71
Size	200.00 x 20.0
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00

Click the **Test** Behaviour/Bindings button to exercise a screen in Test mode.

In 'Test' mode, Widget/Behaviour values are shown.

Fig. 2-32: Test Mode

Running

Enter 'Run Mode' to run the whole project on the client PC. This requires an Orbit license.

Users, Roles and Permission

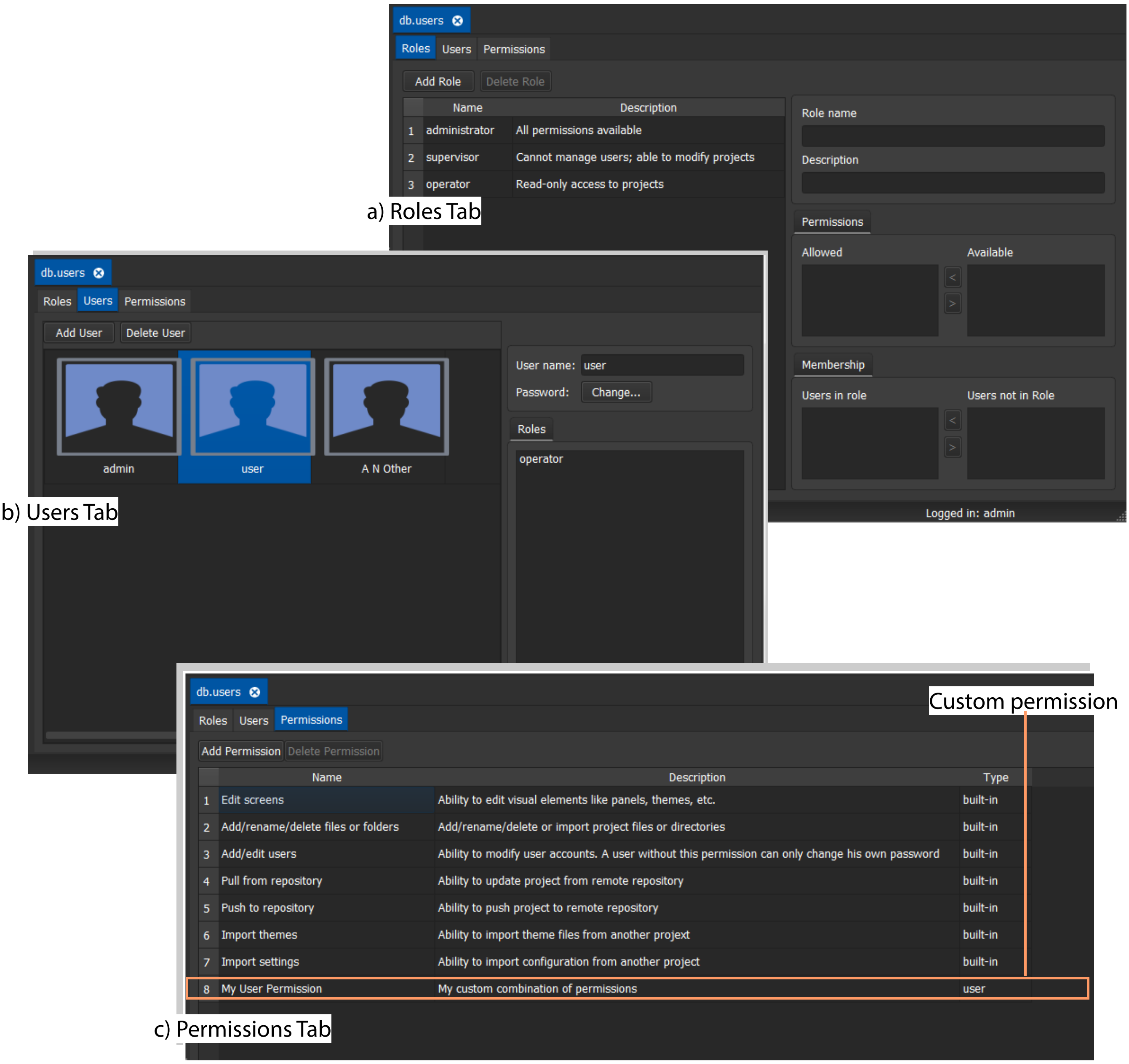
By default, a new Orbit project has one user already set up:

user name = 'admin';

password = 'admin'.

To set up project login details for individual people and/or roles:

1 Click on the **Manage Users** icon on the project home screen.
Three tabs are shown: Roles, Users and Permissions. See Figure 2-33.



a) Roles Tab

	Name	Description
1	administrator	All permissions available
2	supervisor	Cannot manage users; able to modify projects
3	operator	Read-only access to projects

b) Users Tab

User name: user

Password: Change...

Roles: operator

c) Permissions Tab

Custom permission

	Name	Description	Type
1	Edit screens	Ability to edit visual elements like panels, themes, etc.	built-in
2	Add/rename/delete files or folders	Add/rename/delete or import project files or directories	built-in
3	Add/edit users	Ability to modify user accounts. A user without this permission can only change his own password	built-in
4	Pull from repository	Ability to update project from remote repository	built-in
5	Push to repository	Ability to push project to remote repository	built-in
6	Import themes	Ability to import theme files from another project	built-in
7	Import settings	Ability to import configuration from another project	built-in
8	My User Permission	My custom combination of permissions	user

Fig. 2-33: Manage Users Tabs: a) Roles; b) Users; and c) Permissions

2 Create roles with certain permissions as required in the Roles tab.

3 Create new users in the Users tab.

4 Assign a role to a user in the Roles tab.

Note: Refer to the 'Orbit Introduction' user manual for more information.

Custom Permissions

Custom permissions can be created alongside the existing permissions and they can be assigned to roles. These allow user access to be tailored to operational needs.

Using custom permissions, access-control to Orbit MapView project screens or widgets can be designed into an Orbit MapView project.

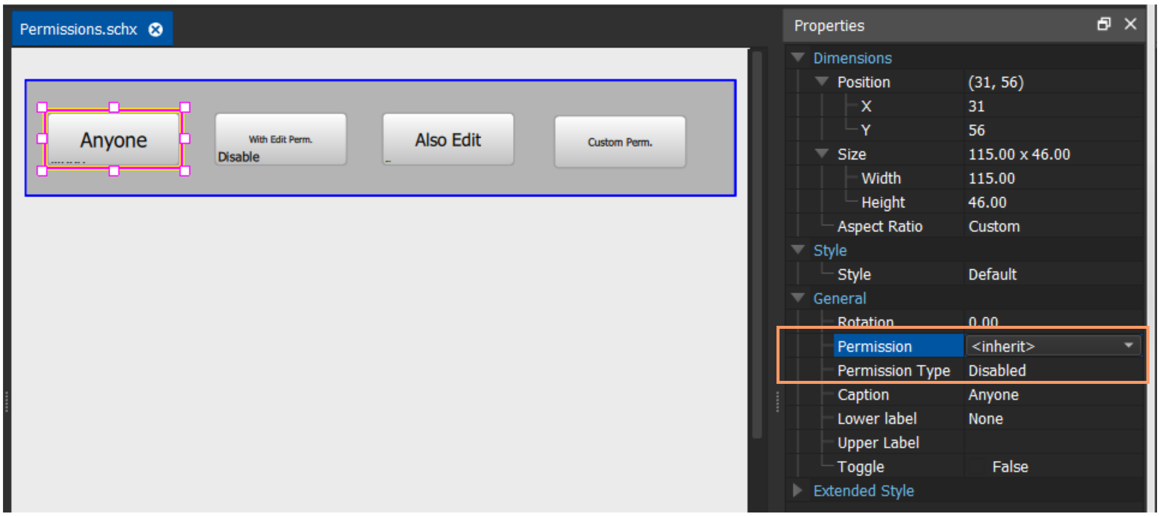
Permission Widget Properties

Widgets have 'Permission' properties. These determine whether a) a logged-in user may use a widget or access screens in a running Orbit MapView project; and b) how it appears on-screen. This is based on a user's 'Role', which will include or exclude various permissions.

For example, access to a screen with certain operationally-sensitive controls can be limited to any logged-in users whose role includes a certain permission. For example, the 'Edit screens' permission, or a custom permission. Access to a **Button** widget which accesses sensitive screen(s) can thus be controlled.

Widget permission properties include:

- **Permission**
- **Permission Type**

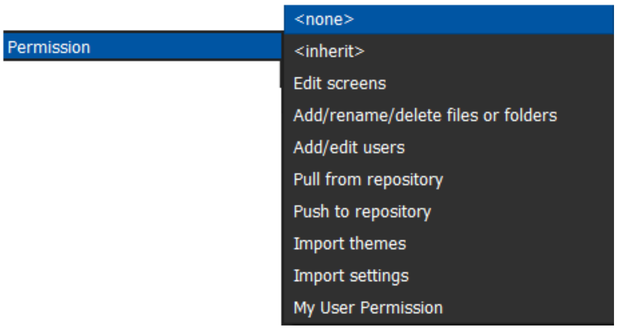


Properties

Dimensions	
Position	(31, 56)
X	31
Y	56
Size	115.00 x 46.00
Width	115.00
Height	46.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	Anyone
Lower label	None
Upper Label	
Toggle	False
Extended Style	

Fig. 2-34: Widget Permission Properties

Table 2-6: Permission Properties

Property	Description
Permission	Drop down box.  Permission <none> <inherit> Edit screens Add/rename/delete files or folders Add/edit users Pull from repository Push to repository Import themes Import settings My User Permission Options: • **<none>** - No access control on the widget. • **<inherit>** - Widget inherits permission from parent widget/screen/component. A widget may only be used if user role has: • **Edit screens** - screen/component/theme editing permission. • **Add/rename/delete files or folders** - project file/folder editing permission. • **Add/edit users** - ability to modify user accounts etc. • **Pull from repository** - project ‘push to a repository/device’ capability. • **Push to repository** - project ‘pull from a repository/device’ capability. • **Import themes** - ability to import a theme (from another project). • **Import settings** - ability to import configurations from another project. • **My User Permission** - user-defined permission. Figure 2-33c lists the permissions.
Permission Type	Drop down box. Defines how a widget appears on-screen if a user does not have the correct permission. • **Disabled** - Select to show widget grayed-out on screen if user does not have sufficient permission. • **Hidden** - Select to hide widget on screen if user does not have sufficient permission.

Themes

Orbit MapView projects are typically a set of user screens with graphical widgets. The appearance of widgets can be customized by using **Themes**. A **Theme** in Orbit is a collection of widget styles. A **Theme** can be applied to a whole project or on a widget-by-widget basis.

Note: For information about **Themes**, see the 'Orbit for Multiviewers' User Manual.

Create a Theme

To create a new **Theme**:

1 Right-click in the **Project View** on the 'themes' folder. And select 'New Theme'. See Figure 2-35.

A new **Theme** is created.

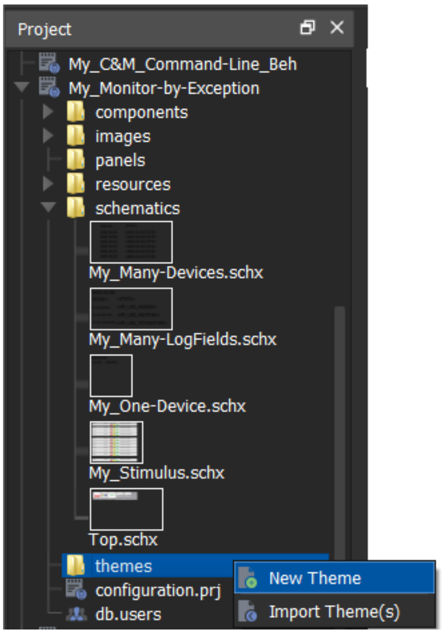
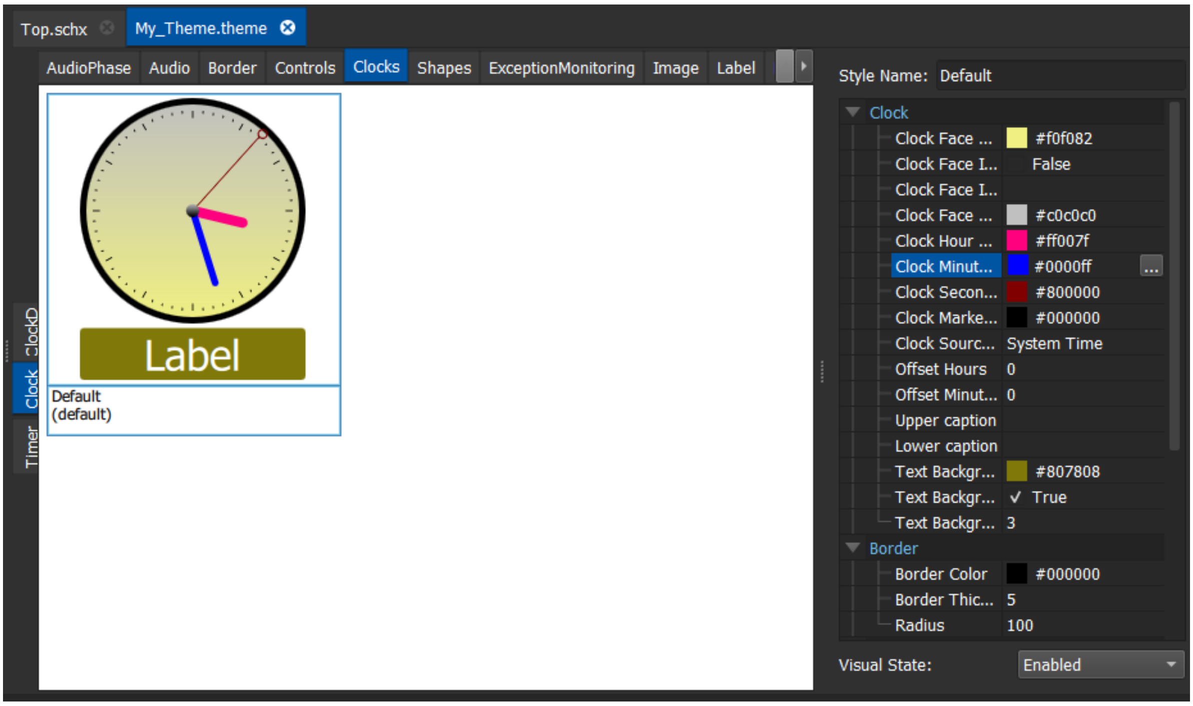


Fig. 2-35: New Theme

2 You are prompted if you wish the new theme to be the current theme. (This can be changed at any time.)

The **Theme Editor** is shown in the Orbit MapView window. For example, Figure 2-36.



Style Name: Default

Clock	
Clock Face ...	#f0f082
Clock Face I...	False
Clock Face I...	
Clock Face ...	#c0c0c0
Clock Hour ...	#ff007f
Clock Minut...	#0000ff
Clock Secon...	#800000
Clock Marke...	#000000
Clock Sourc...	System Time
Offset Hours	0
Offset Minut...	0
Upper caption	
Lower caption	
Text Backgr...	#807808
Text Backgr...	✓ True
Text Backgr...	3
Border	
Border Color	#000000
Border Thic...	5
Radius	100

Visual State: Enabled

Fig. 2-36: Theme Editor

In the **Theme** editor, widget appearance can be changed for the new **Theme**.

Set as Current Theme

A new Theme can be set as the current **Theme** to be used by:

1 Right-clicking on the **Theme** in the **Project View** and selecting ‘Set as Current Theme’.

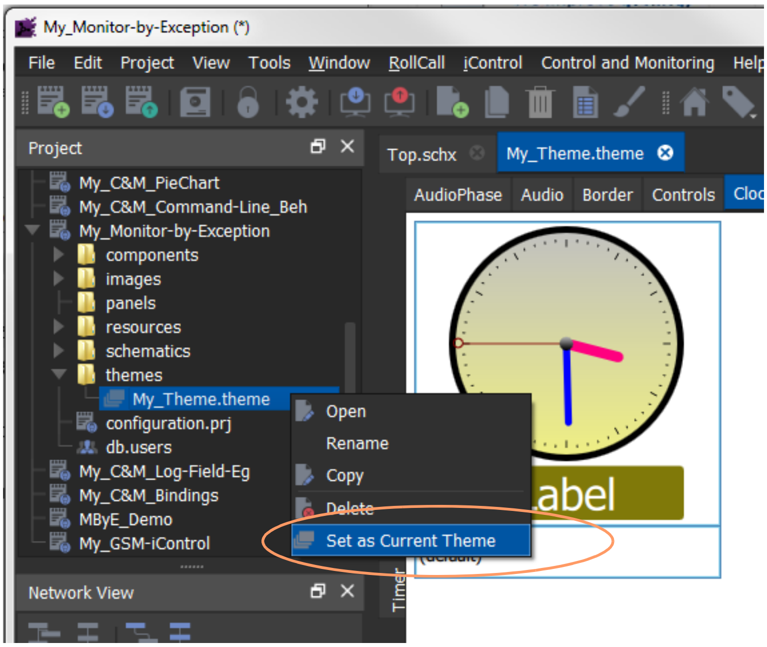


Fig. 2-37: Set as Current Theme

3 Working with Widgets

Summary

Working with Widgets

This chapter describes how to work with the on-screen graphical widgets. Most widgets are described in the 'Orbit for Multiviewers' user manual.

Widget Properties and Value

Orbit on-screen widgets may have:

- Properties.
- A value.

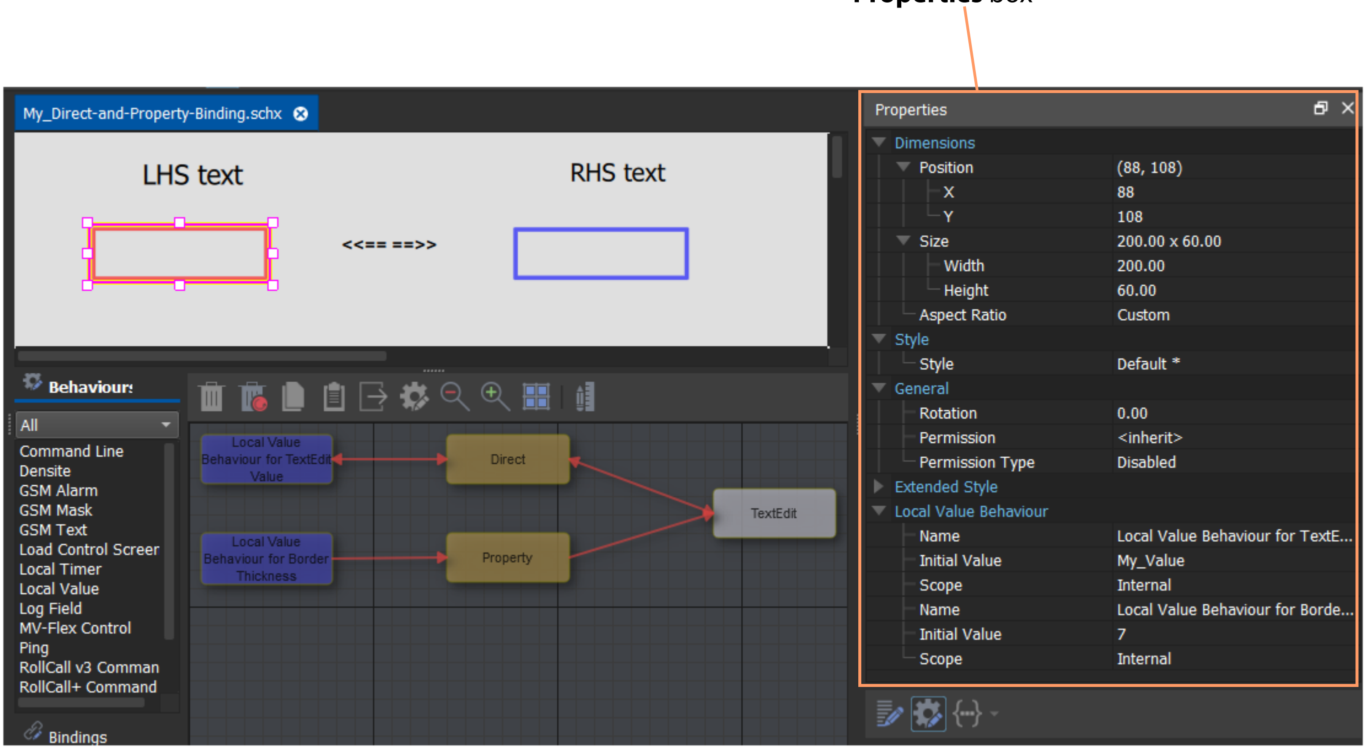
Properties and the Property Binding

Properties control a widget's physical appearance or configuration (for example, border thickness, border color).

Properties Box

Properties can be *manually* set in the **Properties Box**. (See Figure 3-1.)

Properties box



Dimensions	
Position	(88, 108)
X	88
Y	108
Size	200.00 x 60.00
Width	200.00
Height	60.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Extended Style	
Local Value Behaviour	
Name	Local Value Behaviour for TextE...
Initial Value	My_Value
Scope	Internal
Name	Local Value Behaviour for Borde...
Initial Value	7
Scope	Internal

Fig. 3-1: Properties Box

Property Binding and Direct Binding

A **Property** Binding is used to access widget properties via Orbit Behaviours and Bindings; whereas a **Direct** Binding accesses a widget value. Figure 3-2a and b shows an example widget with Orbit Bindings accessing a widget's value and properties for Orbit Behaviours.

Value and the Direct Binding

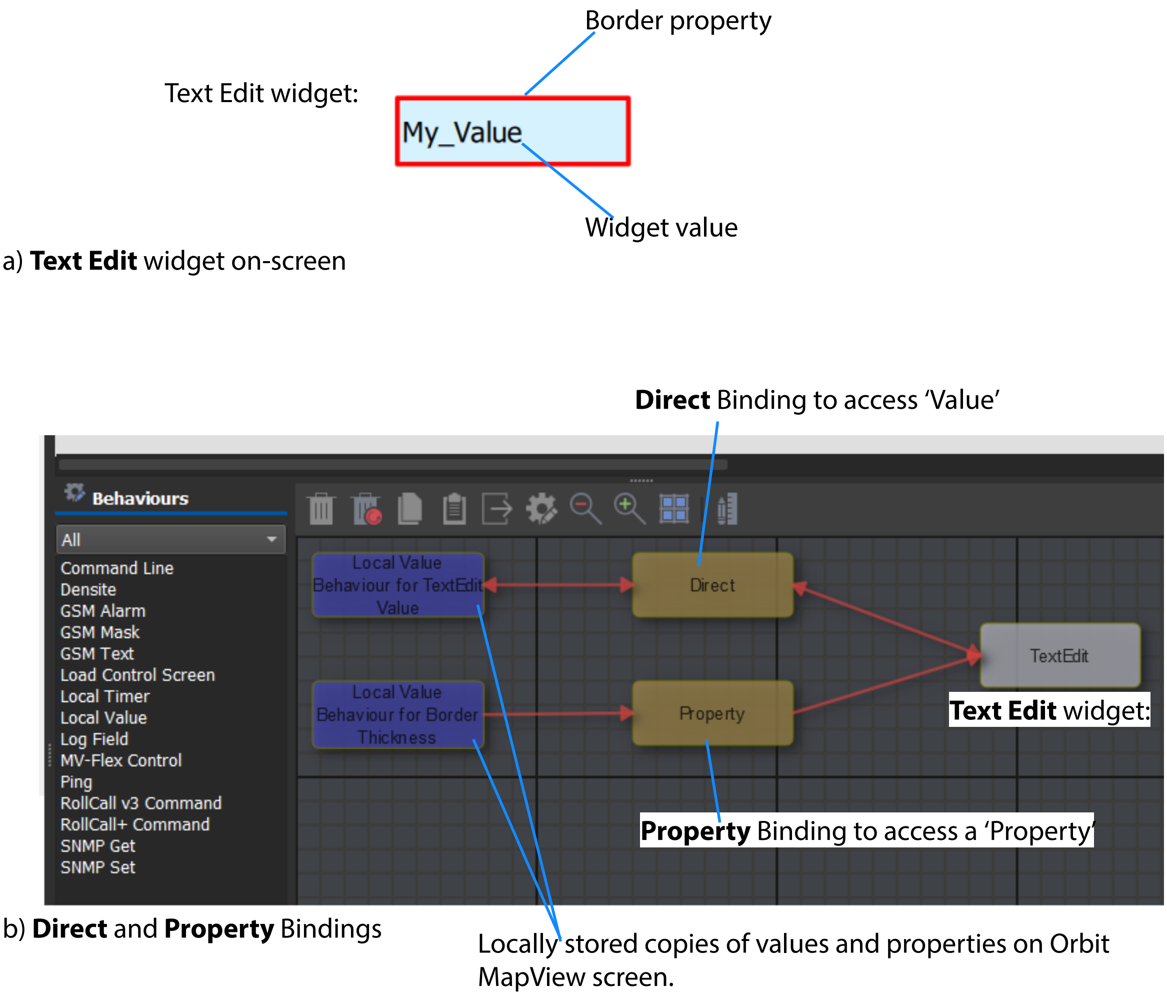
A widget value normally matches the content/current value of the widget. For example, content of a text box.

> Note: A widget's value may also include some additional value attributes (for example, 'visibility', 'read-only' and 'error state') which may be hidden from the user and only used programmatically.

However, some widgets do not have a 'value' (for example, a Rectangle widget has no sensible 'value').

A **Direct** Binding is used to access a widget's value.

Figure 3-2 shows the use of Bindings to access a widget's value and properties.



a) **Text Edit** widget on-screen

b) **Direct** and **Property** Bindings

Fig. 3-2: Widget Example:
a) Text Edit Widget On-Screen.
b) Direct and Property Bindings.

Using Behaviours and Bindings

Many different functionalities may be defined on Orbit MapView screens. Screens may contain graphical widgets whose inter-action with other widgets and to other monitored values is defined with Behaviours and Bindings.

To define the logic behind widgets, widget property values and Behaviour values are linked with Bindings.

A Binding can link:

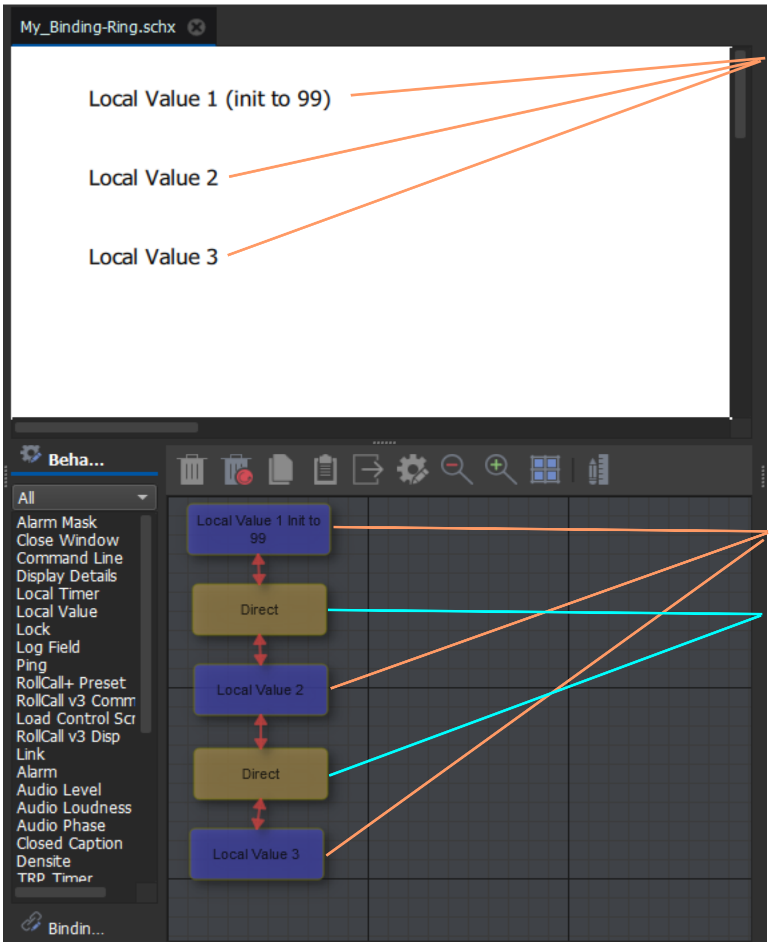
- A Behaviour value and another Behaviour value; or
- A Behaviour value and a property value.

When a Binding is made between Behaviour_A and Behaviour_B/Property_B,
then the value of Behaviour_B/Value_B is always kept up-to-date from the value of Behaviour_A, and vice versa.

Behaviours and Bindings are listed in Appendix B List of Behaviours and Bindings, on page 355.

Stabilized Value

Example:
Figure 3-3 uses **Direct** Bindings to link three Behaviours (**Local Value** Behaviours) together. One Behaviour value is initialized to 99 on the screen.



Label widgets, used to display the values of the **Local Value** Behaviours

Local Value Behaviours

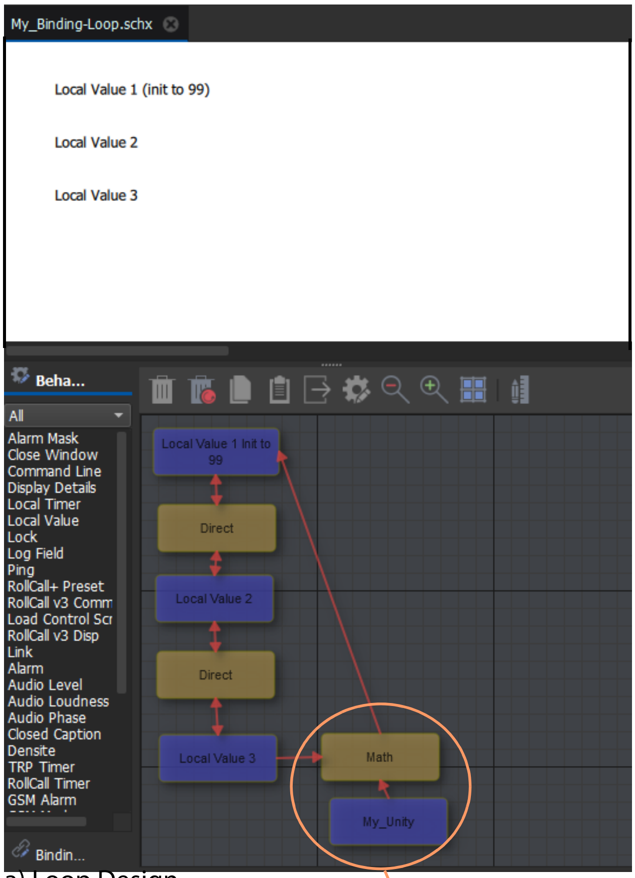
Direct Bindings

Fig. 3-3: Direct Bindings Connecting Behaviour Values

When this is run, the three Behaviour values are quickly all set to 99. The values have been updated and the values have settled, stabilized.

Non-Stabilized Values

However, if a Binding is used which changes a value (for example, a **Math** Binding which increments the value by one), then a loop could be created which upsets the 'settling down' of the value when the Orbit project is run. When the example in Figure 3-4 is run, all **Local Value** Behaviour values increase continually and the values do not settle - they have not stabilized. This kind of loop should be avoided.



My_Binding-Loop.schx

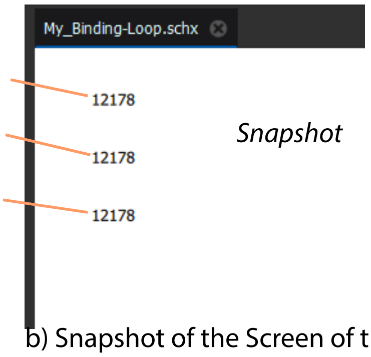
Local Value 1 (init to 99)

Local Value 2

Local Value 3

a) Loop Design

Math Binding adds one.



My_Binding-Loop.schx

'Local Value 1' value 12178

'Local Value 2' value 12178

'Local Value 3' value 12178

Snapshot

Values continually increment
by 1 indefinitely
(unstable loop).

b) Snapshot of the Screen of the Running Loop Design

Fig. 3-4: Avoid Non-Stabilized Loops

Note: Avoid unstable loops forming with Bindings and values.

Setting Up a Test Bench

As with any design procedure, it is useful and beneficial to be able to exercise and test parts and/or all of a design; this is the case with Orbit widgets, Behaviours and Bindings etc. A 'test bench' screen can be set up to provide stimulus to and monitor the behavior of a design, a screen or set of screens. This is shown with an example is this section.

The **Pie Chart** widget will be the widget under test here. A widget test bench screen can be set up to provide dynamic data to the widget by using a **Property** Binding to bind Orbit Behaviour values to the pie chart segments.

For an example system here, data will be available from some system device ('current data rate' and a 'maximum data rate') via Orbit Behaviours. (These are derived from RollCall log fields or from some other source available to Orbit.) In our test bench example, on-screen **Text Edit** widgets are used to source the data values and the following are covered:

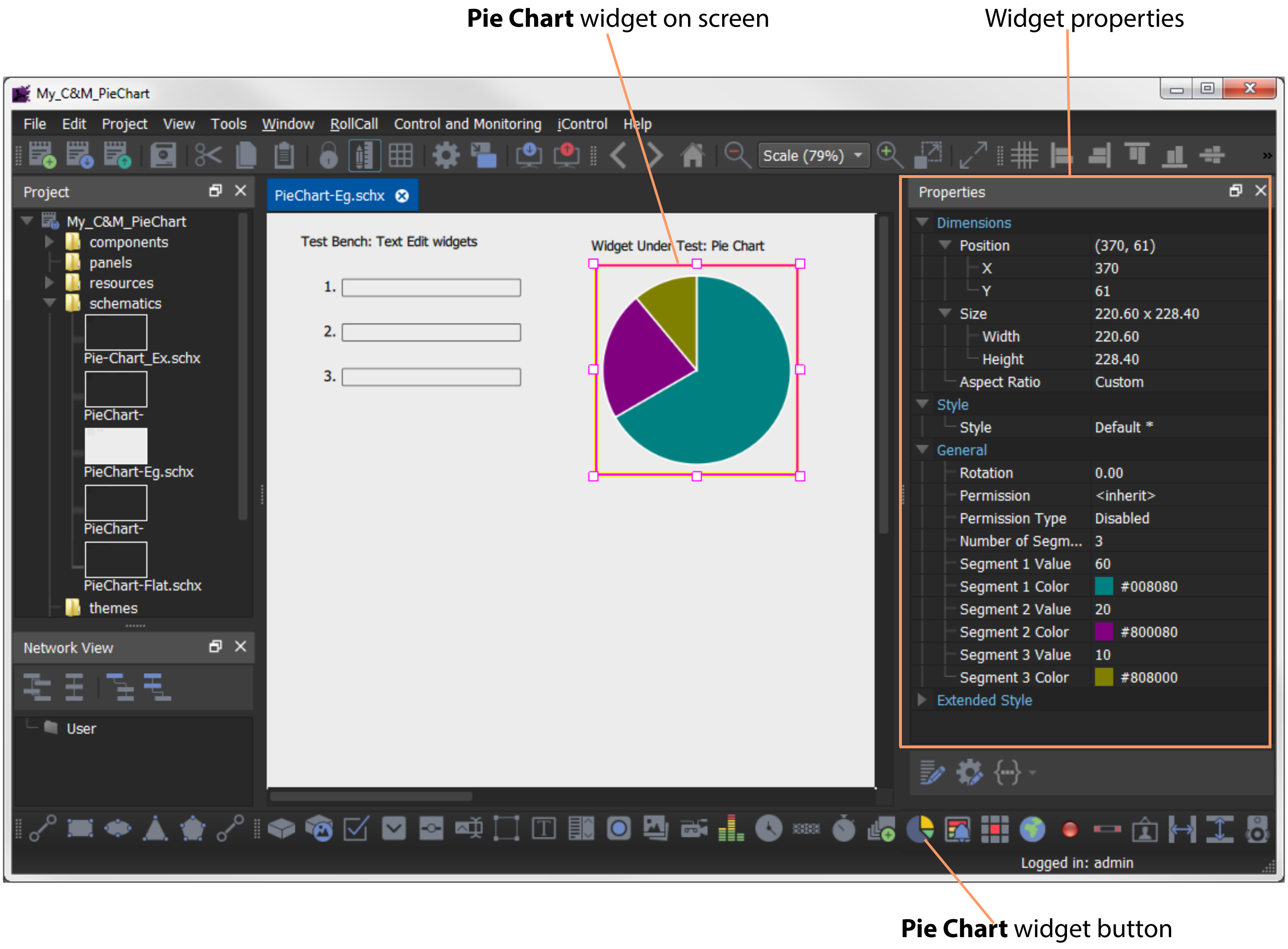
- Pie Chart Widget, on page 47;
- Create a Test Bench Screen, on page 48;
- Calculate an Intermediate Data Value, on page 50;
- Bind 'Widget Under Test' to Stimulus Values, on page 53; and
- Exercise the 'Widget Under Test' with Test Bench, on page 56.

Pie Chart Widget

The **Pie Chart** widget is presented here and is used in some of the following examples to demonstrate designing and running Orbit MapView projects.

The **Pie Chart** widget can be configured with 2 or more segments and is designed to show the relative proportions of two or more data values. See Figure 3-5. The segment sizes displayed represent each data value's proportion of the total. Data values are entered as property values (property 'segment value').

For example, for segment values set to 60, 20, and 10, the displayed pie chart segments show 67%, 22%, and 11% proportions (i.e. 60/(60+20+10); 20/(60+20+10); and 10/(60+20+10)). See Figure 3-5.



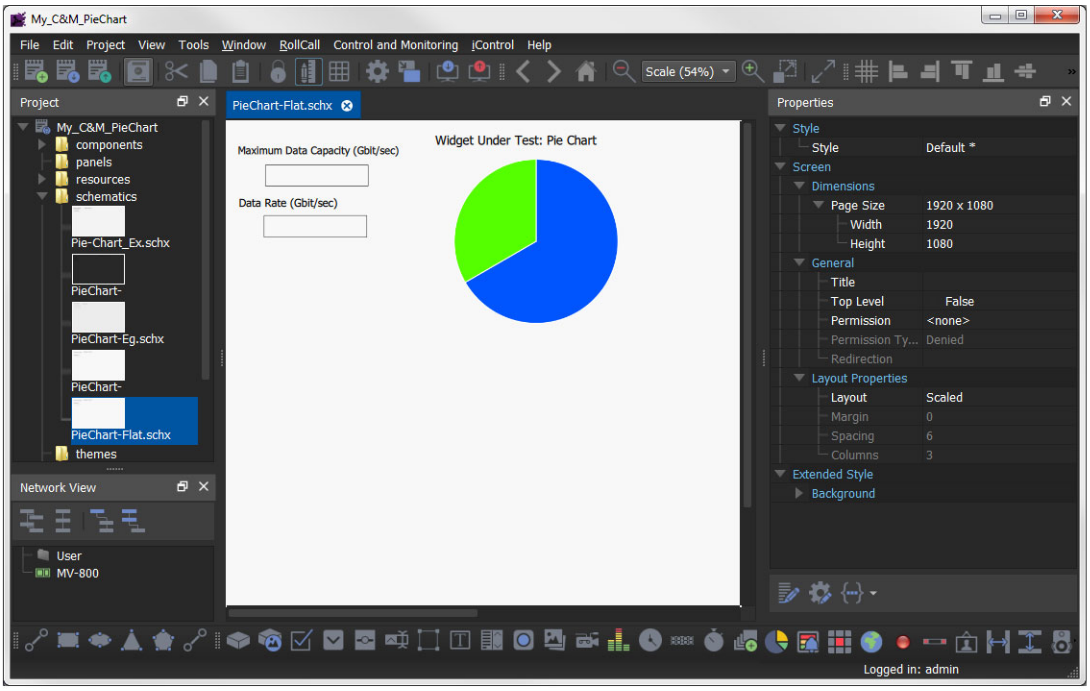
Properties	
Dimensions	
Position	(370, 61)
X	370
Y	61
Size	220.60 x 228.40
Width	220.60
Height	228.40
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Number of Segm...	3
Segment 1 Value	60
Segment 1 Color	#008080
Segment 2 Value	20
Segment 2 Color	#800080
Segment 3 Value	10
Segment 3 Color	#808000
Extended Style	

Fig. 3-5: Pie Chart Widget

Create a Test Bench Screen

The following example exercises a **Pie Chart** widget on a "test bench" screen. This uses data from Orbit Behaviours which bound to some **Pie Chart** properties:

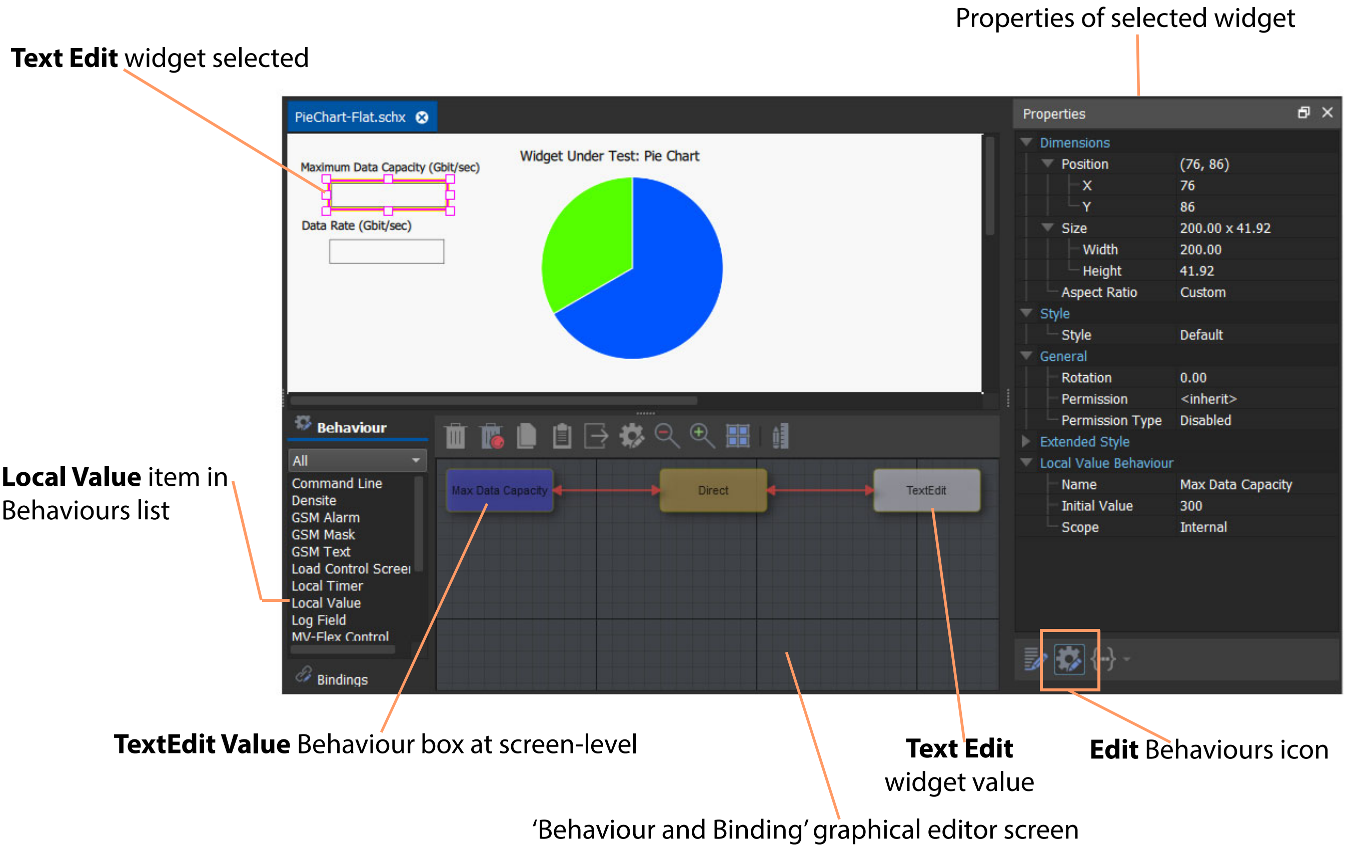
1 Generate a screen schematic and drag a **Pie Chart** and two **Text Edit** widgets onto the screen. See Figure 3-6.



Style	
Style	Default *
Screen	
Dimensions	
Page Size	1920 x 1080
Width	1920
Height	1080
General	
Title	
Top Level	False
Permission	<none>
Permission Ty...	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	
Background	

Fig. 3-6: Screen with a Pie Chart and two Text Edit Widgets

2 Select the first **Text Edit** widget and click the **Edit** Behaviours icon.
The 'Behaviour and Binding' graphical editor is shown in the lower half of the schematic editor area. See Figure 3-7.



Properties of selected widget
Text Edit widget selected
Local Value item in Behaviours list
TextEdit Value Behaviour box at screen-level
Text Edit widget value
Edit Behaviours icon
'Behaviour and Binding' graphical editor screen
PieChart-Flat.schx
Maximum Data Capacity (Gbit/sec)
Data Rate (Gbit/sec)
Widget Under Test: Pie Chart
Behaviour
All
Command Line
Densite
GSM Alarm
GSM Mask
GSM Text
Load Control Screen
Local Timer
Local Value
Log Field
MV-Flex Control
Bindings
Max Data Capacity
Direct
TextEdit
Properties

Dimensions	
Position	(76, 86)
X	76
Y	86
Size	200.00 x 41.92
Width	200.00
Height	41.92
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Extended Style	
Local Value Behaviour	
Name	Max Data Capacity
Initial Value	300
Scope	Internal

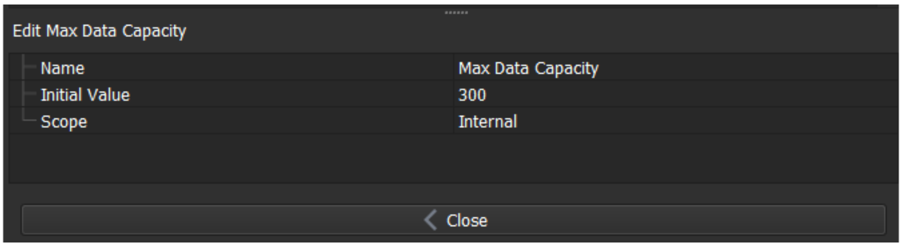
Fig. 3-7: Setting Up Behaviours

3 Click on the **Local Value** item in the Behaviours list.
This places a **TextEdit Value** Behaviour box onto the graphical editor screen. See Figure 3-7.

4 Double-click on the **TextEdit Value** Behaviour box in the graphical editor screen to change some Behaviour settings (see Figure 3-8):

- Change **Name** to Max Data Capacity
- Change **Initial Value** to 300

5 Click **Close**.



Edit Max Data Capacity

Name	Max Data Capacity
Initial Value	300
Scope	Internal

Fig. 3-8: Editing Text Edit Behaviour Settings

6 Repeat these steps for the second **Text Edit** widget, setting:

- Change **Name** to Current Data Rate
- Change **Initial Value** to 20

This has created two Orbit Behaviours on a test bench screen to hold two data values for our example interface ('maximum data capacity' and 'current data rate'). This will allow us to exercise the **Pie Chart** widget with the test bench screen.

Calculate an Intermediate Data Value

To make a **Pie Chart** show the amount of used and unused data capacity, an ‘Available Data Capacity’ figure must be calculated on the test bench screen from the available data. This calculation is done in a Binding and the value is held in a Behaviour.

1 Select the **Pie Chart** on the test bench screen
and click the **Edit** Behaviours icon to show the ‘Behaviour and Binding’ graphical editor.

2 Display the Bindings list alongside the ‘Behaviour and Bindings’ graphical editor (click on ‘Bindings’ side-tab).

3 Click the **Math** item in the Bindings list.
A **Math** Binding box appears on the graphical editor. See Figure 3-9.

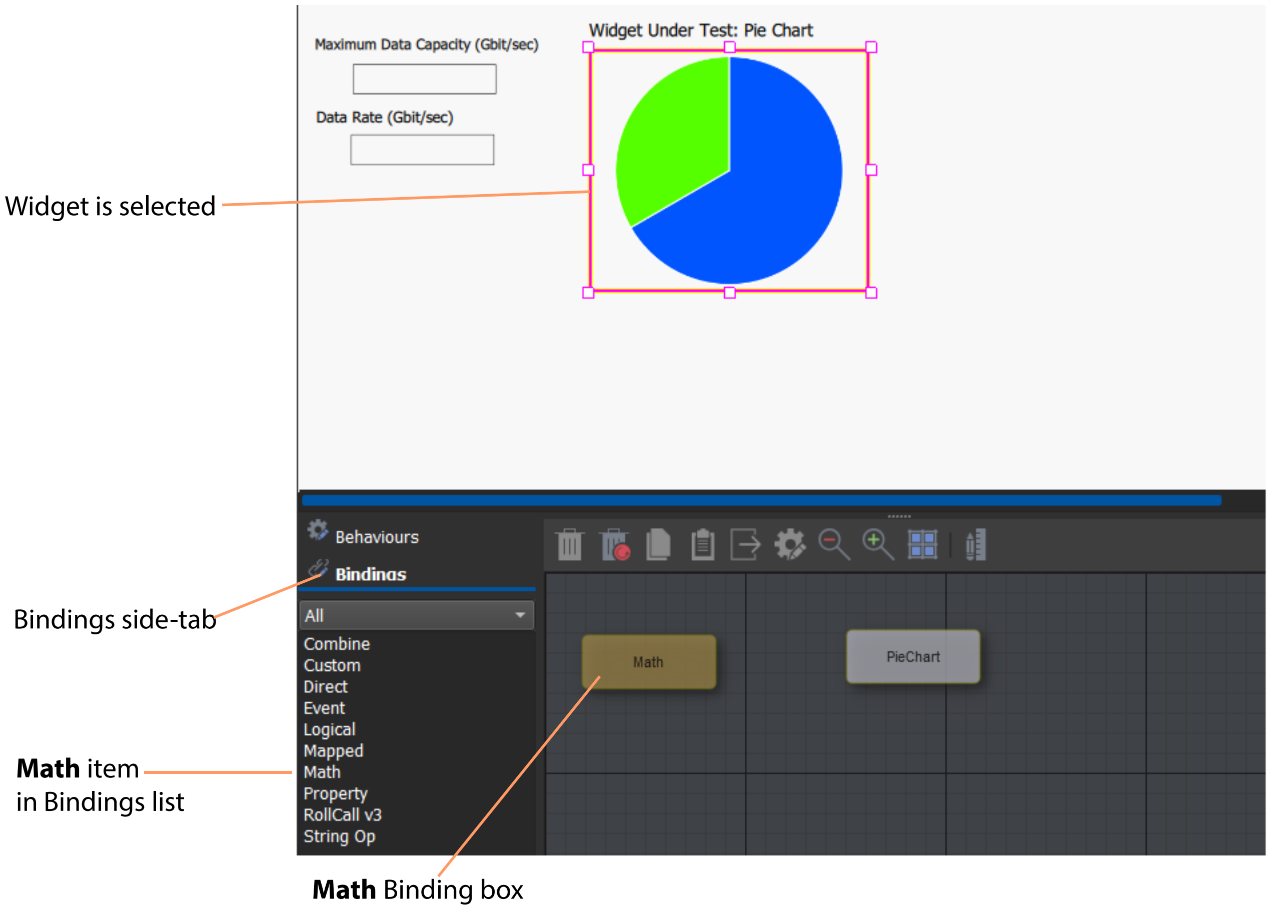
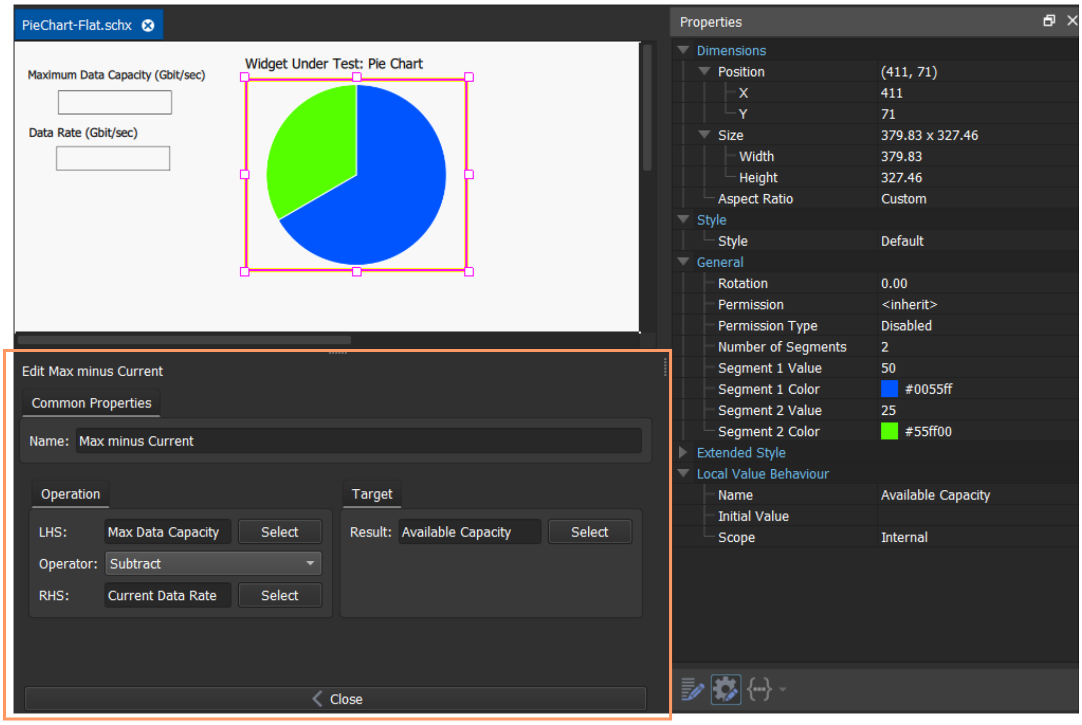


Fig. 3-9: Math Binding

4 Double-click on the **Math** Binding box.
A **Edit Math** Binding edit screen is shown. See Figure 3-10.



Edit Max minus Current

Name: Max minus Current

LHS: Max Data Capacity

Operator: Subtract

RHS: Current Data Rate

Result: Available Capacity

Dimensions	
Position	(411, 71)
X	411
Y	71
Size	379.83 x 327.46
Width	379.83
Height	327.46
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Number of Segments	2
Segment 1 Value	50
Segment 1 Color	#0055ff
Segment 2 Value	25
Segment 2 Color	#55ff00
Extended Style	
Local Value Behaviour	
Name	Available Capacity
Initial Value	
Scope	Internal

Fig. 3-10: Edit Math Binding Screen

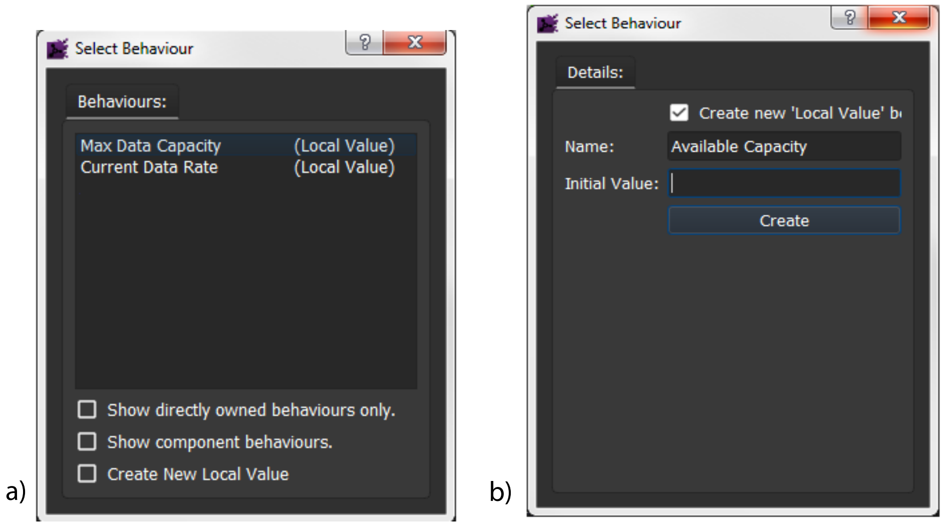
5 In the **Edit** Binding screen:

- Set **Name** to Max minus Current
- Select **LHS** to be the Behaviour value Max Data Capacity
- Set **Operator** to be subtract
- Select **RHS** to be the Behaviour value Current Data Rate

6 Select **Target Result** to 'Create New Local Value' (check box, see Figure 3-11a) and a further dialog is shown (Figure 3-11b).

Then set **Name** to be Available Capacity and click **Create**.

The resulting **Math** Binding settings are shown in Figure 3-12.



Max Data Capacity (Local Value)
Current Data Rate (Local Value)

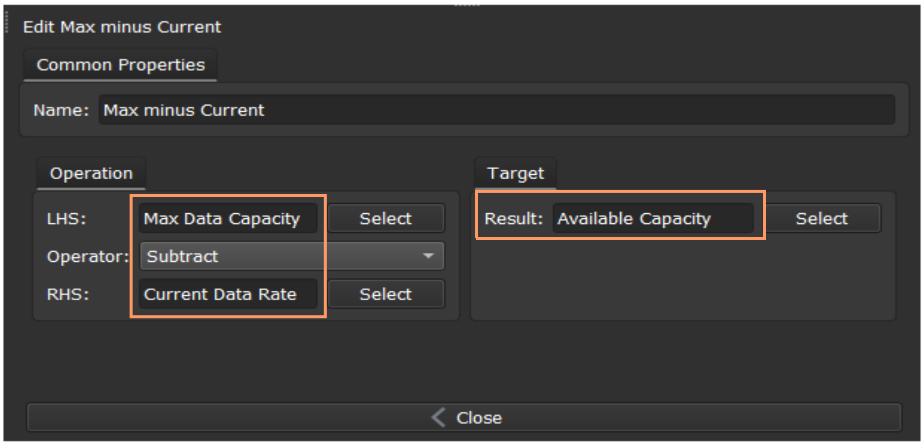
□ Show directly owned behaviours only.
□ Show component behaviours.
□ Create New Local Value

☑ Create new 'Local Value' b

Name: Available Capacity

Initial Value:

Fig. 3-11: Select Behaviour Dialogs



Edit Max minus Current

Common Properties

Name: Max minus Current

Operation

LHS: Max Data Capacity [Select]

Operator: Subtract

RHS: Current Data Rate [Select]

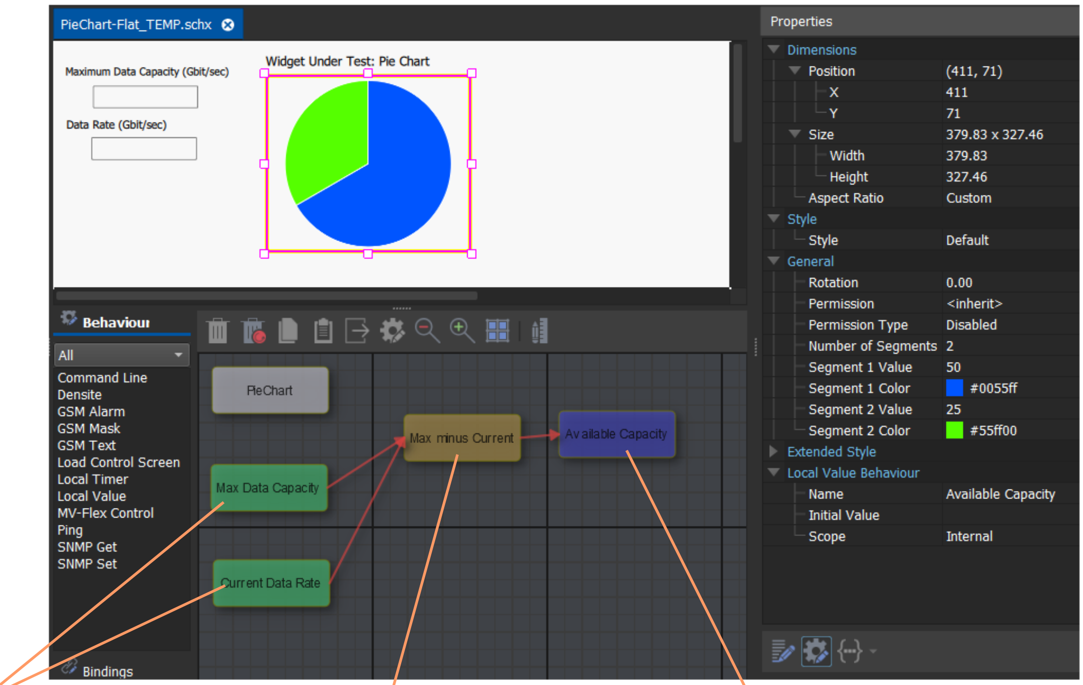
Target

Result: Available Capacity [Select]

Close

Fig. 3-12: Math Binding Settings

7 Click **Close**.
The resulting graphical editor screen is shown in Figure 3-13.



PieChart-Flat_TEMP.schx

Maximum Data Capacity (Gbit/sec)

Data Rate (Gbit/sec)

Widget Under Test: Pie Chart

Behaviour

All

Command Line
Densite
GSM Alarm
GSM Mask
GSM Text
Load Control Screen
Local Timer
Local Value
MV-Flex Control
Ping
SNMP Get
SNMP Set

Bindings

Properties

Dimensions	
Position	(411, 71)
X	411
Y	71
Size	379.83 x 327.46
Width	379.83
Height	327.46
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Number of Segments	2
Segment 1 Value	50
Segment 1 Color	🟦 #0055ff
Segment 2 Value	25
Segment 2 Color	🟩 #55ff00
Extended Style	
Local Value Behaviour	
Name	Available Capacity
Initial Value	
Scope	Internal

Behaviour values at screen-level

Math Binding at widget-level

Local Value Behaviour holds the 'available data capacity'.

Fig. 3-13: Math Binding Set Up to Calculate 'Available Data Capacity'

A **Math** Binding is now set up to compute the **Available Capacity** (i.e. the difference between the 'Max Data Capacity' and 'Current Data Rate'), which is held in a new 'Local Value' Behaviour.

The **Math** Binding is associated with the instance of the selected **Pie Chart** widget.
The **Math** Binding is at widget-level on the screen.
The **Math** Binding uses two Behaviour values which are defined at screen-level.

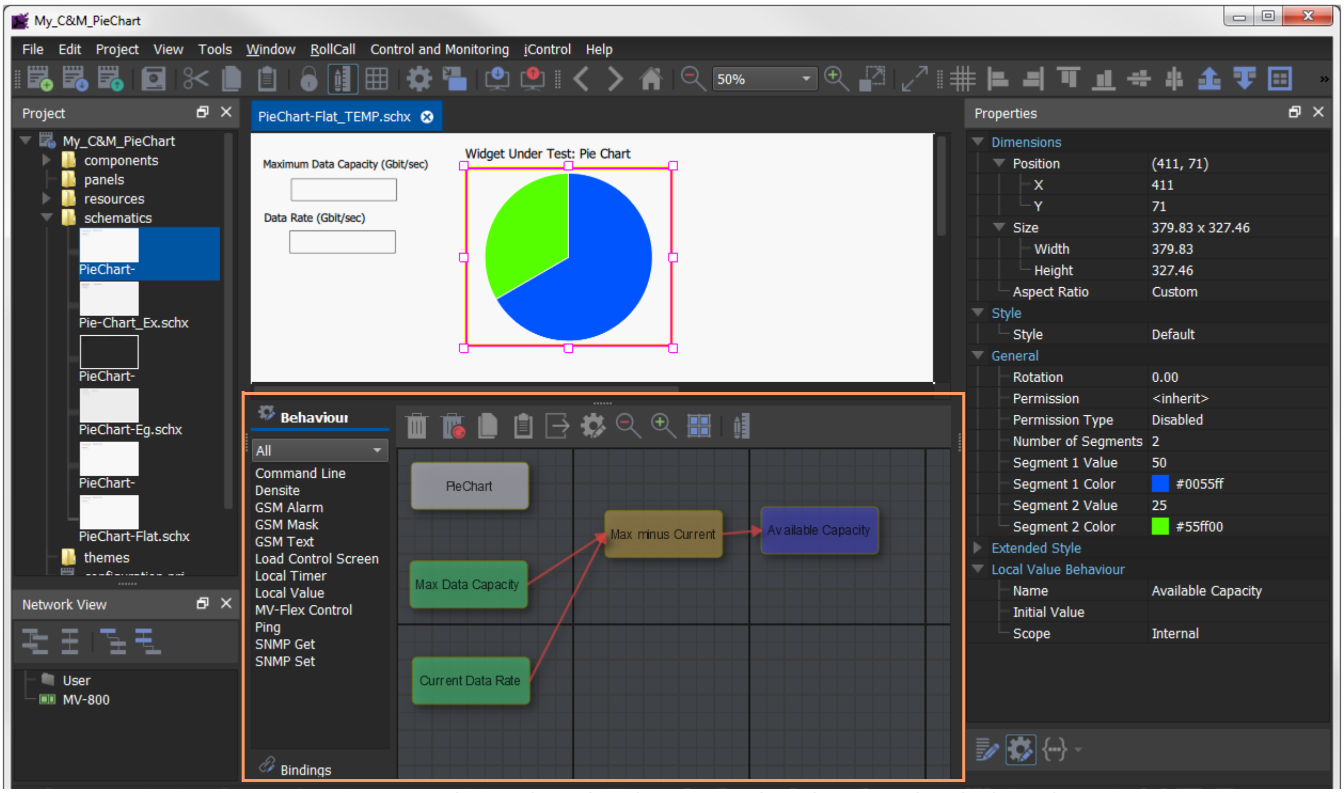
Note: The scope of a Behaviour or Binding or variable is either:

- screen-level; or
- component-level; or
- widget-level.

Bind ‘Widget Under Test’ to Stimulus Values

The ‘Current Data Rate’ and ‘Available Data Capacity’ data values are ready to be connected to the **Pie Chart** widget.

1 Select the **Pie Chart** on the test bench screen
and click the ‘Edit Behaviours’ icon to show the ‘Behaviour and Binding’ graphical editor. See Figure 3-14.



Dimensions	
Position	(411, 71)
X	411
Y	71
Size	379.83 x 327.46
Width	379.83
Height	327.46
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Number of Segments	2
Segment 1 Value	50
Segment 1 Color	🟦 #0055ff
Segment 2 Value	25
Segment 2 Color	🟩 #55ff00
Extended Style	
Local Value Behaviour	
Name	Available Capacity
Initial Value	
Scope	Internal

Fig. 3-14: Pie Chart Selected, ‘Behaviour and Binding’ Graphical Editor Shown

There are two properties of the Pie Chart widget that we need to connect (bind) to: ‘Segment 1 Value’ and ‘Segment 2Value’. Do this with **Property** Bindings:

2 Click the ‘Property’ item in the Bindings list *twice* to add two **Property** Bindings to the graphical editor. See Figure 3-15.

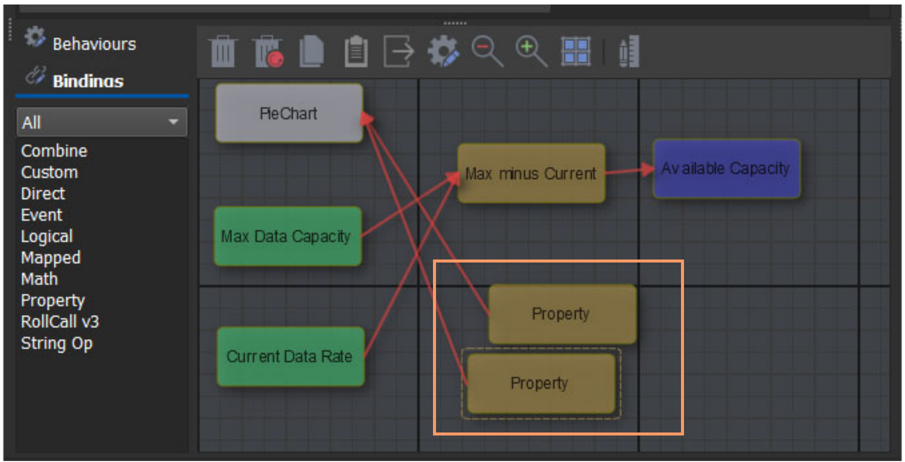
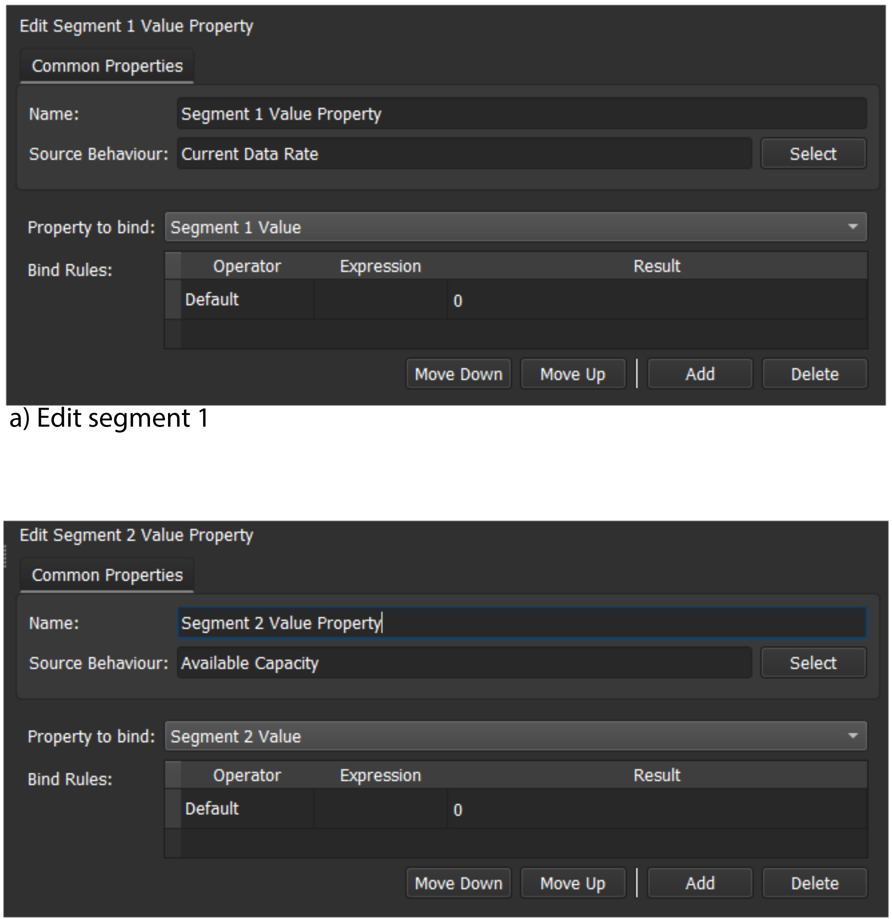


Fig. 3-15: Two New Property Bindings

Now configure each **Property** Binding:

3 Double-click on one of the **Property** Bindings.
The **Edit** Binding screen is shown. See Figure 3-16a.



Edit Segment 1 Value Property

Common Properties

Name: Segment 1 Value Property

Source Behaviour: Current Data Rate

Property to bind: Segment 1 Value

Bind Rules:

Operator	Expression	Result
Default		0

a) Edit segment 1

Edit Segment 2 Value Property

Common Properties

Name: Segment 2 Value Property

Source Behaviour: Available Capacity

Property to bind: Segment 2 Value

Bind Rules:

Operator	Expression	Result
Default		0

b) Edit segment 2

Fig. 3-16: Edit Property Binding Screens

4 In the **Edit** Binding screen, set the following:

Name	Segment 1 Value Property
Source Behaviour	Current Data Rate
Property to Bind	Segment 1 Value

And click **Add**.

5 Click **Close**.

This has bound the 'Current Data Rate' Behaviour value to the Pie Chart widget's 'Segment 1 Value' property.

6 Double-click on the other **Property** Binding and it its Edit Binding screen (Figure 3-16b), set the following:

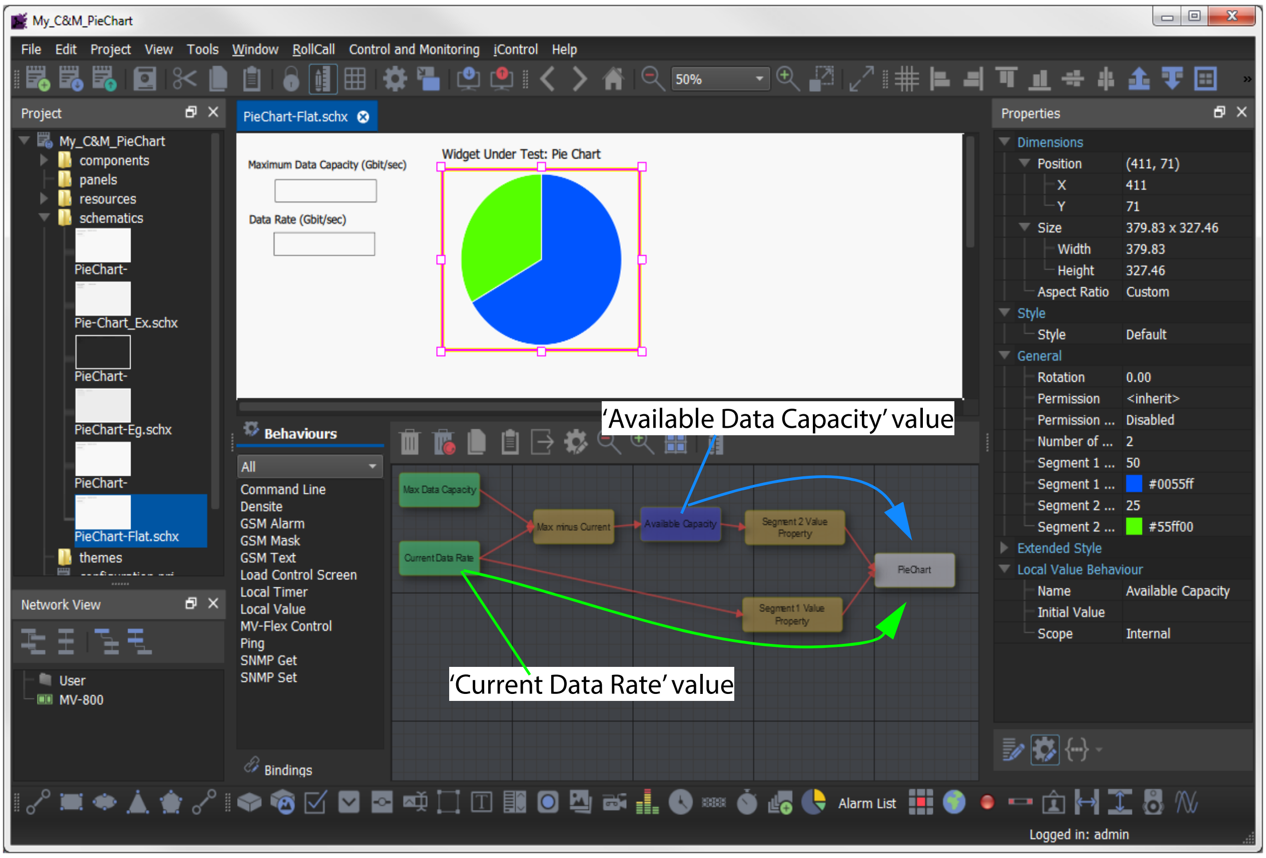
Name	Segment 2 Value Property
Source Behaviour	Available Capacity
Property to Bind	Segment 2 Value

And click **Add**.

7 Click **Close**.

8 Click **Save**.

This has bound the ‘Available Capacity’ Behaviour value to the Pie Chart widget’s ‘Segment 2 Value’ property. Figure 3-17 shows the resulting Pie Chart ‘Behaviour and Binding’ graphical editor.



‘Available Data Capacity’ value

‘Current Data Rate’ value

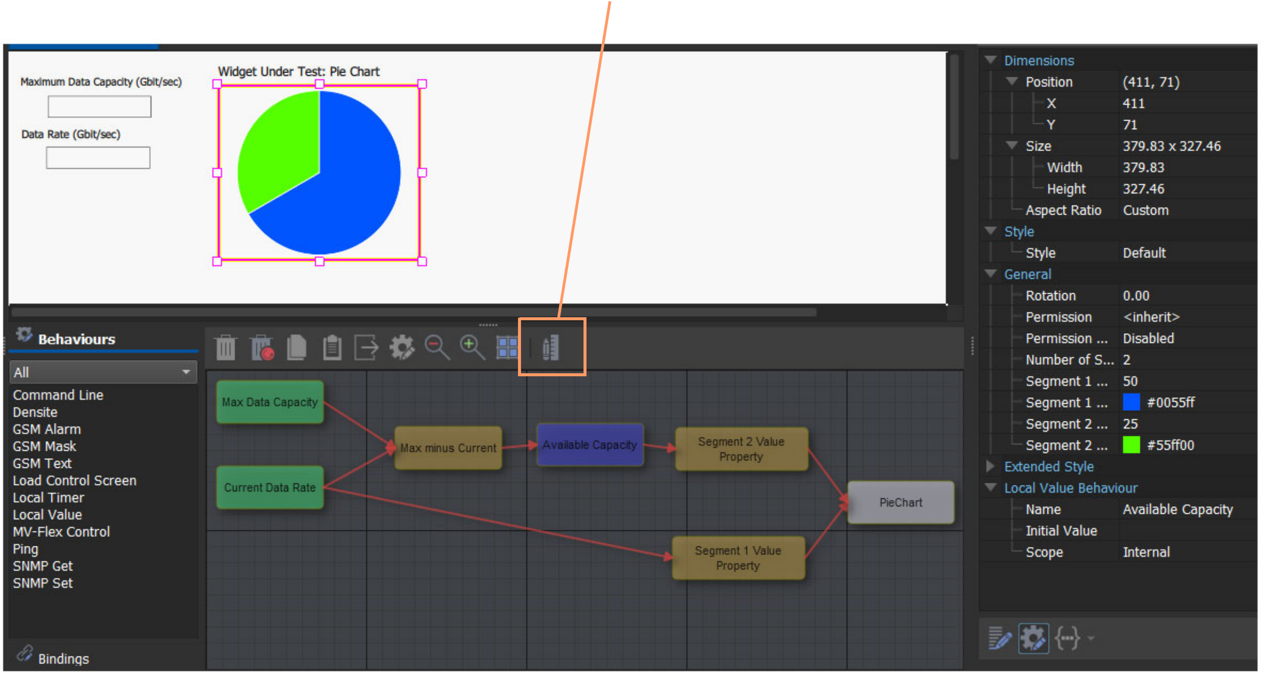
Dimensions	
Position	(411, 71)
X	411
Y	71
Size	379.83 x 327.46
Width	379.83
Height	327.46
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission ...	Disabled
Number of ...	2
Segment 1 ...	50
Segment 1 ...	🟦 #0055ff
Segment 2 ...	25
Segment 2 ...	🟩 #55ff00
Extended Style	
Local Value Behaviour	
Name	Available Capacity
Initial Value	
Scope	Internal

Fig. 3-17: ‘Current Data Rate’ and ‘Available Data Capacity’ Data Values Bound to Pie Chart Widget

Exercise the ‘Widget Under Test’ with Test Bench

Using the test bench screen from Bind ‘Widget Under Test’ to Stimulus Values, on page 53, the **Pie Chart** widget can be exercised:

1 Select the **Pie Chart** widget.
The Behaviours and Bindings for the **Pie Chart** widget are shown. See Figure 3-18.



Test Behaviours and Bindings button

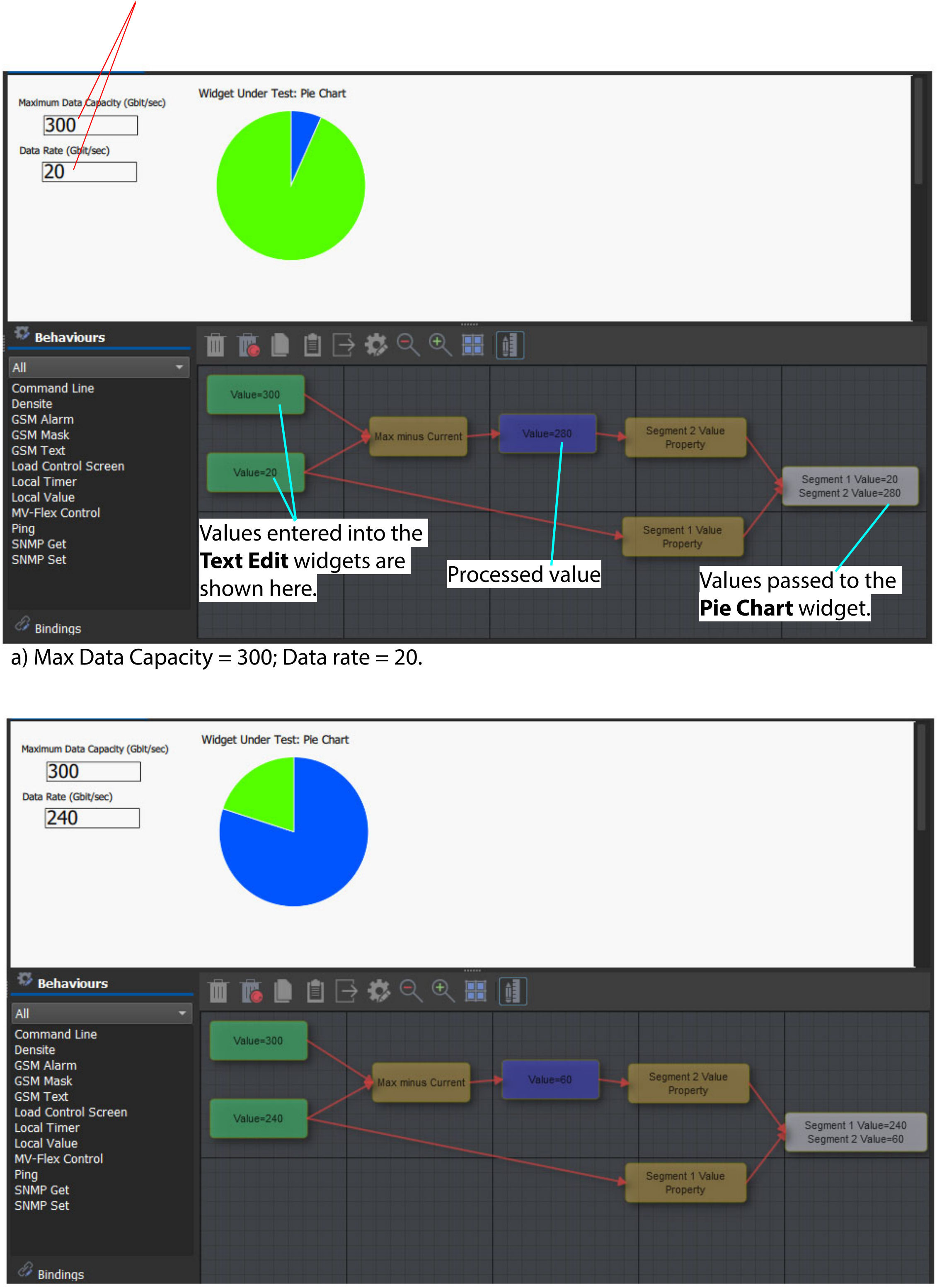
Dimensions	
Position	(411, 71)
X	411
Y	71
Size	379.83 x 327.46
Width	379.83
Height	327.46
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission ...	Disabled
Number of S...	2
Segment 1 ...	50
Segment 1 ...	🟦 #0055ff
Segment 2 ...	25
Segment 2 ...	🟩 #55ff00
Extended Style	
Local Value Behaviour	
Name	Available Capacity
Initial Value	
Scope	Internal

Fig. 3-18: Behaviours and Bindings for the Pie Chart Widget on Test Bench Screen

2 Click the **Test** button.
Orbit enters ‘Test’ mode and allows the widget to be exercised. The the Behaviour and Binding values can be seen in the graphical editor. See Figure 3-19a.

3 Enter values into the **Text Edit** widgets on the test bench screen to exercise the **Pie Chart** widget. See Figure 3-19a and b.

Values entered into the **Text Edit** widgets providing stimulus to the **Pie Chart** widget



Maximum Data Capacity (Gbit/sec): 300

Data Rate (Gbit/sec): 20

Widget Under Test: Pie Chart

Values entered into the **Text Edit** widgets are shown here.

Processed value

Values passed to the **Pie Chart** widget.

a) Max Data Capacity = 300; Data rate = 20.

Maximum Data Capacity (Gbit/sec): 300

Data Rate (Gbit/sec): 240

Widget Under Test: Pie Chart

b) Max Data Capacity = 300; Data rate = 240.

Fig. 3-19: Exercising the Pie Chart Widget, 'Max Data Capacity' and 'Data Rate': a) 300, 20; b) 300, 240.

To finish exercising the **Pie Chart** widget:

4 Click the **Test** button again.
Orbit re-enters 'Design Mode' and allows the user to edit screens etc.

5 Click **Save**.

Extend Widget Functionality with Behaviours and Bindings

The functionality of a widget can be extended with Behaviours and Bindings. The example of Setting Up a Test Bench, on page 46, is extended here.

For our **Pie Chart** widget example, Behaviours and Bindings could be used to:

- work out the data capacity used as a *fraction of the maximum*; and then
- cause a **Pie Chart** segment to change color if data capacity exceeds some threshold amount.

The following are covered in this sub-section:

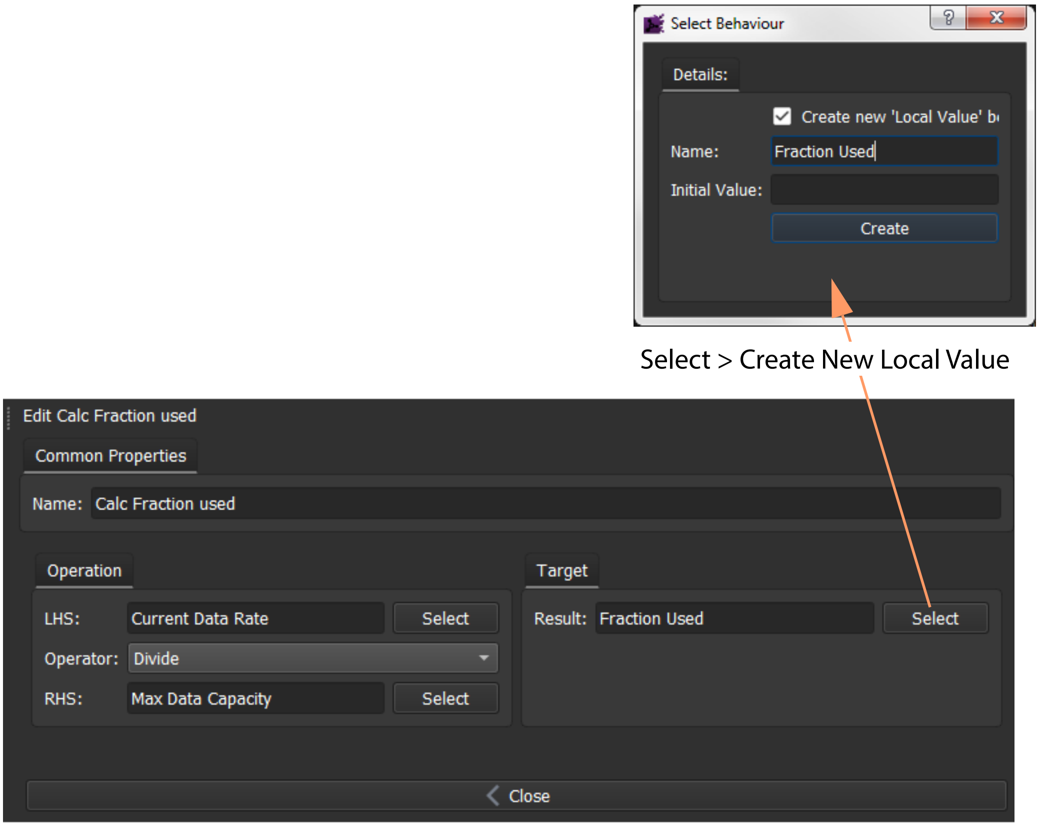
- Adding Extra Functionality, on page 58; and
- Exercising the Extra Functionality, on page 61.

Adding Extra Functionality

Using the test bench screen of Exercise the ‘Widget Under Test’ with Test Bench, on page 56:

1 Select the **Pie Chart** on the test bench screen.
And click the **Edit** Behaviours icon to show the ‘Behaviour and Binding’ graphical editor.

2 Add a **Math** Binding box by clicking on the **Math** item in the Bindings list.
And double-click on this box to show the **Math** Binding edit screen. See Figure 3-20.



Select Behaviour

Details:

☑ Create new 'Local Value' b

Name: Fraction Used

Initial Value:

Select > Create New Local Value

Edit Calc Fraction used

Common Properties

Name: Calc Fraction used

Operation

LHS: Current Data Rate

Operator: Divide

RHS: Max Data Capacity

Target

Result: Fraction Used

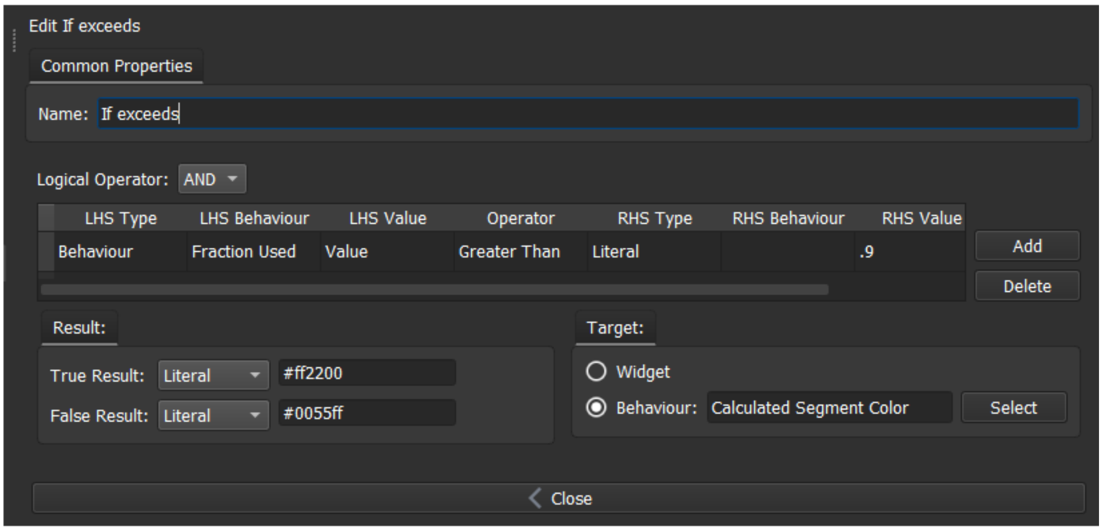
Fig. 3-20: Math Binding Edit Screen

3 Configure the **Math** Binding as shown in Figure 3-20.
This sets up the Binding:

- re-names the **Math** Binding box to ‘Calc Fraction used’;
- configures a ‘Current Data Rate’ divided by the ‘Maximum data rate’ calculation; and
- instructs to place the result in a (new **Local Value**) Behaviour called ‘Fraction Used’.

4 Click **Close**.

5 Add a **Logic** Binding.
And double-click on it to show the **Logic** Binding edit screen. See Figure 3-21.



Edit If exceeds

Common Properties

Name: If exceeds

Logical Operator: AND

LHS Type	LHS Behaviour	LHS Value	Operator	RHS Type	RHS Behaviour	RHS Value
Behaviour	Fraction Used	Value	Greater Than	Literal		.9

Result:

True Result: Literal #ff2200

False Result: Literal #0055ff

Target:

○ Widget

● Behaviour: Calculated Segment Color

Fig. 3-21: Logic Binding Edit Screen

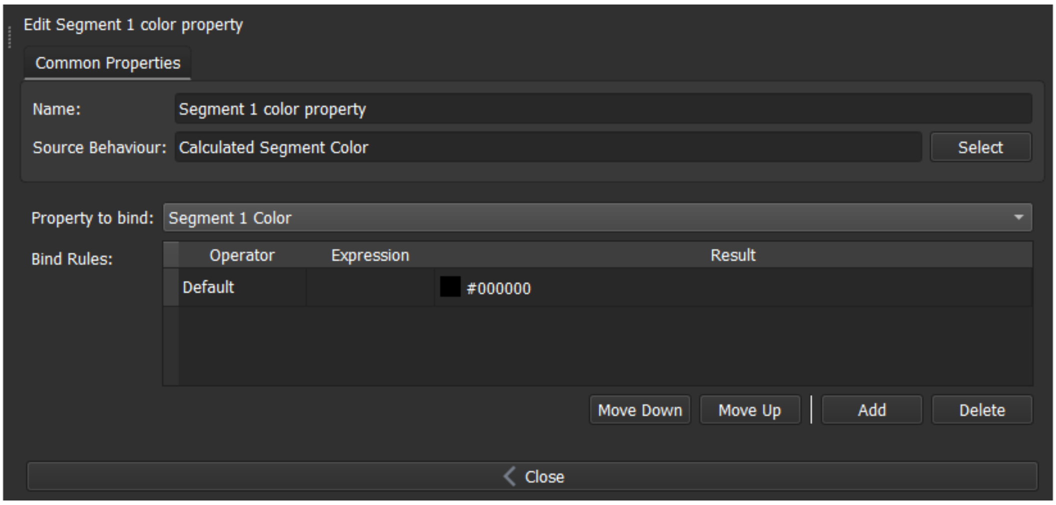
6 Configure the **Logic** Binding as shown in Figure 3-21.
This:

- re-names the Logic Binding box to ‘If exceeds’;
- works out if ‘Fraction Used’ is greater than 0.9;
- generates a result based on the comparison (if fraction > 0.9, result = ‘#ff2200’, else = ‘#0055ff’); and
- places the result in a (new) **Local Value** Behaviour called ‘Calculated Segment Color’.

Note: ‘#ff2200’ and ‘#0055ff’ represent color RGB values.

7 Click **Close**.

8 Add a **Property** Binding.
And double-click on it to show the **Property** Binding edit screen. See Figure 3-22.



Edit Segment 1 color property

Common Properties

Name: Segment 1 color property

Source Behaviour: Calculated Segment Color

Property to bind: Segment 1 Color

Bind Rules:

Operator	Expression	Result
Default		■ #000000

Fig. 3-22: Property Binding Edit Screen

9 Configure the **Property** Binding as shown in Figure 3-22.
This:

- re-names the Property Binding box to 'Segment 1 color property';
- uses a data value from the Behaviour 'Calculated Segment Color';
- binds this value to the (Pie Chart widget's) 'Segment 1 Color' property; and
- adds this to the Binding's bind rules.

10 Click **Close**.

11 Click **Save**.

The resulting 'behind-the-scenes' logic is shown in the 'Behaviour and Bindings' graphical editor. See Figure 3-23.

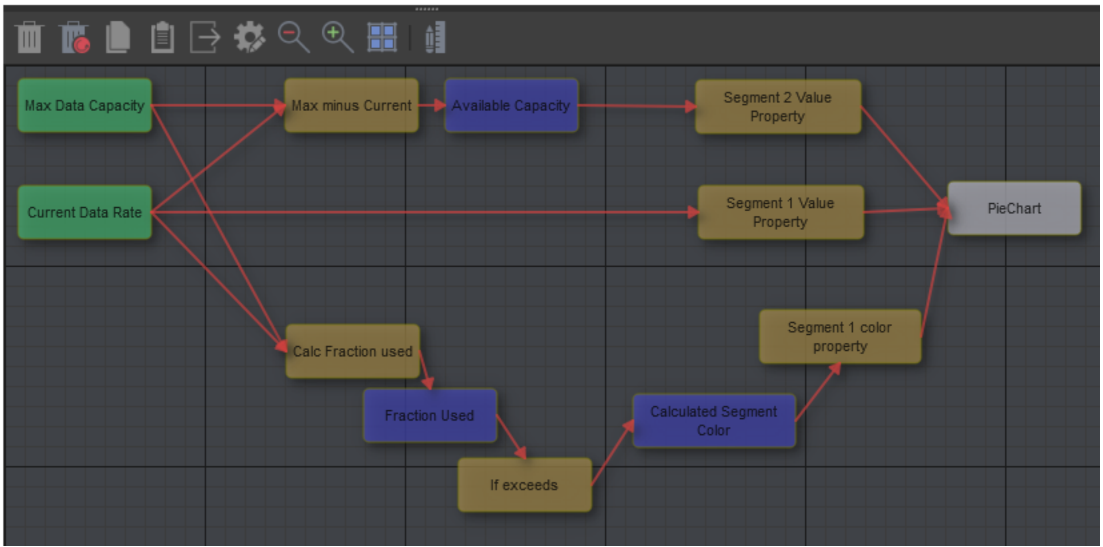


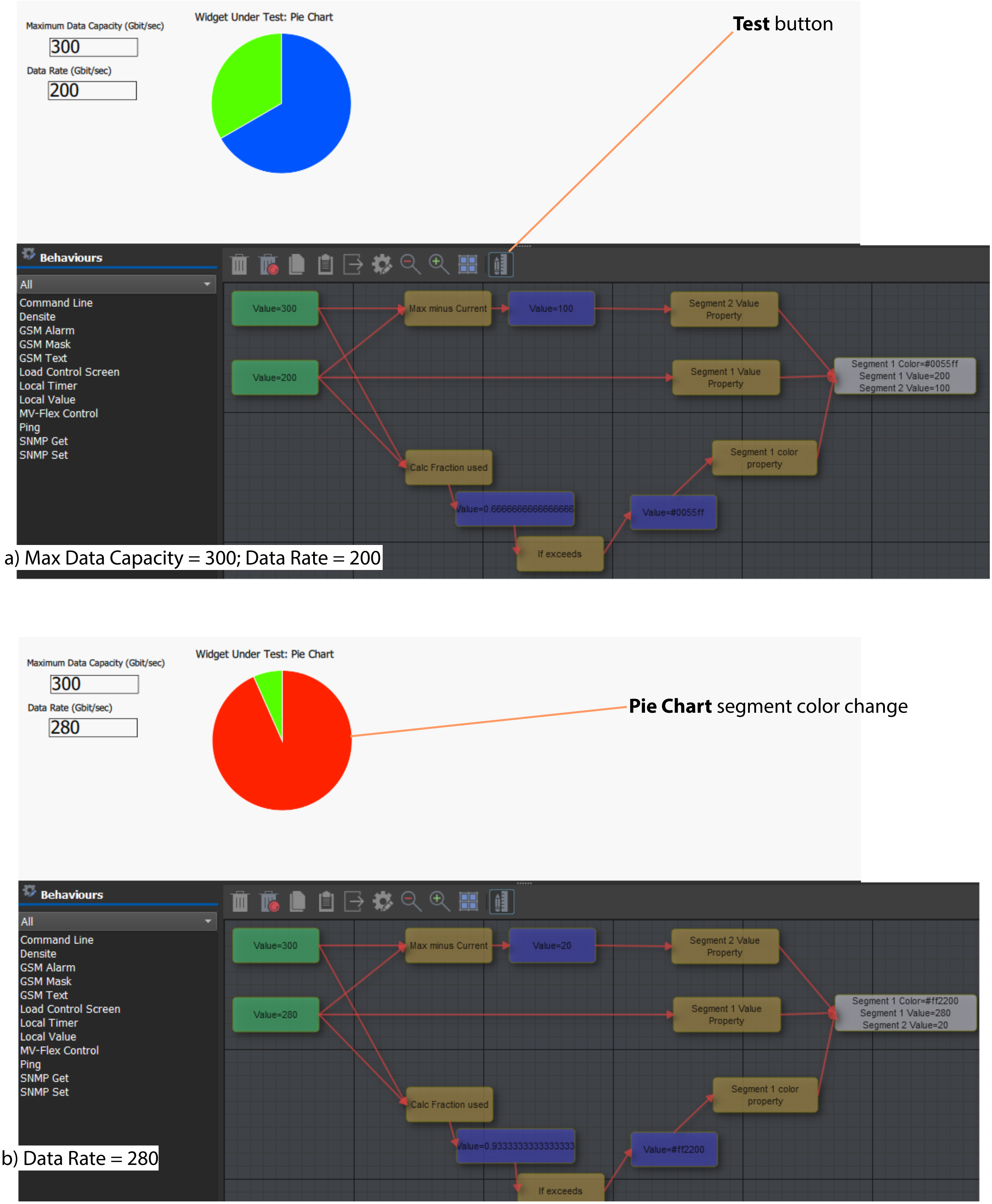
Fig. 3-23: Behaviour and Bindings defining 'behind-the-scenes' Logic

Note: This extended functionality for our **Pie Chart** widget may be moved into a component for easy re-use on other screens.

Exercising the Extra Functionality

1 Click the **Test** button to test the **Pie Chart** widget. See Figure 3-24.

2 Enter values in the **Text Edit** widgets of the test bench screen.



Test button

a) Max Data Capacity = 300; Data Rate = 200

Pie Chart segment color change

b) Data Rate = 280

Fig. 3-24: Exercising Pie Chart Segment Color Change

Monitoring by Exception

The **Monitoring by Exception (MbyE)** widget is used to summarily highlight system errors to a user, easing the task of monitoring an abundance of status information. The **MbyE** widget can be configured to only show certain state overview information and in an ordered way.

The **MbyE** widget is configured with one or more 'MbyE overviews'. The widget can handle a mixture of the following overview modes:

- **Schematic MbyE** monitoring.

  The **Schematic MbyE** overview monitors the aggregate state of a MapView screen. Only the state of alarms *on* the screen are monitored: The state of any further 'linked-to' screens are *not* monitored. (A screen may contain links to other screens - allowing a user to navigate down to further detail - but these links do not contribute to its 'MbyE overview' state.)

  Each **Schematic MbyE** overview is typically configured with a screen which contains alarms for all Log Fields and devices to be monitored. All devices and log fields for a particular system function may be assembled on a screen, for example, for a channel overview application.

  The overview is configured with a unique, user-defined RollCall address, and with a link to the screen containing all alarms etc to monitor.

- **Device MbyE** monitoring.

  The **Device MbyE** overview monitors the state of an individual device. The overview is configured with the RollCall address of the device being monitored. A link to a screen may still be added but it is not part of the 'Device MbyE' monitoring - for example, such a link may just be used to obtain information about a device.

Monitoring by Exception (MbyE) Widget

Figure 3-25 shows the **Monitoring by Exception** (**MbyE**) widget on-screen. It displays icons for each Orbit MapView screen it is monitoring.

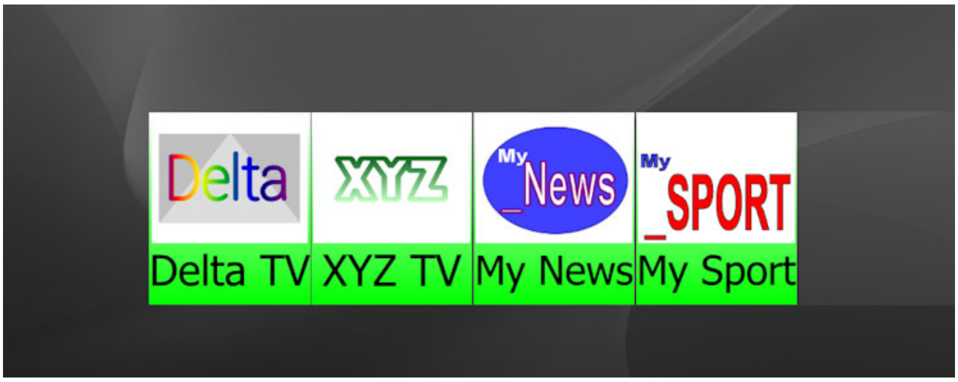
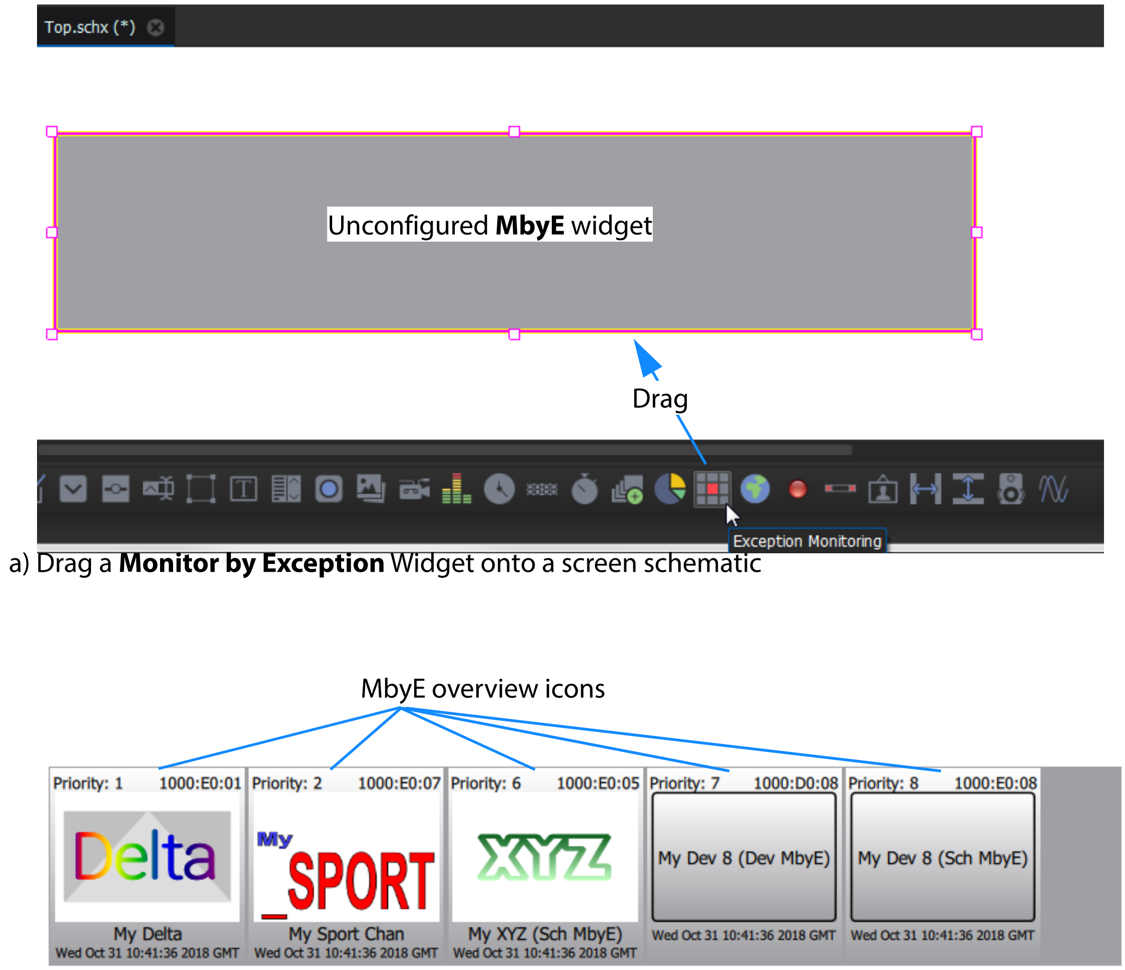


Fig. 3-25: Monitoring by Exception Widget

Figure 3-26 shows the widget after it has been dragged onto a screen and then configured, before it is running in Orbit.

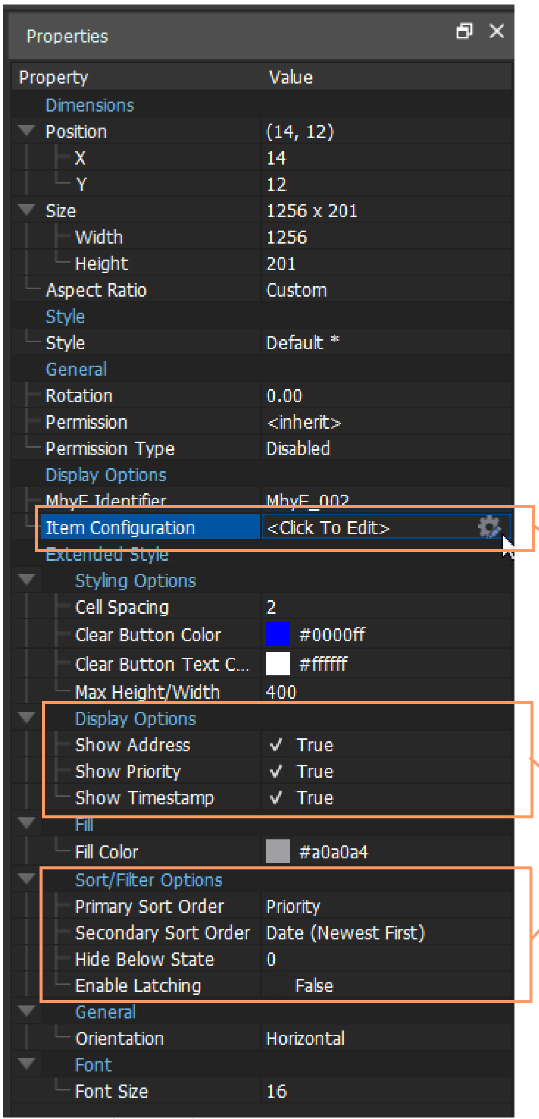


a) Drag a **Monitor by Exception** Widget onto a screen schematic

b) Example Configured **Monitor by Exception** Widget

Fig. 3-26: Monitoring by Exception Widget:
a) After being dragged on screen.
b) After configuration (but before being run in Orbit)

Configuration of the **MbyE** widget is done via the **Properties** box. See Figure 3-27.



Properties

Property	Value
Dimensions	
Position	(14, 12)
X	14
Y	12
Size	1256 x 201
Width	1256
Height	201
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Display Options	
MbyE Identifier	MbyE_002
Item Configuration	<Click To Edit>
Extended Style	
Styling Options	
Cell Spacing	2
Clear Button Color	#0000ff
Clear Button Text C...	#ffffff
Max Height/Width	400
Display Options	
Show Address	✓ True
Show Priority	✓ True
Show Timestamp	✓ True
Fill	
Fill Color	#a0a0a4
Sort/Filter Options	
Primary Sort Order	Priority
Secondary Sort Order	Date (Newest First)
Hide Below State	0
Enable Latching	False
General	
Orientation	Horizontal
Font	
Font Size	16

Configure screens to be monitored

Set widget options for display and display order

Fig. 3-27: MbyE Widget Properties Box

Table 3-1: MbyE Widget Configuration Properties

Configuration Property	Description
Item Configuration	Click the ⚙ icon to show the configuration dialog. See Configuration Dialog of the MbyE Widget, on page 66.
Display Options:	See Figure 3-28, MbyE Overview Icon.
Show Address	Check box. Select to show RollCall Address on widget.
Show Priority	Check box. Select to show priority setting on widget.
Show Timestamp	Check box. Select to show timestamp on widget.

Table 3-1: MbyE Widget Configuration Properties (continued)

Configuration Property	Description
Sort/Filter Options:	
Primary Sort Order	Drop down box. Select the order in which icons are shown on the **MbyE** widget. Option: • **Priority** - In priority order, lowest priority shown left-most. Priority is set in the Configuration Dialog of the MbyE Widget, on page 66. • **Date (Newest First)** - Display newest alarm condition left-most. • **Date (Oldest First)** - Display oldest alarm condition left-most. • **State** - 'State' value of the alarm condition order. • **Address** - Address order.
Secondary Sort Order	Drop down box. Select secondary icon ordering. Options are the same as the Primary Sort Order.
Hide Below State	Text box. Enter a numeric state value. (0 to 100, see State Value 0 to 100, on page 17.) Select the threshold alarm state value below which icons are not shown on the widget. (Alarm state: 0 = masked; 1 = OK; 50 = Warning; 100 = Error) For example: • =50, shows unacknowledged 'Warnings' and all 'Failures'. • =0, show all icons.
Enable Latching	Check box. Select to latch the alarm shown in the widget.

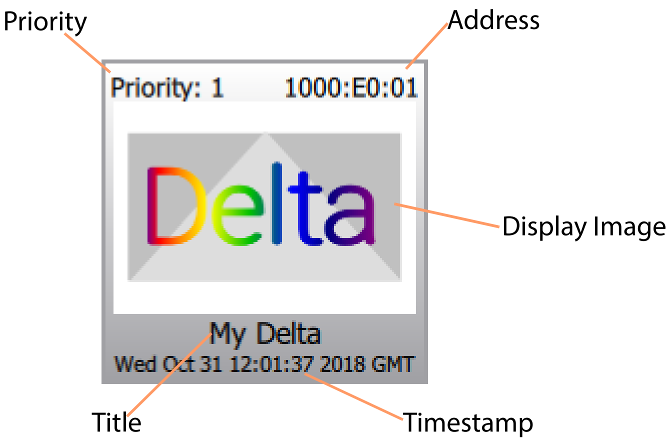
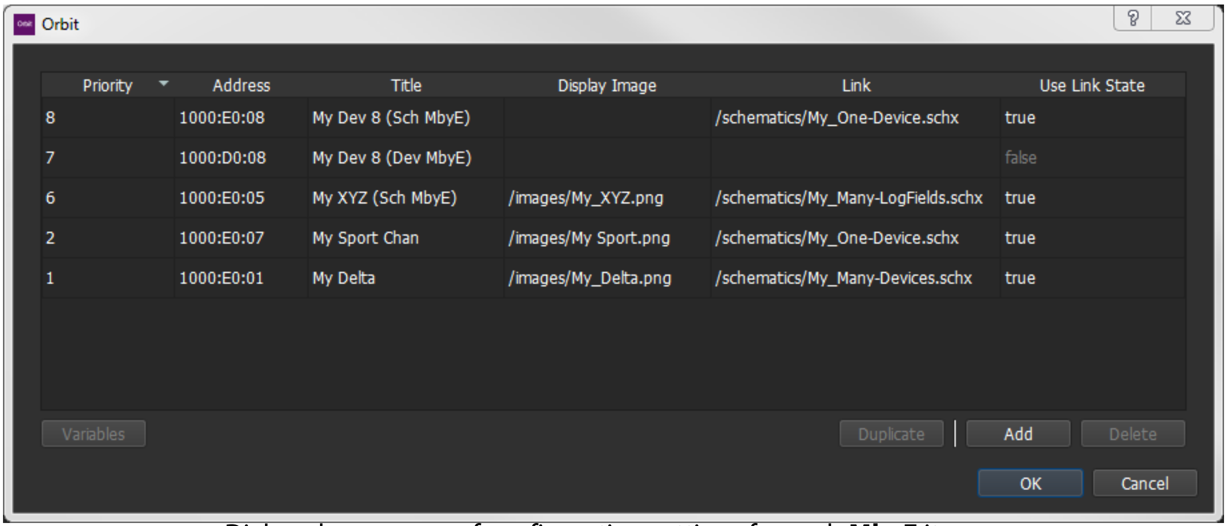


Fig. 3-28: MbyE Overview Icon

Configuration Dialog of the MbyE Widget

Open the **Monitor by Exception** widget configuration dialog by clicking the ⚙ icon for the 'Item Configuration' property. See Figure 3-29.
The configuration dialog shows a row of settings for each **MbyE** item.



Priority	Address	Title	Display Image	Link	Use Link State
8	1000:E0:08	My Dev 8 (Sch MbyE)		/schematics/My_One-Device.schx	true
7	1000:D0:08	My Dev 8 (Dev MbyE)			false
6	1000:E0:05	My XYZ (Sch MbyE)	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2	1000:E0:07	My Sport Chan	/images/My Sport.png	/schematics/My_One-Device.schx	true
1	1000:E0:01	My Delta	/images/My_Delta.png	/schematics/My_Many-Devices.schx	true

Dialog shows a row of configuration settings for each **MbyE** item.

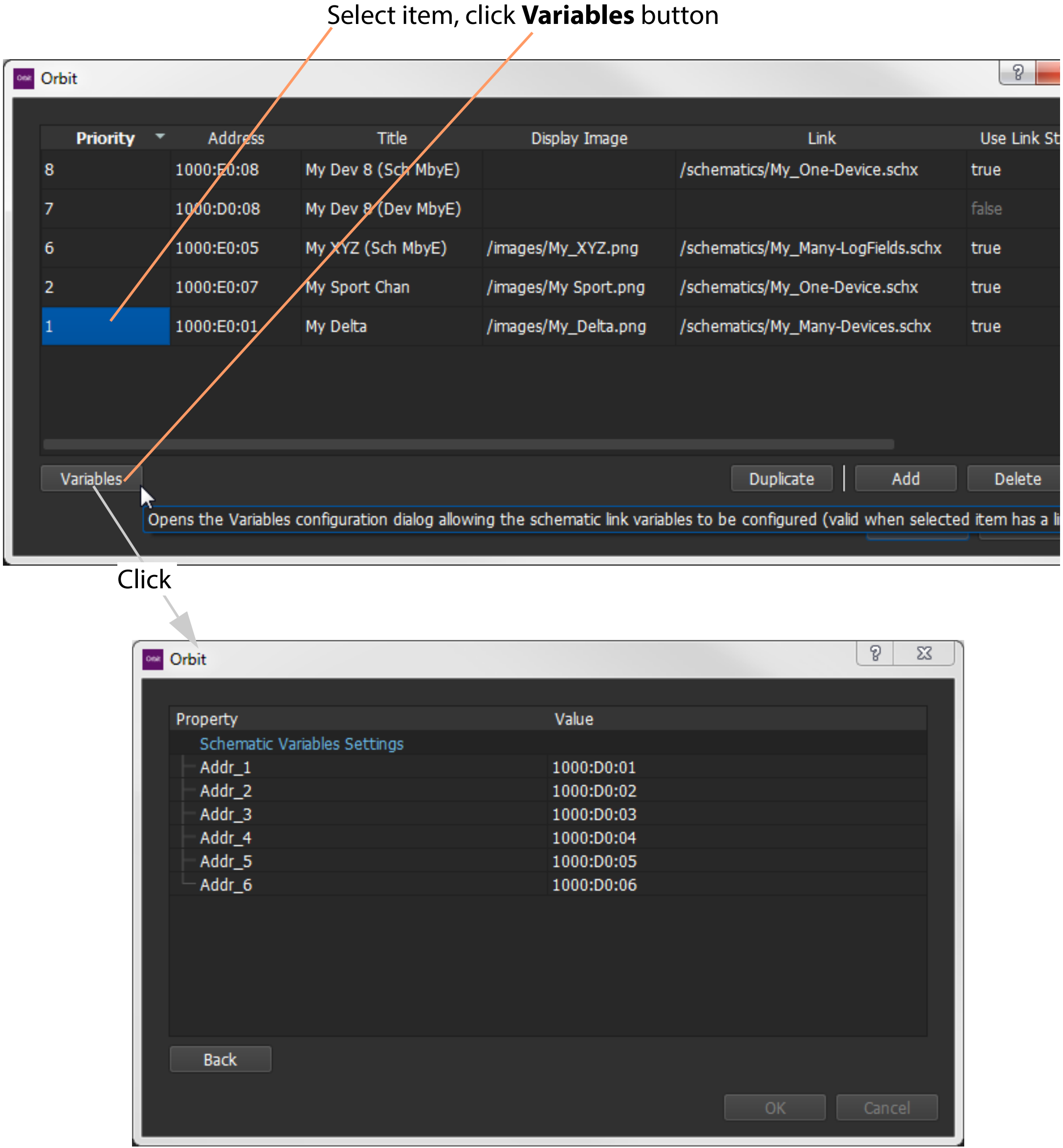
Fig. 3-29: Monitoring by Exception Widget Configuration Dialog

Table 3-2: MbyE Widget Configuration Properties

Configuration Item	Description
Priority	Set a priority number. Lower numbers are displayed left-most in the widget when the Orbit project in running.
Address	RollCall address to be used for the **Schematic MbyE** overview.
Title	Name to appear on the **MbyE** overview icon.
Display Image	Image (in the Orbit MapView project) to show on the **MbyE** overview icon.
Link	Screen (in the Orbit MapView project) to go to when **MbyE** overview icon is clicked.

Table 3-2: MbyE Widget Configuration Properties (continued)

Configuration Item	Description
Use Link State	Check box. The **MbyE** widget is on a screen and the overall 'Link State' of the screen has contributions from all of the screen widgets etc., including from each **MbyE** item. • **True** - **MbyE** item is used in overall screen 'State'. (I.e. the 'Link State' of the 'Linked-to' screen/device is used.) • **False** - **MbyE** item is *not used* in the overall screen 'State'. (I.e. the 'Link State' of the 'Linked-to' screen/device is not used.) **Note:** The **Use Link State** setting can be set to **False** if the **MbyE** widget is on some banner tool bar which should not contribute to the 'Link State' of the screen.
Variables	Button. With an MbyE item selected, click the **Variables** button to display a **Schematic Variable Settings** dialog. See Figure 3-30. In the dialog, set values of schematic variables for the 'linked-to' screen. Typically, the variables include the addresses of all devices being monitored by the **MbyE** overview.



Select item, click **Variables** button

Click

Priority	Address	Title	Display Image	Link	Use Link St
8	1000:E0:08	My Dev 8 (Sch MbyE)		/schematics/My_One-Device.schx	true
7	1000:D0:08	My Dev 8 (Dev MbyE)			false
6	1000:E0:05	My XYZ (Sch MbyE)	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2	1000:E0:07	My Sport Chan	/images/My Sport.png	/schematics/My_One-Device.schx	true
1	1000:E0:01	My Delta	/images/My_Delta.png	/schematics/My_Many-Devices.schx	true

Opens the Variables configuration dialog allowing the schematic link variables to be configured (valid when selected item has a li

Property	Value
Schematic Variables Settings	
Addr_1	1000:D0:01
Addr_2	1000:D0:02
Addr_3	1000:D0:03
Addr_4	1000:D0:04
Addr_5	1000:D0:05
Addr_6	1000:D0:06

Fig. 3-30: Schematic Variables Settings Dialog

Monitoring by Exception Example

See Example - Monitoring by Exception, on page 291.

Auto-Fill Property Values

Orbit contains an **Auto-Fill** facility for use on graphical schematics to set up or configure many widget or component properties at the same time.

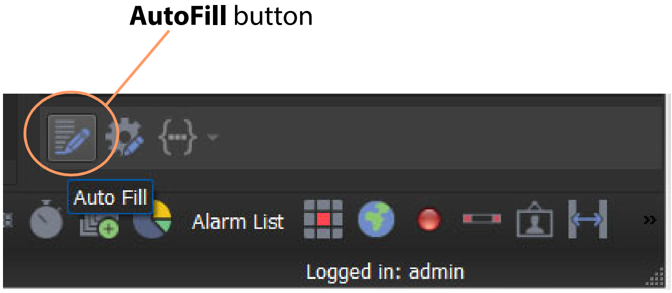


Fig. 3-31:

Auto-Fill can:

- Set properties to string values in some numerical sequence.
  For example, “SRC 01”, “SRC 02”, “SRC03”, etc.
- Range over variables whose names are in some numerical sequence and set the variables’ values.
  For example, “Caption 1”, “Caption 2”, etc.

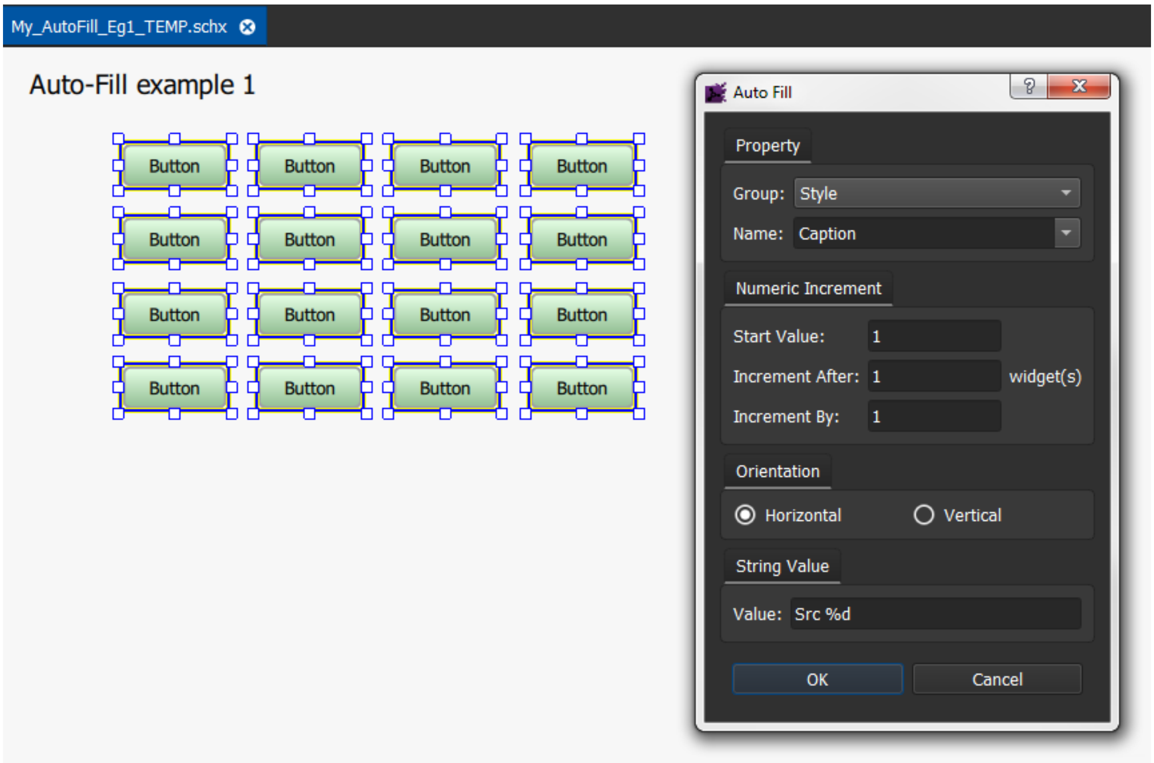
Auto-Fill Example 1: Set a Single Property across Multiple Widgets

In this example, **Auto-Fill** is used on a single property of multiple widgets. 16 button widgets arranged in a 4 x 4 grid on an Orbit screen are used. See Figure 3-32. This example will re-label the ‘Caption’ property of all the buttons to be “Src 1”, “Src 2”, ..., “Src 16”.

To Re-label the Buttons using Auto-Fill

1 Select all the buttons on the screen.

2 Click the **Auto-Fill** icon to show the Auto-Fill dialog. See Figure 3-32.



Property

Group: Style

Name: Caption

Numeric Increment

Start Value: 1

Increment After: 1 widget(s)

Increment By: 1

Orientation

● Horizontal ○ Vertical

String Value

Value: Src %d

Fig. 3-32: Auto-Fill Dialog

3 In the **Auto-Fill** dialog:

  a Set property **Name** to 'Caption'.

  b Leave the **Numeric Increment** values all set to '1'.

  c Leave **Orientation** set to 'Horizontal'.

  d In the **String Value** text box, enter: Src %d

4 Click **OK**.

The buttons are then all re-labeled left-to-right, top-to-bottom: A number sequence starting from '1' and incrementing by '1' is used. The result is shown in Figure 3-33.

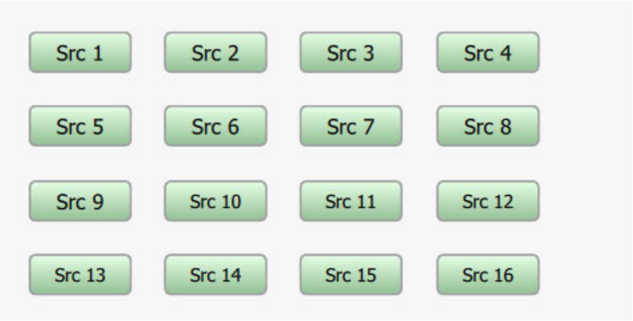
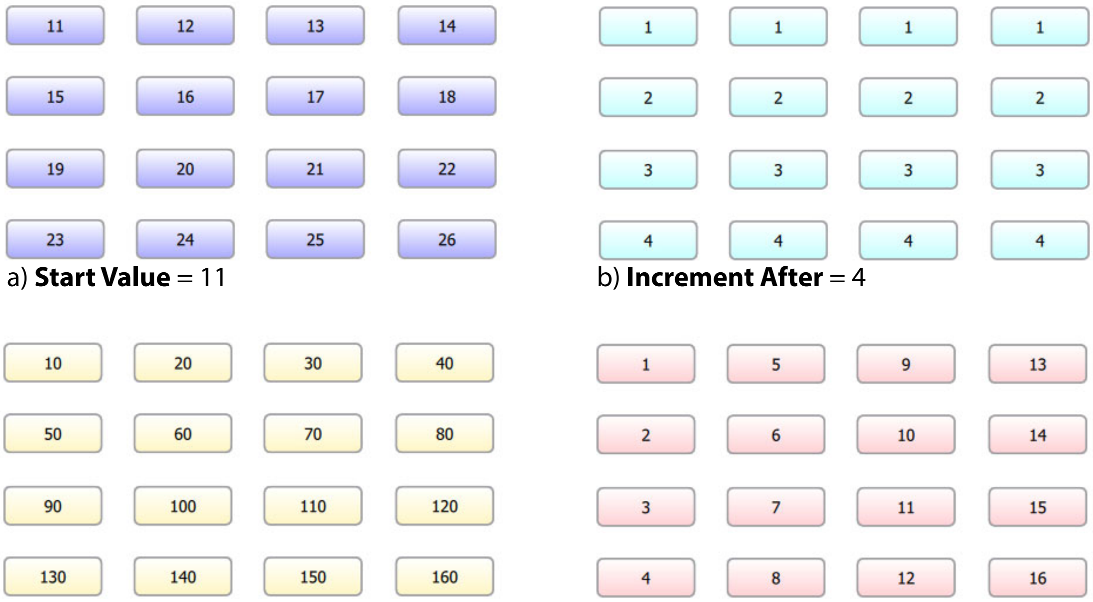


Fig. 3-33: Re-Labeled Buttons

Numeric Increments and Orientation

Setting the **Numeric Increment** values (**Start Value**, **Increment After**, and **Increment By**) affects the numbering sequence used in **Auto-Fill**. See Figure 3-34a, b and c.

Setting **Orientation** to 'Vertical' will re-label top-to-bottom first. See Figure 3-34d.



a) **Start Value** = 11

b) **Increment After** = 4

c) **Increment By** = 10 and Start Value = 10

d) **Orientation** = 'Vertical'

Fig. 3-34: Other Auto-Filled Button Labels

Format Syntax of a String Value

The "%d" item entered in the **String Value** text box in the above example is a special format syntax which enables an incrementing value to be used in the Auto-Fill operation. Table 3-3 lists other format syntaxes that may be used.

Table 3-3: Format Syntax

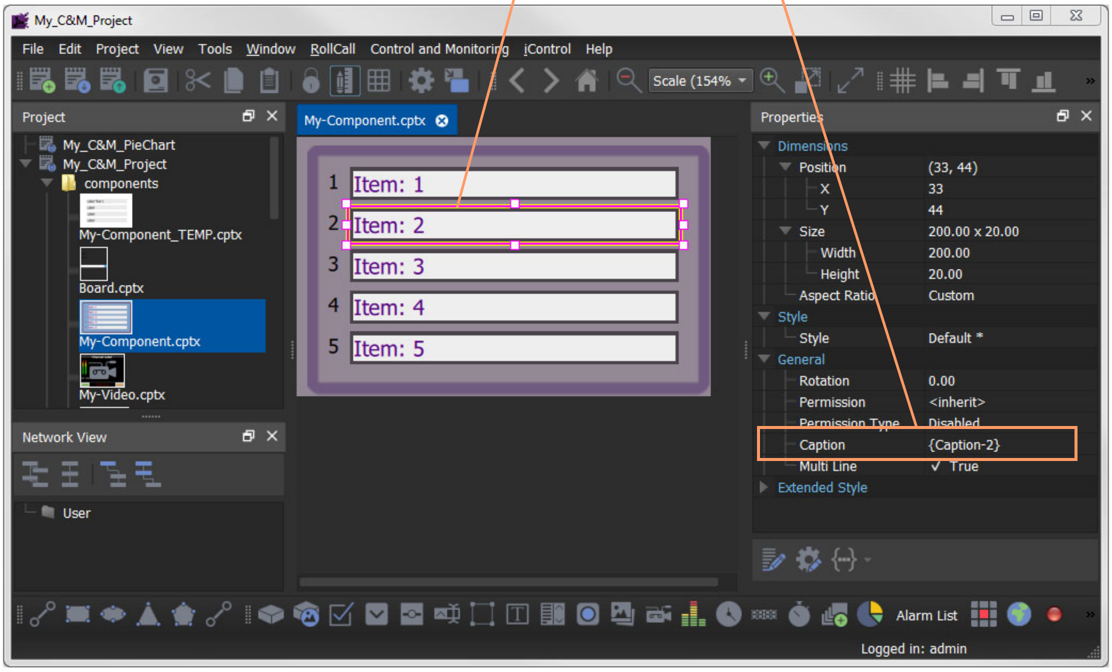
Format Syntax	Example Incrementing Sequence
Decimal:	
%d	1, 2, 3, 4, 5, ... 9, 10, 11, 12, 13,...
%02d	01, 02, 03,...
%03d	001, 002, 003,...
Hexadecimal:	
%X	1, 2, 3,... 9, A, B, C, D, E, F, 10, 11, 12,...
%x	1, 2, 3,... 9, a, b, c, d, e, f, 10, 11, 12,...
%02X	01, 02, 03,... 0E, 0F, 10, 11, 12,...

Auto-Fill Example 2: Set Multiple Variables on a Single Widget/Component

The **Auto-Fill** facility is normally used on a single property item from multiple widgets but it also works on multiple items from a single widget or component. A common use-case is a component with multiple, similarly-named variables - **Auto-Fill** can set these up.

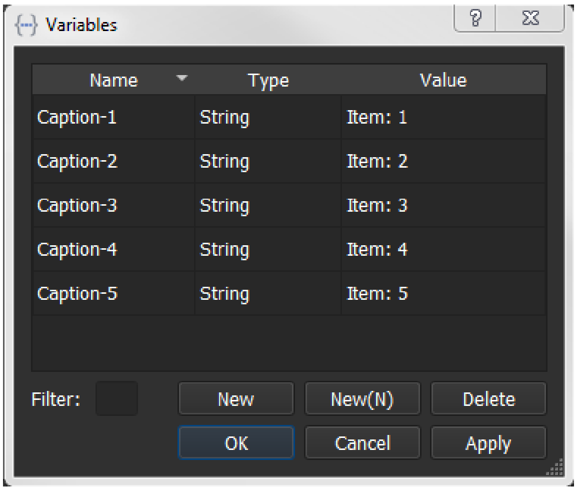
This example uses a component containing some text labels whose 'Caption' properties have been linked to component variables. The component is shown in Figure 3-35 and all the component variables are shown in the Variables dialog of Figure 3-36. (Right-click in the component background with nothing selected and select 'Variables...' in the context menu.)

Label widget 2:
'Caption' property is set
to be value of component variable, {Caption-2}.



Dimensions	
Position	(33, 44)
X	33
Y	44
Size	200.00 x 20.00
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{Caption-2}
Multi Line	✓ True
Extended Style	

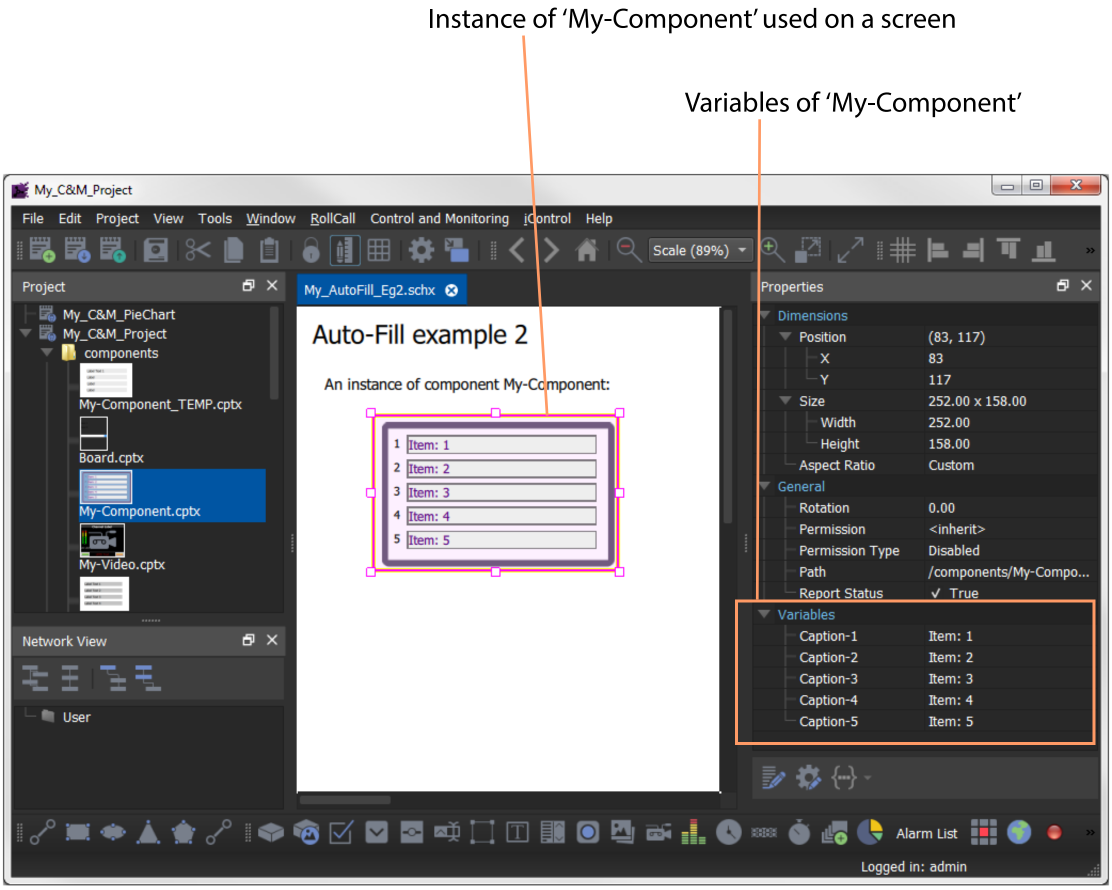
Fig. 3-35: Example Component 'My-Component'



Name	Type	Value
Caption-1	String	Item: 1
Caption-2	String	Item: 2
Caption-3	String	Item: 3
Caption-4	String	Item: 4
Caption-5	String	Item: 5

Fig. 3-36: Example Component Variables

Once a component is made, it may be used on screen(s) one or more times. Figure 3-37 shows a newly-placed component on a screen and the component variables that are available.



Auto-Fill example 2

An instance of component My-Component:

1 Item: 1
2 Item: 2
3 Item: 3
4 Item: 4
5 Item: 5

Properties

Dimensions	
Position	(83, 117)
X	83
Y	117
Size	252.00 x 158.00
Width	252.00
Height	158.00
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/My-Compo...
Report Status	✓ True
Variables	
Caption-1	Item: 1
Caption-2	Item: 2
Caption-3	Item: 3
Caption-4	Item: 4
Caption-5	Item: 5

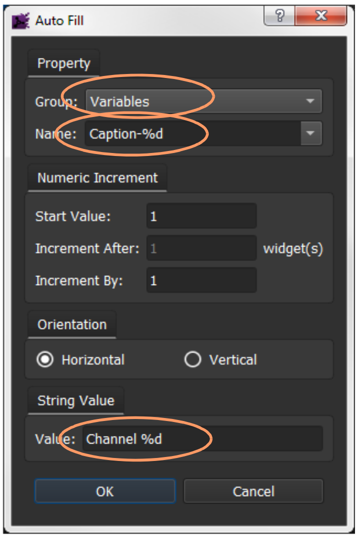
Fig. 3-37: My-Component on a Screen Schematic

To Set Component Variables

1 Select the component on the screen schematic.

2 Click the **Auto-Fill** icon to show the **Auto-Fill** dialog.

3 Select the 'Group' setting to be 'Variables' and set other dialog values to those shown in Figure 3-38.



Group: Variables

Name: Caption-%d

Start Value: 1

Increment After: 1 widget(s)

Increment By: 1

● Horizontal ○ Vertical

Value: Channel %d

Fig. 3-38: Auto-Fill Dialog Settings

Set the following in the dialog:

Table 3-4.

Property:	
Group	Variables
Name	Caption-%d
String Value:	
Value	Channel %d

4 Click **OK**.

The variables of the component are filled out with ‘Channel 1’, ‘Channel 2’, etc.
And widget properties linked to component variables reflect these component variable values. See Figure 3-39.

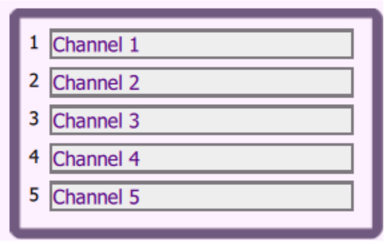
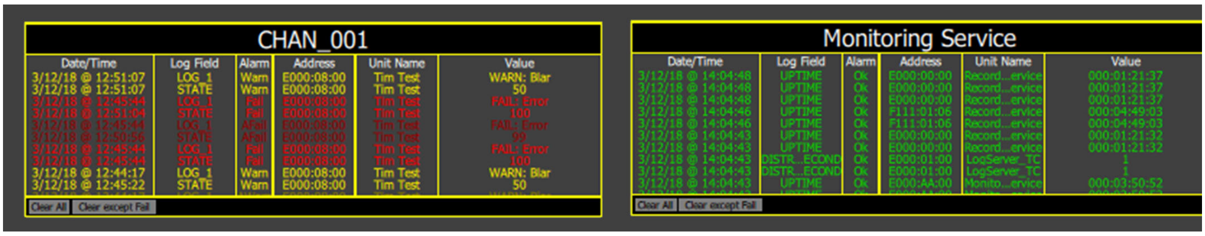


Fig. 3-39: ‘My-Component’ Changed via Auto-Fill

Viewing Log Messages (Alarm List Widget)

Introduction

Log messages can be viewed on an Orbit MapView control and monitoring screen, or on a multiviewer video wall. Live messages can be viewed via an Orbit MapView Monitoring Service, or logged messages can be viewed via an Orbit MapView Recording Service. Displayed messages can be simply filtered at the widget.



CHAN_001

Date/Time	Log Field	Alarm	Address	Unit Name	Value
3/12/18 @ 12:51:07	LOG_1	Warn	E000:08:00	Tim Test	WARN: Blar
3/12/18 @ 12:51:07	STATE	Warn	E000:08:00	Tim Test	50
3/12/18 @ 12:45:44	LOG_1	Fail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:51:04	STATE	Fail	E000:08:00	Tim Test	100
3/12/18 @ 12:45:44	LOG_1	AFail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:50:56	STATE	AFail	E000:08:00	Tim Test	99
3/12/18 @ 12:45:44	LOG_1	Fail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:45:44	STATE	Fail	E000:08:00	Tim Test	100
3/12/18 @ 12:44:17	LOG_1	Warn	E000:08:00	Tim Test	WARN: Blar
3/12/18 @ 12:45:22	STATE	Warn	E000:08:00	Tim Test	50

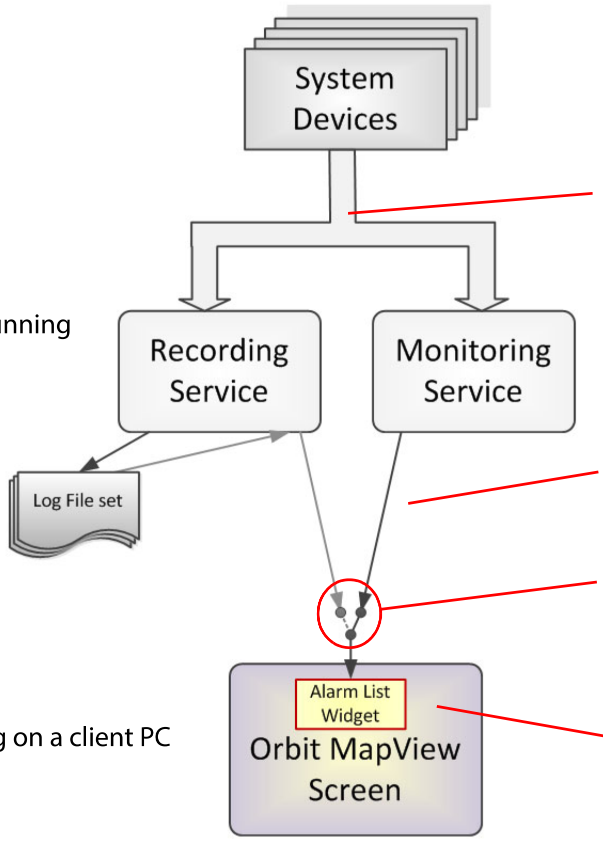
Monitoring Service

Date/Time	Log Field	Alarm	Address	Unit Name	Value
3/12/18 @ 14:04:48	UPTIME	Ok	E000:00:00	Record...ervice	000:01:21:37
3/12/18 @ 14:04:48	UPTIME	Ok	E000:00:00	Record...ervice	000:01:21:37
3/12/18 @ 14:04:48	UPTIME	Ok	E000:00:00	Record...ervice	000:01:21:37
3/12/18 @ 14:04:46	UPTIME	Ok	F111:01:06	Record...ervice	000:04:49:03
3/12/18 @ 14:04:46	UPTIME	Ok	F111:01:06	Record...ervice	000:04:49:03
3/12/18 @ 14:04:43	UPTIME	Ok	E000:00:00	Record...ervice	000:01:21:32
3/12/18 @ 14:04:43	UPTIME	Ok	E000:00:00	Record...ervice	000:01:21:32
3/12/18 @ 14:04:43	DISTR...ECOND	Ok	E000:01:00	LogServer_TC	1
3/12/18 @ 14:04:43	DISTR...ECOND	Ok	E000:01:00	LogServer_TC	1
3/12/18 @ 14:04:43	UPTIME	Ok	E000:AA:00	Monito...ervice	000:03:50:52

Fig. 3-40: Log Messages Displayed on MapView Screen

Note: PCs running Orbit MapView Client(s) or multiviewer device(s) must be on the *same network* as the Orbit MapView services.

Note: Orbit MapView MapView projects or Orbit MapView Multiviewer projects must be on the *same RollCall+ domain* as the Orbit MapView services.



Log messages on RollCall+ network

Orbit services running on server PC

Log messages from service for widget

Select source of log messages for widget with the widget 's 'Connect to' property.

Running on a client PC

MapView project with an Alarm List widget

Fig. 3-41: Log Messages for Alarm List Widget

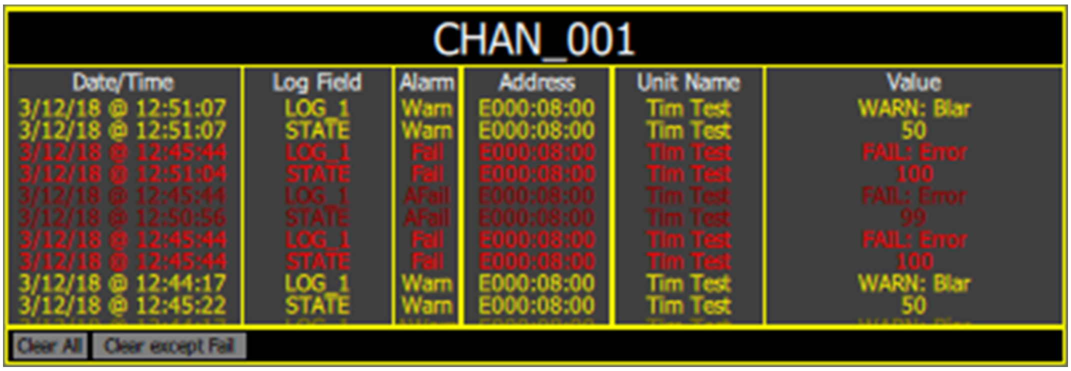
Alarm List Widget

The **Alarm List** widget is a table-list widget which displays a historical list of log messages, including alarms. It can be placed onto Orbit MapView screens and onto MV-8 Series Multiviewer video walls with Orbit projects.

The widget may be connected to:

- an Orbit MapView Monitoring Service (for "live" log data); or to
- an Orbit MapView Recording Service's log files (for historical logged data).

> IMPORTANT:
> The **Publish to Alarm Widget** setting must be enabled within the Log File configuration of the Recording Service for the Alarm List widget to connect correctly to a Recording Service.



CHAN_001

Date/Time	Log Field	Alarm	Address	Unit Name	Value
3/12/18 @ 12:51:07	LOG_1	Warn	E000:08:00	Tim Test	WARN: Blar
3/12/18 @ 12:51:07	STATE	Warn	E000:08:00	Tim Test	50
3/12/18 @ 12:45:44	LOG_1	Fail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:51:04	STATE	Fail	E000:08:00	Tim Test	100
3/12/18 @ 12:45:44	LOG_1	AFail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:50:56	STATE	AFail	E000:08:00	Tim Test	99
3/12/18 @ 12:45:44	LOG_1	Fail	E000:08:00	Tim Test	FAIL: Error
3/12/18 @ 12:45:44	STATE	Fail	E000:08:00	Tim Test	100
3/12/18 @ 12:44:17	LOG_1	Warn	E000:08:00	Tim Test	WARN: Blar
3/12/18 @ 12:45:22	STATE	Warn	E000:08:00	Tim Test	50

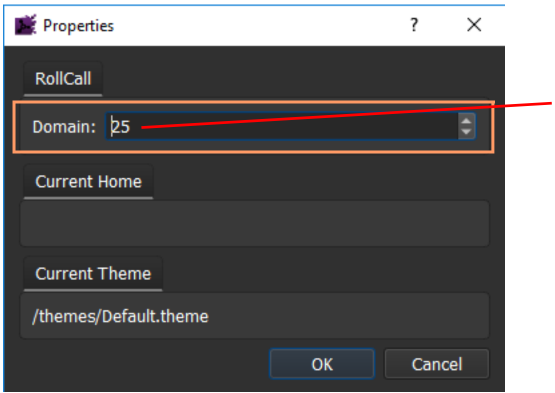
Fig. 3-42: Alarm List Widget

Using an Alarm List Widget in Orbit MapView

1 Run the Orbit MapView Client software and open/create a 'control and monitoring' (C&M) project.

Set up RollCall+ Domain:

2 Click 'Control and Monitoring' > Properties in the main menu.
The Properties dialog is shown. See Figure 3-43.



Domain: 25

/themes/Default.theme

RollCall+ domain number

Fig. 3-43: Control and Monitoring > Properties Dialog

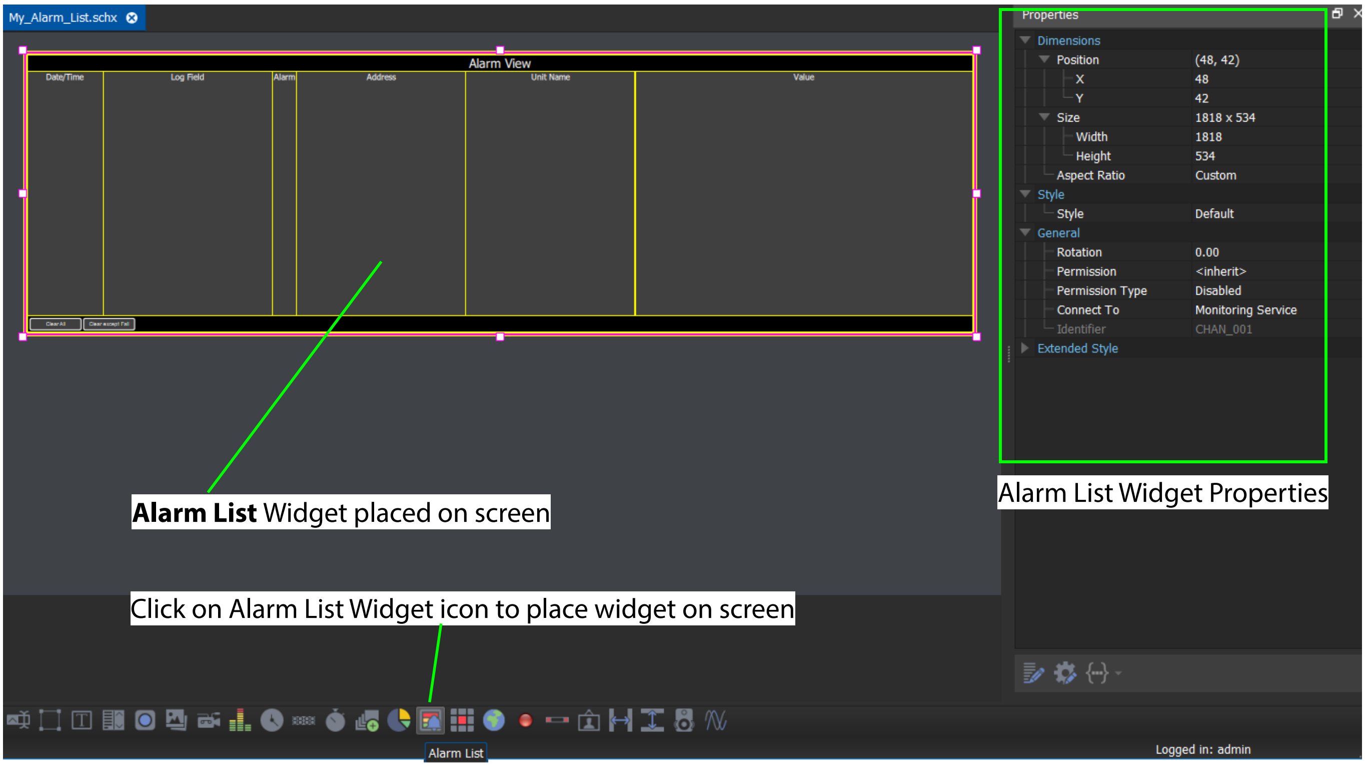
3 Enter the RollCall+ domain number.
Note: RollCall+ domain must be the same as the Orbit Services being connected to.

4 Click **OK**.

Add an Alarm List widget:

5 Open an Orbit MapView MapView control and monitoring project and enter 'Design Mode'.

6 Place an **Alarm List** widget onto a MapView screen. See Figure 3-44.



Properties

- Dimensions
  - Position (48, 42)
    - X 48
    - Y 42
  - Size 1818 x 534
    - Width 1818
    - Height 534
  - Aspect Ratio Custom
- Style
  - Style Default
- General
  - Rotation 0.00
  - Permission <inherit>
  - Permission Type Disabled
  - Connect To Monitoring Service
  - Identifier CHAN_001
- Extended Style

Alarm List Widget placed on screen

Alarm List Widget Properties

Click on Alarm List Widget icon to place widget on screen

Fig. 3-44: Alarm List Widget Added to MapView Screen

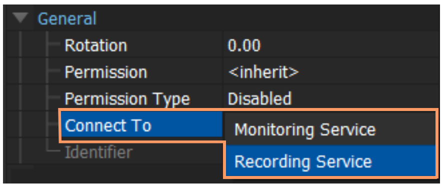
Configure the widget:

To select which data source to connect to:

7 Set the widget's 'Connect To' property to:

- 'Monitoring Service' - for live log message data; or to
- 'Recording Service' - to view data that is already logged.

.



- General
  - Rotation 0.00
  - Permission <inherit>
  - Permission Type Disabled
  - Connect To: Monitoring Service / Recording Service
  - Identifier

8 If 'Recording Service' is selected,
then one or more sets of logged data will be available. The required data set name (log file configuration identifier) must be entered in the 'Identifier' property.
Thus, in the 'Identifier' property,
enter the name of the Log File configuration set up in the Recording Service.
Note: Use the *same identifier name* as used in the Recording Service.

9 Click **Save File**.

10 Click **Save Project**.

The **Alarm List** widget has been set up.

IMPORTANT:
The **Publish to Alarm Widget** setting must be enabled within the Log File configuration of the Recording Service for the widget to connect correctly to a Recording Service.

Recording Service Checks:

If connecting to a Recording Service:

11 Access the Recording Service web page and view the required Log File configuration.

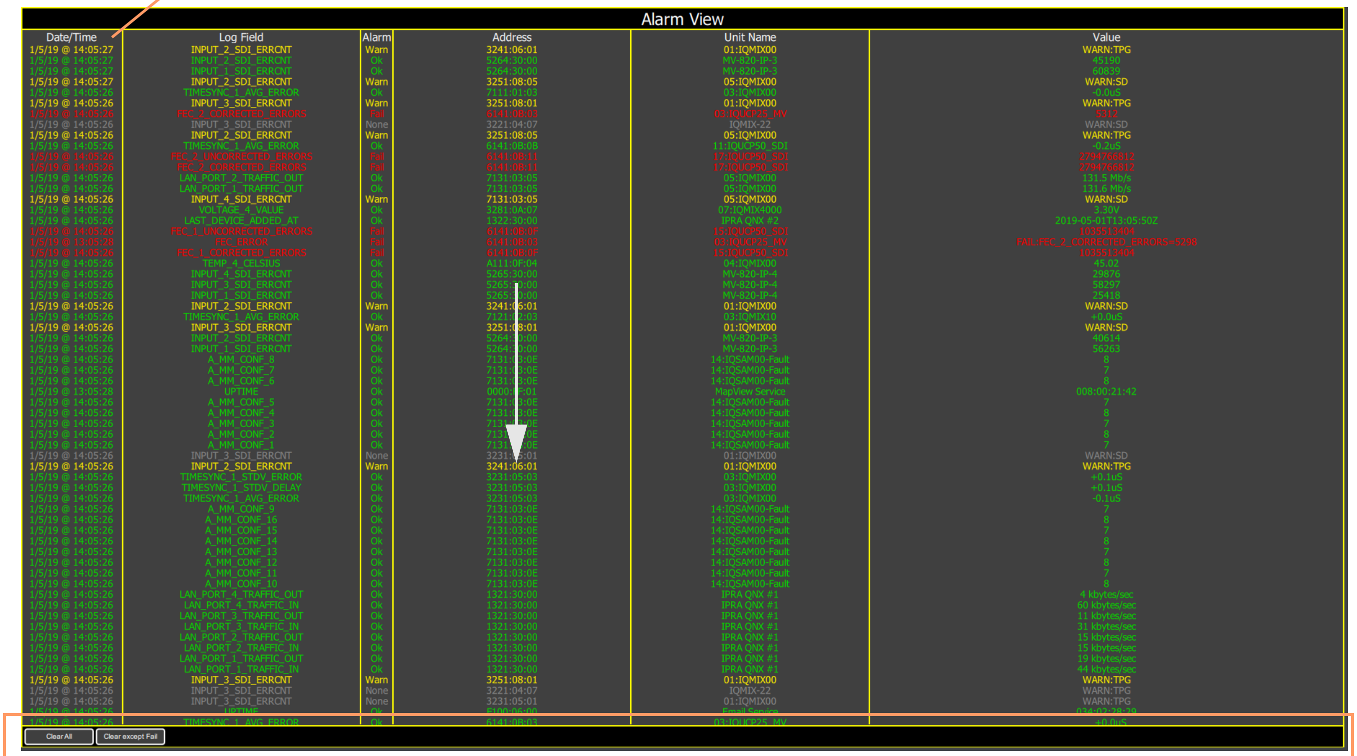
12 check the following settings:

- **Enable this log file** - Set to 'Yes'.
- **Recording Mode** - Set to 'Everything' (to record all messages', or is set to 'Alarms Only' to record alarm messages only.
  Note: This setting affects what may be seen by any connected Alarm List widgets.
- **Identifier** - Check that the identifier name configured for the widget is correct.

Exercise the widget:

13 Click the 'Run Mode' icon to enter 'Run Mode'.
The MapView project runs and the Alarm List widget is connected to the data source configured. Log messages scroll down the table, the most recent messages at the top.

Most recent messages at top.
Messages scroll down in the table as new messages are added.



Alarm View

Date/Time	Log Field	Alarm	Address	Unit Name	Value
1/5/19 @ 14:05:27	INPUT_2_SDI_ERRCNT	Warn	3241:06:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:05:27	INPUT_2_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	45190
1/5/19 @ 14:05:27	INPUT_1_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	60839
1/5/19 @ 14:05:27	INPUT_2_SDI_ERRCNT	Warn	3251:08:05	05:IQMIX00	WARN:SD
1/5/19 @ 14:05:26	TIMESYNC_1_AVG_ERROR	Ok	7111:01:03	03:IQMIX00	-0.0uS
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	Warn	3251:08:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:05:26	FEC_2_CORRECTED_ERRORS	Fail	6141:0B:03	03:IQUCP25_MV	5312
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	None	3221:04:07	IQMIX-22	WARN:SD
1/5/19 @ 14:05:26	INPUT_2_SDI_ERRCNT	Warn	3251:08:05	05:IQMIX00	WARN:TPG
1/5/19 @ 14:05:26	TIMESYNC_1_AVG_ERROR	Ok	6141:0B:0B	11:IQUCP50_SDI	-0.2uS
1/5/19 @ 14:05:26	FEC_2_UNCORRECTED_ERRORS	Fail	6141:0B:11	17:IQUCP50_SDI	2794766812
1/5/19 @ 14:05:26	FEC_2_CORRECTED_ERRORS	Fail	6141:0B:11	17:IQUCP50_SDI	2794766812
1/5/19 @ 14:05:26	LAN_PORT_2_TRAFFIC_OUT	Ok	7131:03:05	05:IQMIX00	131.5 Mb/s
1/5/19 @ 14:05:26	LAN_PORT_1_TRAFFIC_OUT	Ok	7131:03:05	05:IQMIX00	131.6 Mb/s
1/5/19 @ 14:05:26	INPUT_4_SDI_ERRCNT	Warn	7131:03:05	05:IQMIX00	WARN:SD
1/5/19 @ 14:05:26	VOLTAGE_4_VALUE	Ok	3281:0A:07	07:IQMIX4000	3.30V
1/5/19 @ 14:05:26	LAST_DEVICE_ADDED_AT	Ok	1322:30:00	IPRA QNX #2	2019-05-01T13:05:50Z
1/5/19 @ 14:05:26	FEC_1_UNCORRECTED_ERRORS	Fail	6141:0B:0F	15:IQUCP50_SDI	1035513404
1/5/19 @ 13:05:28	FEC_ERROR	Fail	6141:0B:03	03:IQUCP25_MV	FAIL:FEC_2_CORRECTED_ERRORS=5298
1/5/19 @ 14:05:26	FEC_1_CORRECTED_ERRORS	Fail	6141:0B:0F	15:IQUCP50_SDI	1035513404
1/5/19 @ 14:05:26	TEMP_4_CELSIUS	Ok	A111:0F:04	04:IQMIX00	45.02
1/5/19 @ 14:05:26	INPUT_4_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	29876
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	58297
1/5/19 @ 14:05:26	INPUT_1_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	25418
1/5/19 @ 14:05:26	INPUT_2_SDI_ERRCNT	Warn	3241:06:01	01:IQMIX00	WARN:SD
1/5/19 @ 14:05:26	TIMESYNC_1_AVG_ERROR	Ok	7121:02:03	03:IQMIX10	+0.0uS
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	Warn	3251:08:01	01:IQMIX00	WARN:SD
1/5/19 @ 14:05:26	INPUT_2_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	40614
1/5/19 @ 14:05:26	INPUT_1_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	56263
1/5/19 @ 14:05:26	A_MM_CONF_8	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_7	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_6	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 13:05:28	UPTIME	Ok	0000:FF:01	MapView Service	008:00:21:42
1/5/19 @ 14:05:26	A_MM_CONF_5	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_4	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_3	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_2	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_1	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	None	3231:05:01	01:IQMIX00	WARN:SD
1/5/19 @ 14:05:26	INPUT_2_SDI_ERRCNT	Warn	3241:06:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:05:26	TIMESYNC_1_STDV_ERROR	Ok	3231:05:03	03:IQMIX00	+0.1uS
1/5/19 @ 14:05:26	TIMESYNC_1_STDV_DELAY	Ok	3231:05:03	03:IQMIX00	+0.1uS
1/5/19 @ 14:05:26	TIMESYNC_1_AVG_ERROR	Ok	3231:05:03	03:IQMIX00	-0.1uS
1/5/19 @ 14:05:26	A_MM_CONF_9	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_16	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_15	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_14	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_13	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_12	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	A_MM_CONF_11	Ok	7131:03:0E	14:IQSAM00-Fault	7
1/5/19 @ 14:05:26	A_MM_CONF_10	Ok	7131:03:0E	14:IQSAM00-Fault	8
1/5/19 @ 14:05:26	LAN_PORT_4_TRAFFIC_OUT	Ok	1321:30:00	IPRA QNX #1	4 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_4_TRAFFIC_IN	Ok	1321:30:00	IPRA QNX #1	60 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_3_TRAFFIC_OUT	Ok	1321:30:00	IPRA QNX #1	11 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_3_TRAFFIC_IN	Ok	1321:30:00	IPRA QNX #1	31 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_2_TRAFFIC_OUT	Ok	1321:30:00	IPRA QNX #1	15 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_2_TRAFFIC_IN	Ok	1321:30:00	IPRA QNX #1	15 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_1_TRAFFIC_OUT	Ok	1321:30:00	IPRA QNX #1	19 kbytes/sec
1/5/19 @ 14:05:26	LAN_PORT_1_TRAFFIC_IN	Ok	1321:30:00	IPRA QNX #1	44 kbytes/sec
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	Warn	3251:08:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	None	3221:04:07	IQMIX-22	WARN:TPG
1/5/19 @ 14:05:26	INPUT_3_SDI_ERRCNT	None	3231:05:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:05:26	UPTIME	Ok	E100:06:00	Email Service	034:02:28:29
1/5/19 @ 14:05:26	TIMESYNC_1_AVG_ERROR	Ok	6141:0B:03	03:IQUCP25_MV	+0.0uS

Footer, may be disabled with the 'Display Footer' property.

Fig. 3-45: Log Messages in Alarm List

Log messages are color-coded: Red for Fail/Error messages; Yellow for Warnings, etc.

Alarm List Widget Features

Navigating the List

Most recent messages are added into the top of the list, so the log messages are always auto-scrolling down the screen.

Users can scroll up/down the list by dragging/flicking inside the widget table. This will temporarily disable the auto-scroll, which will be re-enabled after 5 seconds.

Alarm List Widget History Depth

Alarm List widgets are limited to 200 entry items (i.e. 200 table rows maximum).

For widgets connected to a Recording Service, this depth is maintained server-side. This means that even late-joining widgets will see the previous 200 entries.

Widget Columns

Date/Time - Date and time of the log message.

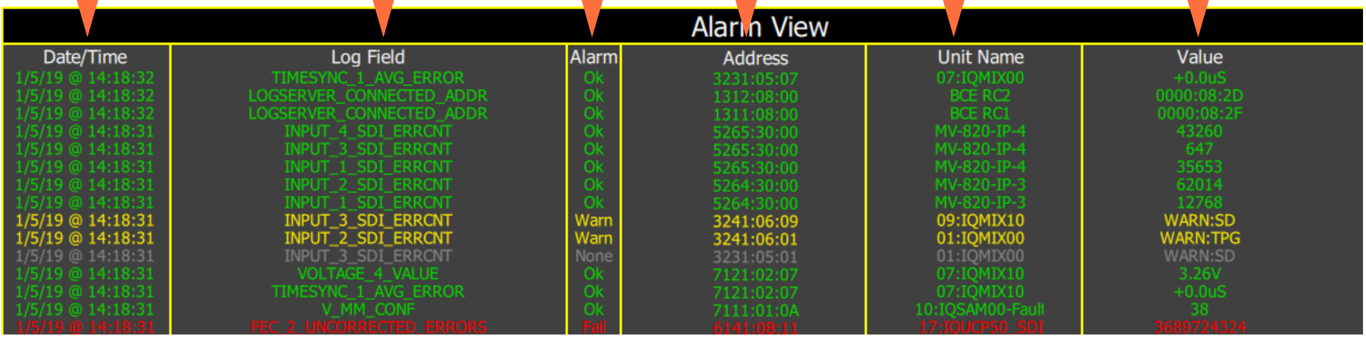
Log Field - Log message type. For example, input SDI error count, voltage reading, status message.

Alarm - Alarm state: OK, Warning, Fail.

Address - RollCall address of source of message.

Unit Name - Namer of the device.

Value - Value associated with message.

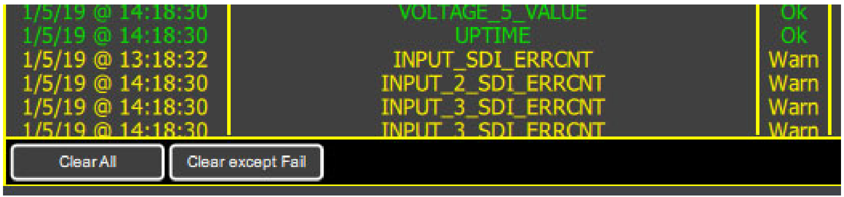


Alarm View

Date/Time	Log Field	Alarm	Address	Unit Name	Value
1/5/19 @ 14:18:32	TIMESYNC_1_AVG_ERROR	Ok	3231:05:07	07:IQMIX00	+0.0uS
1/5/19 @ 14:18:32	LOGSERVER_CONNECTED_ADDR	Ok	1312:08:00	BCE RC2	0000:08:2D
1/5/19 @ 14:18:32	LOGSERVER_CONNECTED_ADDR	Ok	1311:08:00	BCE RC1	0000:08:2F
1/5/19 @ 14:18:31	INPUT_4_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	43260
1/5/19 @ 14:18:31	INPUT_3_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	647
1/5/19 @ 14:18:31	INPUT_1_SDI_ERRCNT	Ok	5265:30:00	MV-820-IP-4	35653
1/5/19 @ 14:18:31	INPUT_2_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	62014
1/5/19 @ 14:18:31	INPUT_1_SDI_ERRCNT	Ok	5264:30:00	MV-820-IP-3	12768
1/5/19 @ 14:18:31	INPUT_3_SDI_ERRCNT	Warn	3241:06:09	09:IQMIX10	WARN:SD
1/5/19 @ 14:18:31	INPUT_2_SDI_ERRCNT	Warn	3241:06:01	01:IQMIX00	WARN:TPG
1/5/19 @ 14:18:31	INPUT_3_SDI_ERRCNT	None	3231:05:01	01:IQMIX00	WARN:SD
1/5/19 @ 14:18:31	VOLTAGE_4_VALUE	Ok	7121:02:07	07:IQMIX10	3.26V
1/5/19 @ 14:18:31	TIMESYNC_1_AVG_ERROR	Ok	7121:02:07	07:IQMIX10	+0.0uS
1/5/19 @ 14:18:31	V_MM_CONF	Ok	7111:01:0A	10:IQSAM00-Fault	38
1/5/19 @ 14:18:31	FEC_2_UNCORRECTED_ERRORS	Fail	6141:0B:11	17:IQUCP50_SDI	3689724324

Fig. 3-46: Alarm List Widget Columns

Widget Controls



1/5/19 @ 14:18:30	VOLTAGE_5_VALUE	Ok
1/5/19 @ 14:18:30	UPTIME	Ok
1/5/19 @ 13:18:32	INPUT_SDI_ERRCNT	Warn
1/5/19 @ 14:18:30	INPUT_2_SDI_ERRCNT	Warn
1/5/19 @ 14:18:30	INPUT_3_SDI_ERRCNT	Warn
1/5/19 @ 14:18:30	INPUT_3_SDI_ERRCNT	Warn

Fig. 3-47: Alarm List Widget Controls

Table 3-5: Alarm List Widget Controls

Setting	Description
Clear All	Button. Click to clear Alarm List of all messages.
Clear except Fail	Button. Click to clear Alarm List of all messages except fail/error messages.

Note:

Clearing Alarm Lists:

When clicking **Clear All** or **Clear except Fail**, a system 'clear' command is sent to the service which the Alarm List widget is connected to.

When connected to:

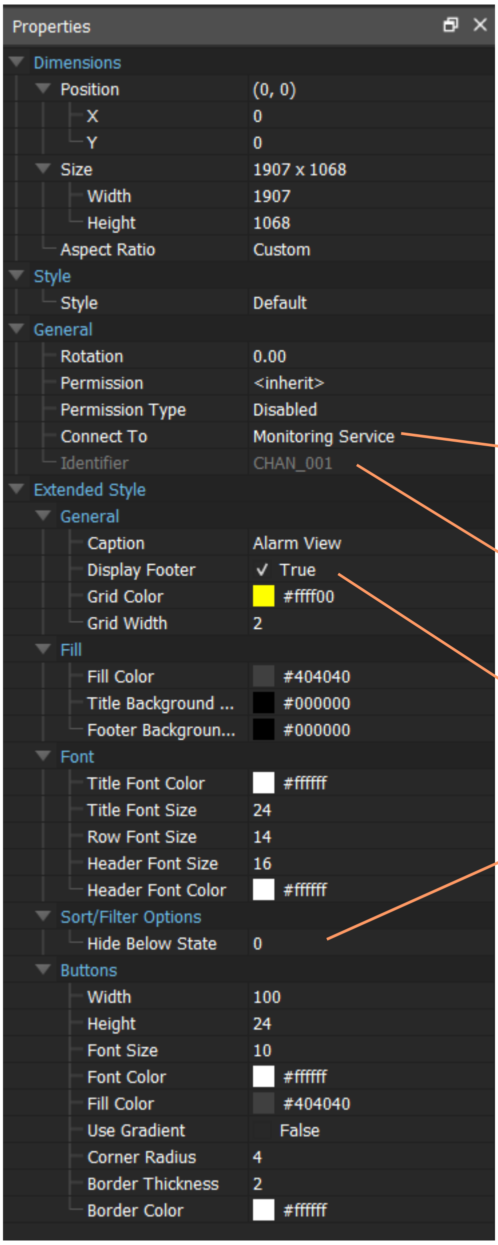
- **Recording Service** - *all* **Alarm List** widgets viewing that recorded log file data are cleared.
  And the system 'clear' command itself is also recorded in the log file.
- **Monitoring Service** ("live" data) - only the *local* **Alarm List** widget is cleared.

Some Widget Properties

The appearance and style of the widget is configurable via property values, like other Orbit MapView widgets. The properties are listed in Figure 3-48.

The 'Display Footer' properties enables the table footer to be hidden from display.

The 'Hide Below State' property enables some simple filtering of displayed messages on a widget-by widget basis. Log messages can be hidden when they are below a threshold State Value.



Properties

Dimensions	
Position	(0, 0)
X	0
Y	0
Size	1907 x 1068
Width	1907
Height	1068
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Connect To	Monitoring Service
Identifier	CHAN_001
Extended Style	
General	
Caption	Alarm View
Display Footer	✓ True
Grid Color	🟨 #ffff00
Grid Width	2
Fill	
Fill Color	⬛ #404040
Title Background ...	⬛ #000000
Footer Backgroun...	⬛ #000000
Font	
Title Font Color	⬜ #ffffff
Title Font Size	24
Row Font Size	14
Header Font Size	16
Header Font Color	⬜ #ffffff
Sort/Filter Options	
Hide Below State	0
Buttons	
Width	100
Height	24
Font Size	10
Font Color	⬜ #ffffff
Fill Color	⬛ #404040
Use Gradient	False
Corner Radius	4
Border Thickness	2
Border Color	⬜ #ffffff

Fig. 3-48: Alarm List Widget Properties

Connect To - Select source of log message data.

Identifier - for Recording Service sources, select log data.

Display Footer - enables/disables display of the table footer row.

Hide Below State - Disables display of log message with State Value below a set value.

(0 = Masked,
1 = OK,
49=Acknowledged warning,
50=Warning,
99 = acknowledged Fail/error,
100=Fail/Error)

Caution: This setting can hide log messages from view.

4 Components and Variables

Summary

Components and Variables

Components are used to hold graphical widgets and Behaviours and Bindings. They can be used many times over on a screen.

Components:

- contain graphical widgets, in a similar way to 'Tiles' when using Orbit for Multiviewer applications;
- are edited in the usual Orbit graphical editor;
- can hold Behaviours/Bindings and variables in a similar way to screens; and
- can be used many times over on Orbit MapView screens (similar to Orbit 'Tiles' on Orbit multiviewer 'Walls'); and
- are typically used in Orbit MapView screens where there is re-use of graphical layouts.

The Component

Creating a New Component

1 Right-click on the ‘components’ item in the **Project View** pane of the Orbit MapView screen.
And select **New Component**.

2 Enter a component **Name** in the dialog and click OK.

A new, blank component sheet is opened in the graphical editor.

3 Check that you are in ‘Design Mode’ in Orbit.
(See Orbit Design Mode and Run Mode, on page 32.)

4 Add widgets to the component sheet to form your new component.

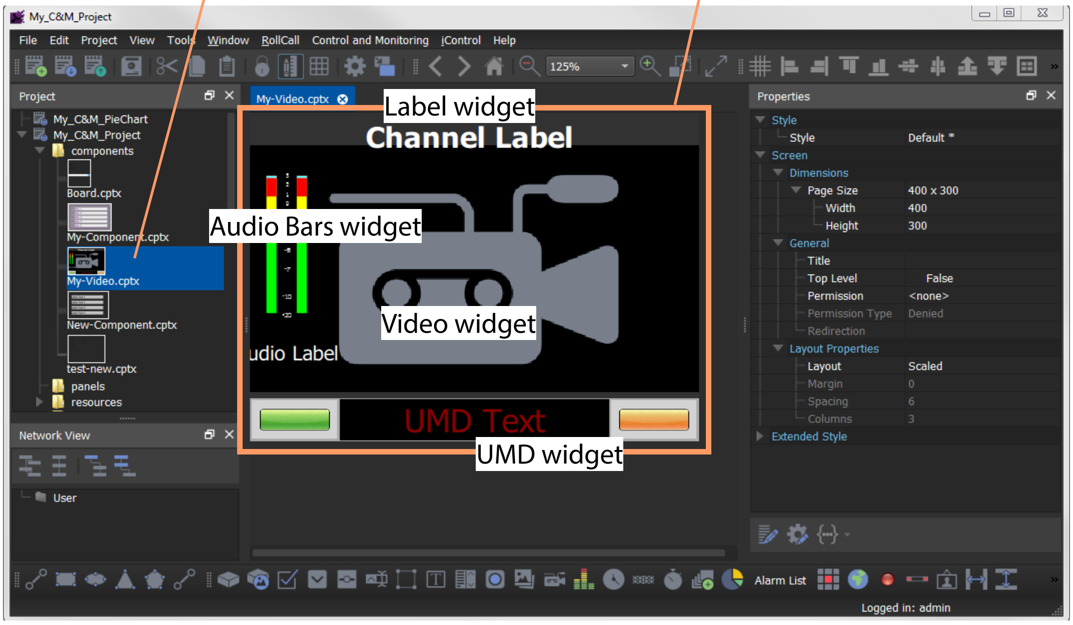
The example shown in Figure 4-1 (new component ‘My-Video’) contains:
a ‘Video’ widget overlaid with an ‘Audio Bars’ widget, all above a ‘UMD’ widget with a ‘Label’ widget providing header text.

5 Click **Save**.

The component has been created and is represented in the **Project View**.

Component ‘My-Video’ in schematic editor

Component ‘My-Video’ in **Project View**



Style
Style Default *
Screen
Dimensions
Page Size 400 x 300
Width 400
Height 300
General
Title
Top Level False
Permission <none>
Permission Type Denied
Redirection
Layout Properties
Layout Scaled
Margin 0
Spacing 6
Columns 3
Extended Style

Fig. 4-1: Component ‘My-Video’

Using a Component on a Screen

The new component 'My-Video' may now be used on an Orbit MapView screen

1. Open a screen schematic.
2. Set Orbit to 'Design Mode'.
   (Toggle modes with the 'Design/Run mode' main menu bar icon, see Orbit Design Mode and Run Mode, on page 32.).
3. In the **Project View** pane,
   expand the 'components' item and select the 'My-Video' component.
4. Drag the component onto the screen. See Figure 4-2.

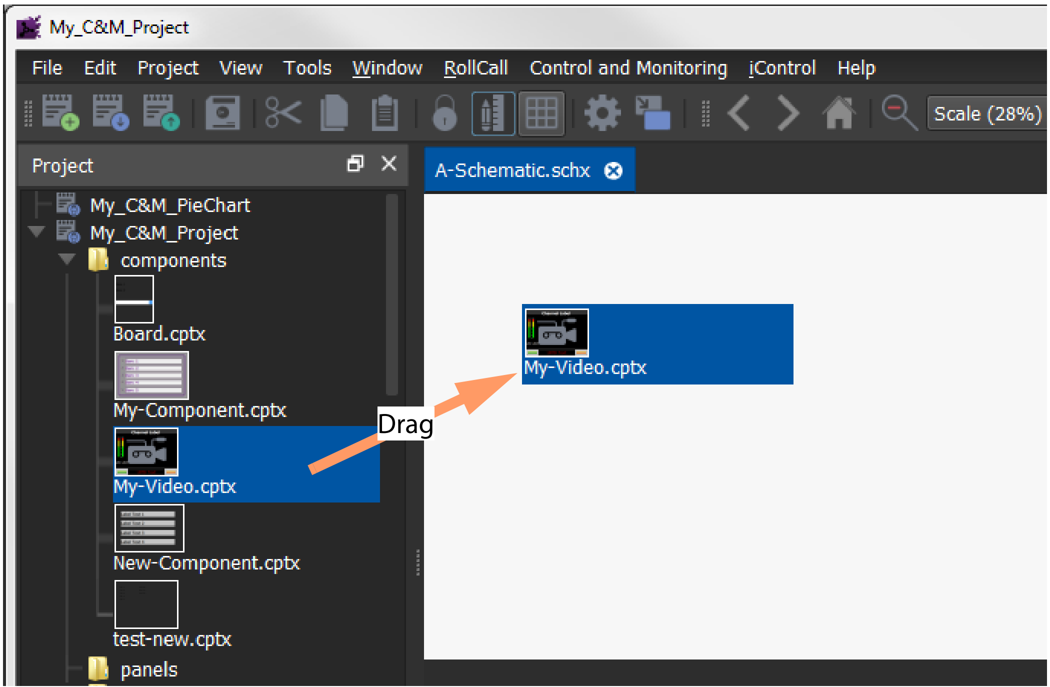


Fig. 4-2: Drag Component onto Screen Schematic

The component may be used on a screen one or more times. See Figure 4-3.

A component may also be placed into another component.

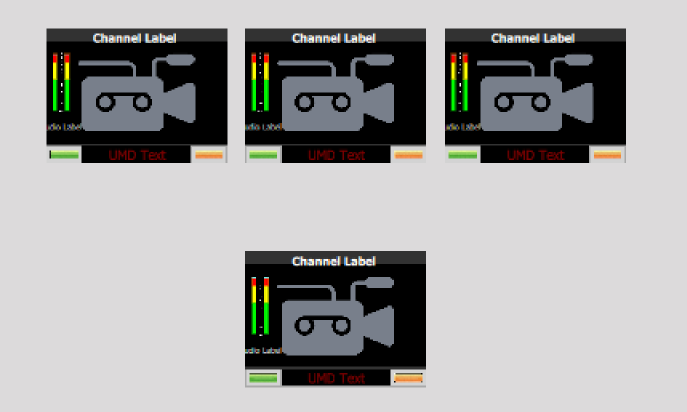


Fig. 4-3: Several Instances of Component 'My-Video'

It is useful to be able to re-use a component, but, at the moment in Figure 4-3, the 'Channel Label' and 'Audio Label' text on the 'My-Video' component are fixed. This is because we have not linked any **Label** widget caption properties to any component variables at the screen-level.

Variables in Orbit

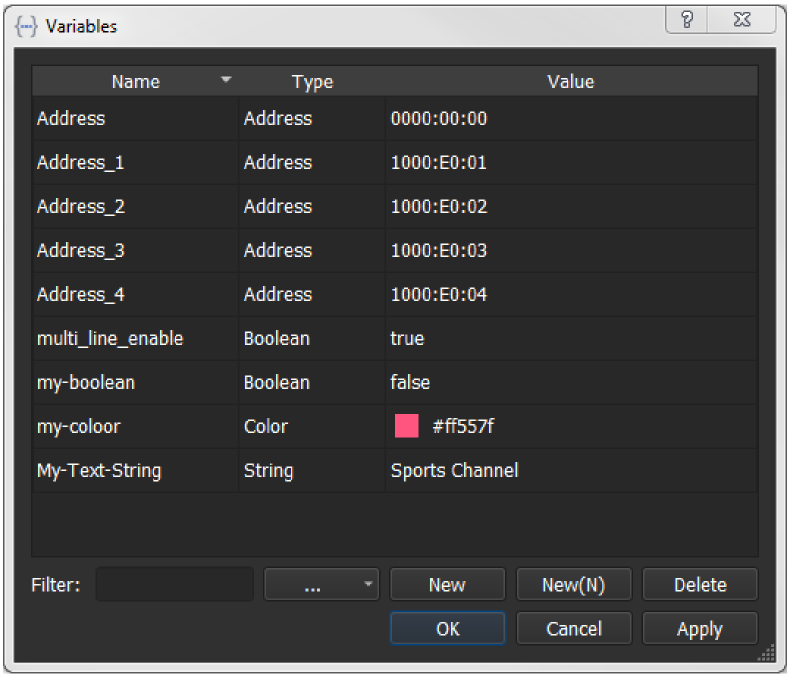
Variables can be set up in screen or component schematics. These can hold values which are defined at a higher level on the Orbit MapView screen or at run time. Variables are useful and can be used to set property values to common settings.

Create a Variable

To create and define a variable on a screen or components:

- Right-click in the background of a screen or component schematic and select '**Variables...**'

  The **Variables** dialog is shown, showing variables of the schematic. See Figure 4-4. New variables can be added. Variable values can be set.



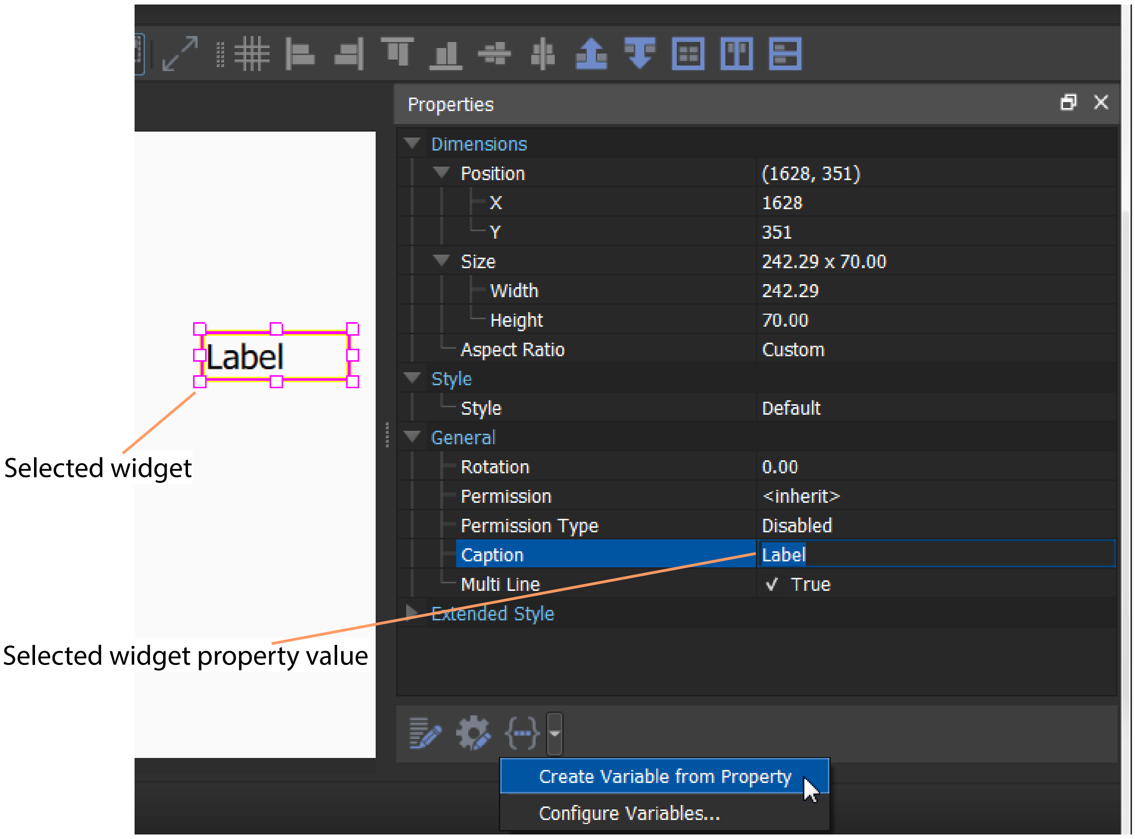
Name	Type	Value
Address	Address	0000:00:00
Address_1	Address	1000:E0:01
Address_2	Address	1000:E0:02
Address_3	Address	1000:E0:03
Address_4	Address	1000:E0:04
multi_line_enable	Boolean	true
my-boolean	Boolean	false
my-coloor	Color	🟥 #ff557f
My-Text-String	String	Sports Channel

Fig. 4-4: Variables Dialog

Create a Variable from a Property Value

To create a variable from a widget property value:

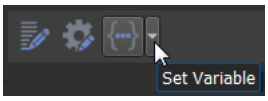
- Select a widget on a screen or on a component schematic.
- Select a widget property value field in the **Property** box. See Figure 4-5.



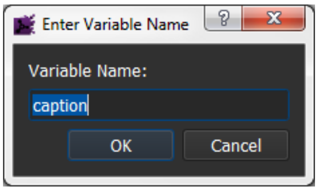
Dimensions	
Position	(1628, 351)
X	1628
Y	351
Size	242.29 x 70.00
Width	242.29
Height	70.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	Label
Multi Line	✓ True
Extended Style	

Fig. 4-5: Create Variable from Property

- Click the ‘Set Variable’ downward-arrow button, , and select ‘Create Variable from Property’.



An **Enter Variable Name** dialog is shown. See Figure 4-6.

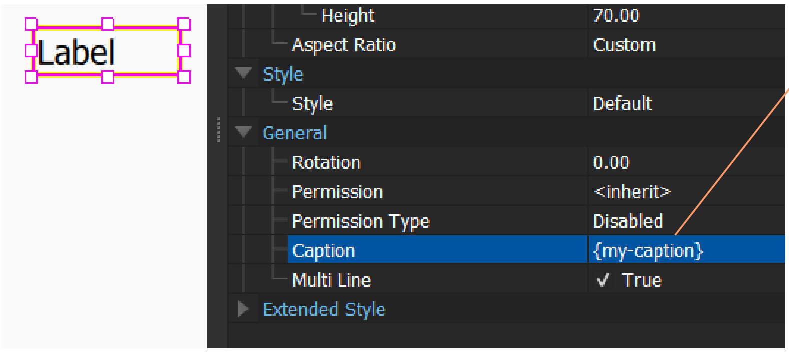


Variable Name: caption

Fig. 4-6: Enter Variable Name Dialog

- The variable name is filled out with the property name and may be edited. Click **OK** when done.
- The new variable name appears in the property value field surrounded by curly brackets. See Figure 4-7.

New variable name surrounded by curly brackets, {my-caption}



Height	70.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{my-caption}
Multi Line	✓ True
Extended Style	

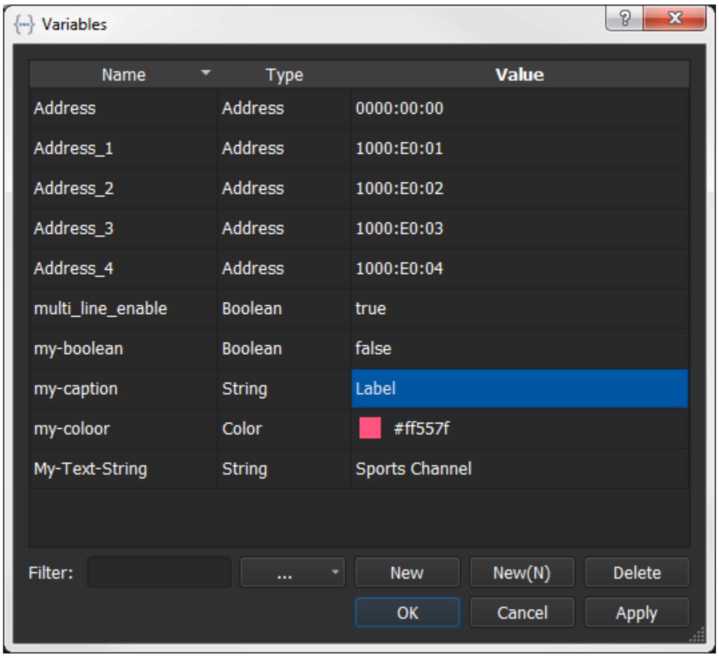
Fig. 4-7: Variable Name in Property Value Field

The new variable is now defined on the screen/component schematic.

View Variables/Set Variable Values

To view all the variables:

- Right-click in the background of a screen or component schematic:, and select '**Variables...'** to show the **Variables** dialog. The variable values can be set in this dialog.



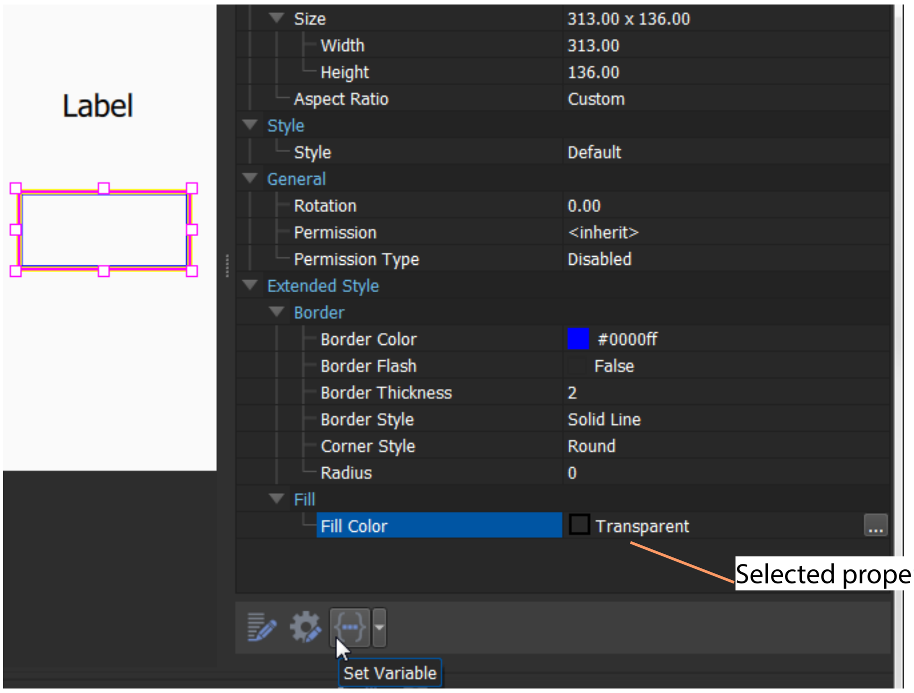
Name	Type	Value
Address	Address	0000:00:00
Address_1	Address	1000:E0:01
Address_2	Address	1000:E0:02
Address_3	Address	1000:E0:03
Address_4	Address	1000:E0:04
multi_line_enable	Boolean	true
my-boolean	Boolean	false
my-caption	String	Label
my-coloor	Color	🟥 #ff557f
My-Text-String	String	Sports Channel

Fig. 4-8: Variables Dialog

Use a Variable in a Property Value

To use a variable in a property value:

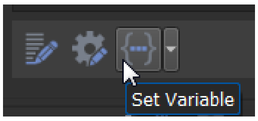
- Select a property value. See Figure 4-9.



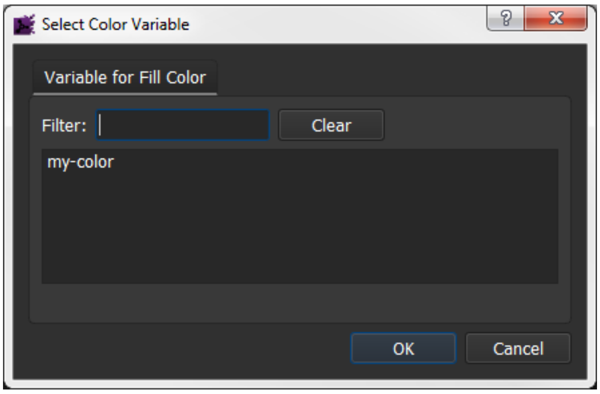
Size	313.00 x 136.00
Width	313.00
Height	136.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Extended Style	
Border	
Border Color	🟦 #0000ff
Border Flash	False
Border Thickness	2
Border Style	Solid Line
Corner Style	Round
Radius	0
Fill	
Fill Color	□ Transparent

Selected property value

Fig. 4-9: Select Property Value

- Click the **{...}** 'Set Variable' button, ,

  A **Select Variable** dialog is shown.
  The variables listed will be compatible with the selected property value type. For example, for color property value types, color type variables are shown.



Select Color Variable

Variable for Fill Color

Filter: Clear

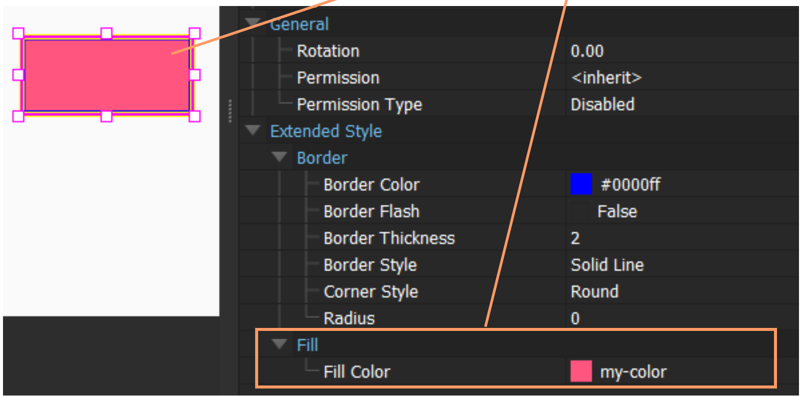
my-color

OK Cancel

Fig. 4-10: Select Variables Dialog

- Select a variable name and click **OK**.

  The property value is now set to the variable selected. See Figure 4-11.



General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Extended Style	
Border	
Border Color	🟦 #0000ff
Border Flash	False
Border Thickness	2
Border Style	Solid Line
Corner Style	Round
Radius	0
Fill	
Fill Color	🟥 my-color

Property value is now set to the selected variable, here it is 'my-color', setting the 'Fill Color 'property value.

Fig. 4-11: Variable Used for Selected Property Value

Enter a Variable Directly into a String Value

To use a variable in a text string, surround the variable name with curly braces { } . For example:

- {Address}
- {Input}
- {My-Variable}

Placing Functionality Inside a Component

Components are a useful way of encapsulating some graphical objects and their 'behind-the-scenes' logic for re-use.

In this sub-section, the test bench screen from Extend Widget Functionality with Behaviours and Bindings, on page 58, is developed by encapsulating some of the screen functionality into a Component with the screen's **Pie Chart** widget and its associated functionality. The test bench screen from Extend Widget Functionality with Behaviours and Bindings, on page 58, is shown in Figure 4-12.

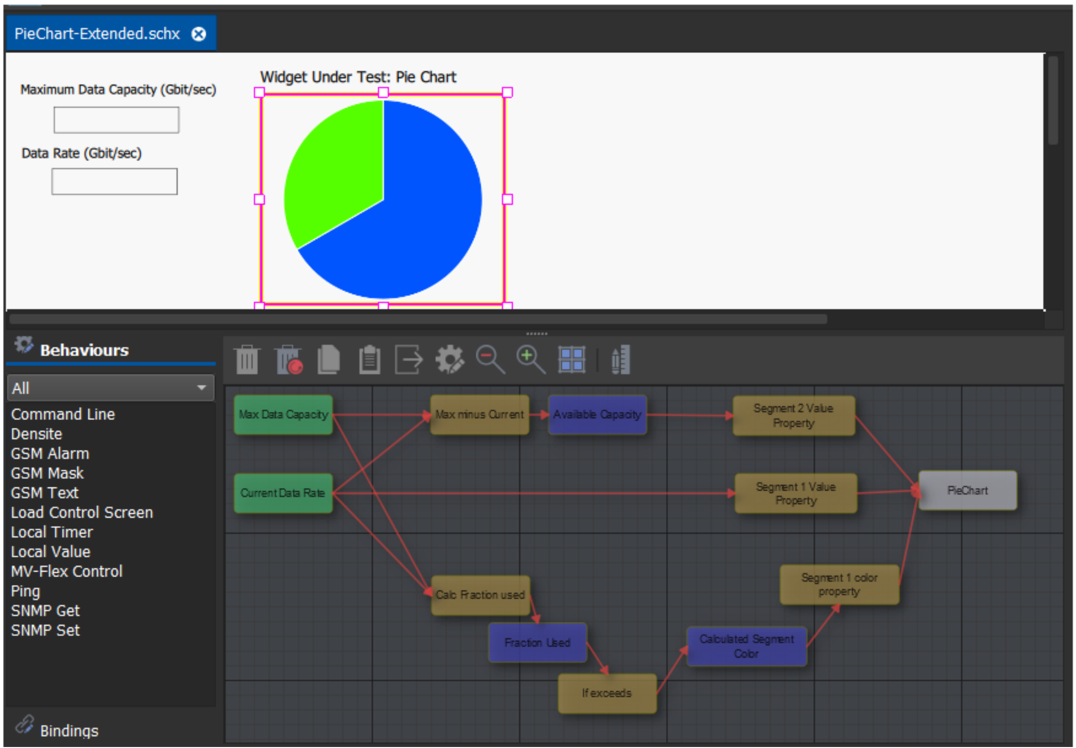


Fig. 4-12: Pie Chart Test Bench Screen (of Extend Widget Functionality with Behaviours and Bindings*, on page 58)*

The following are covered in this sub-section:

- Create New Component, on page 92;
- Connect Widget-Level Bindings to Component Variables, on page 93; and
- Exercise the Component, on page 103;

Create New Component

1. Open the test bench screen and select the **Pie Chart** widget (as shown in Figure 4-12).
2. Right-click on the selected **Pie Chart** widget and select 'Copy'.
   The widget with its associated Behaviours and Bindings is copied to the clipboard.
3. Right-click on the 'components' item in the Project View pane and select **New Component**.
   A new blank component is opened in new tab of the schematic editor.
4. Right-click in the blank area of the component and select 'Paste'.
   The copied widget etc. is pasted into the new component. See Figure 4-13.

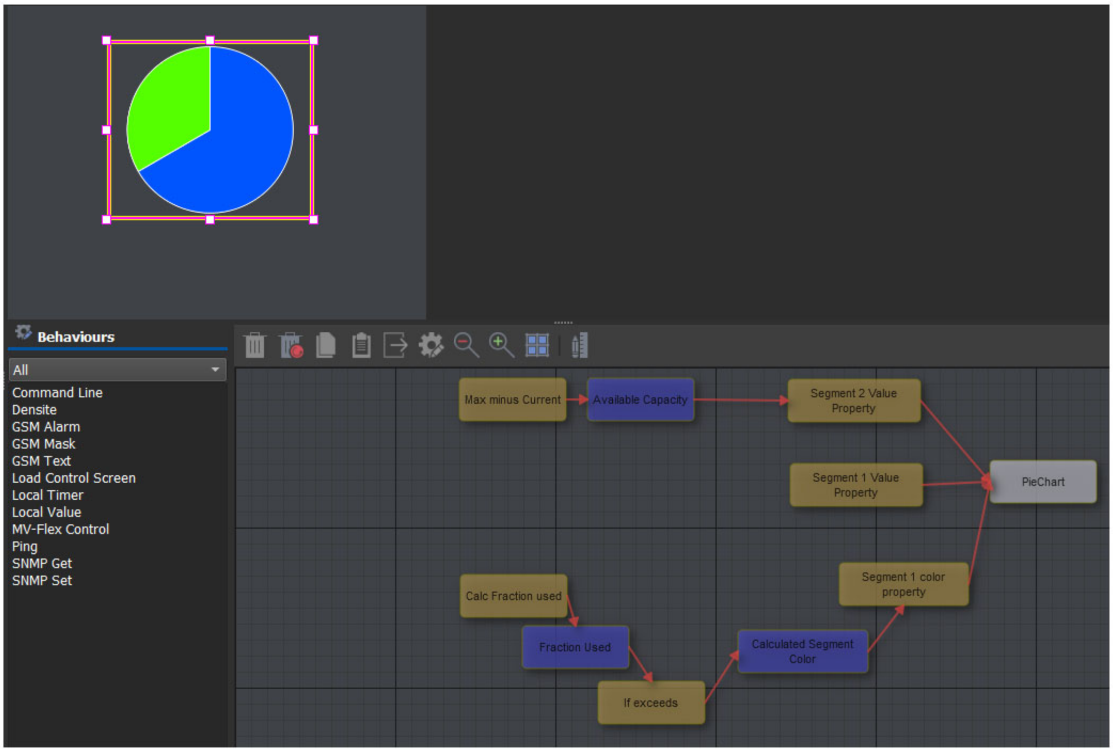


Fig. 4-13: Initial New Component

The pasted items comprise the copied widget with its widget-level Behaviours and Bindings. Items that are NOT copied from the original schematic relate to screen-level items and these need to be recreated in our component, which is described below.

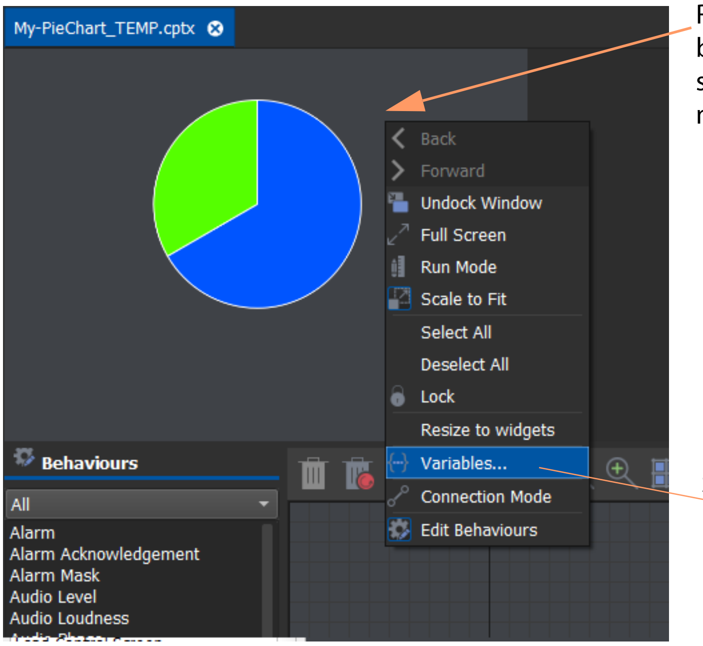
Connect Widget-Level Bindings to Component Variables

Create Component-Level Variables

The first step is to create component-level variables. For our **Pie Chart** example, we need a variable for 'Maximum Data Capacity' and a variable for 'Current Data Rate'.

In the opened component:

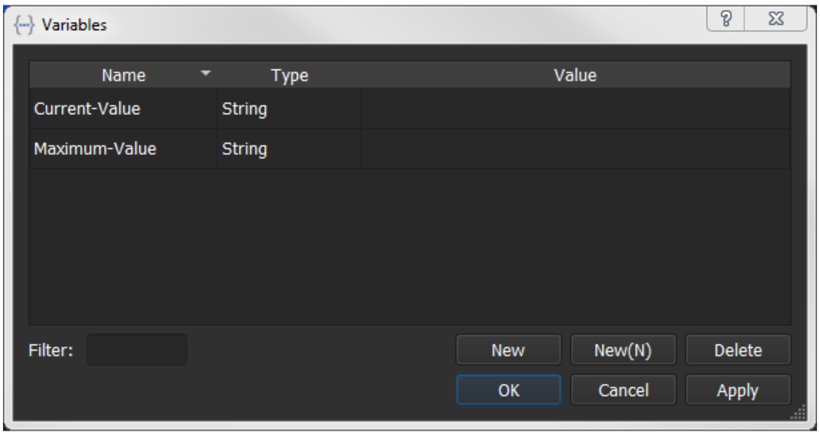
1 Right-click on the component's background and select '**Variables...**', see Figure 4-14. The **Variables** dialog is shown. See Figure 4-15.



Right-click on component background, with nothing selected to show context menu.

Select '**Variables...**'

Fig. 4-14: Right-click on Component Background



Name	Type	Value
Current-Value	String	
Maximum-Value	String	

Fig. 4-15: Variables Dialog

2 Click **New** and a new line item appears in the **Variables** dialog.

3 Enter a **Name** for the new item. This is the name of a new component variable.

4 Repeat to add a second variable. See Figure 4-15.

5 Click **OK** to close the dialog.
The component schematic edit screen is shown.

6 Ensure that no widget is selected in the component schematic.
And click on the **Edit** Behaviours button to show the 'Behaviours and Bindings' graphical editor screen. (Clicking the button again toggles this on/off.)

7 Display the Behaviours list in the 'Behaviours and Bindings' editor screen.
and click on the 'Local Value' item twice to add two **Local Value** Behaviours.
See Figure 4-16.

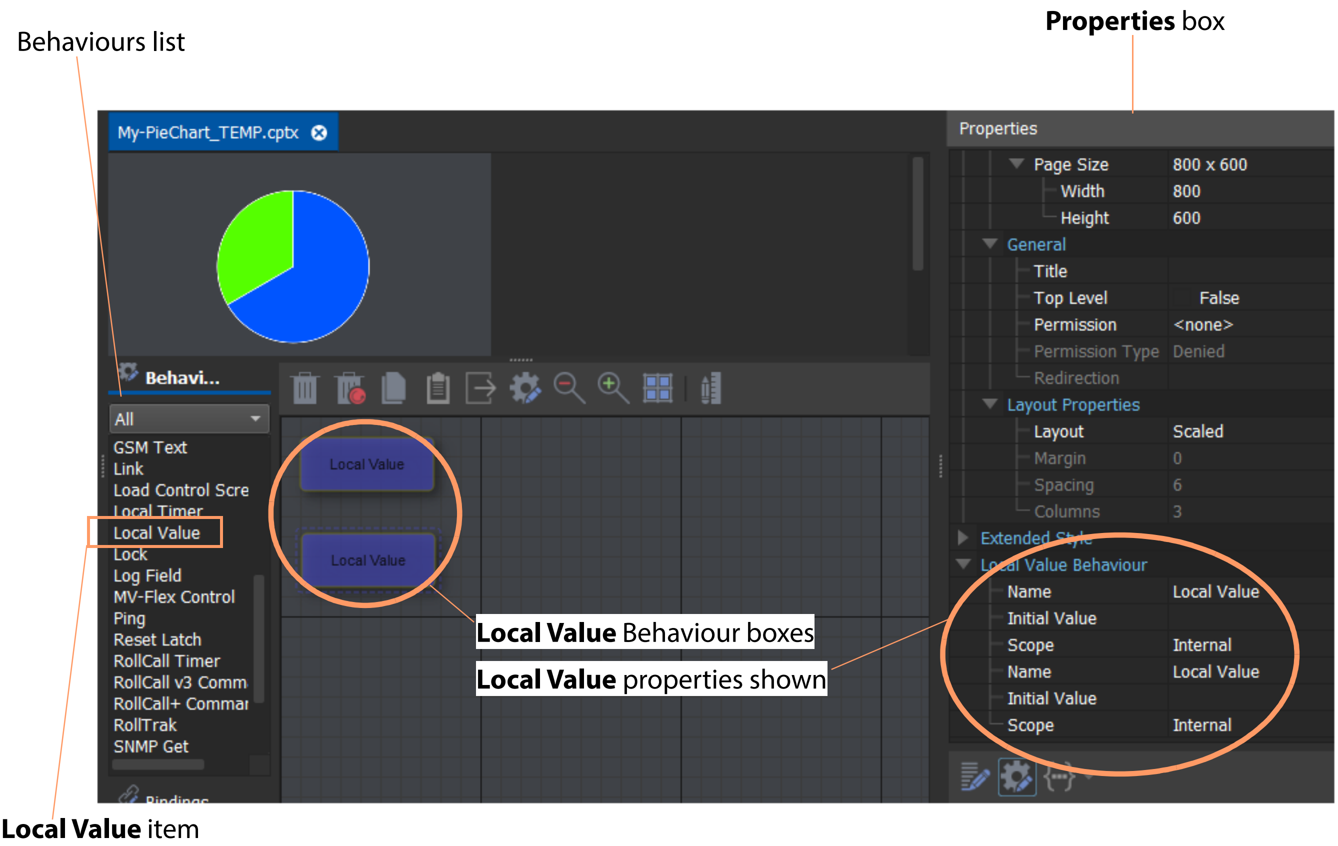


Fig. 4-16: Added Two 'Local Value' Behaviours

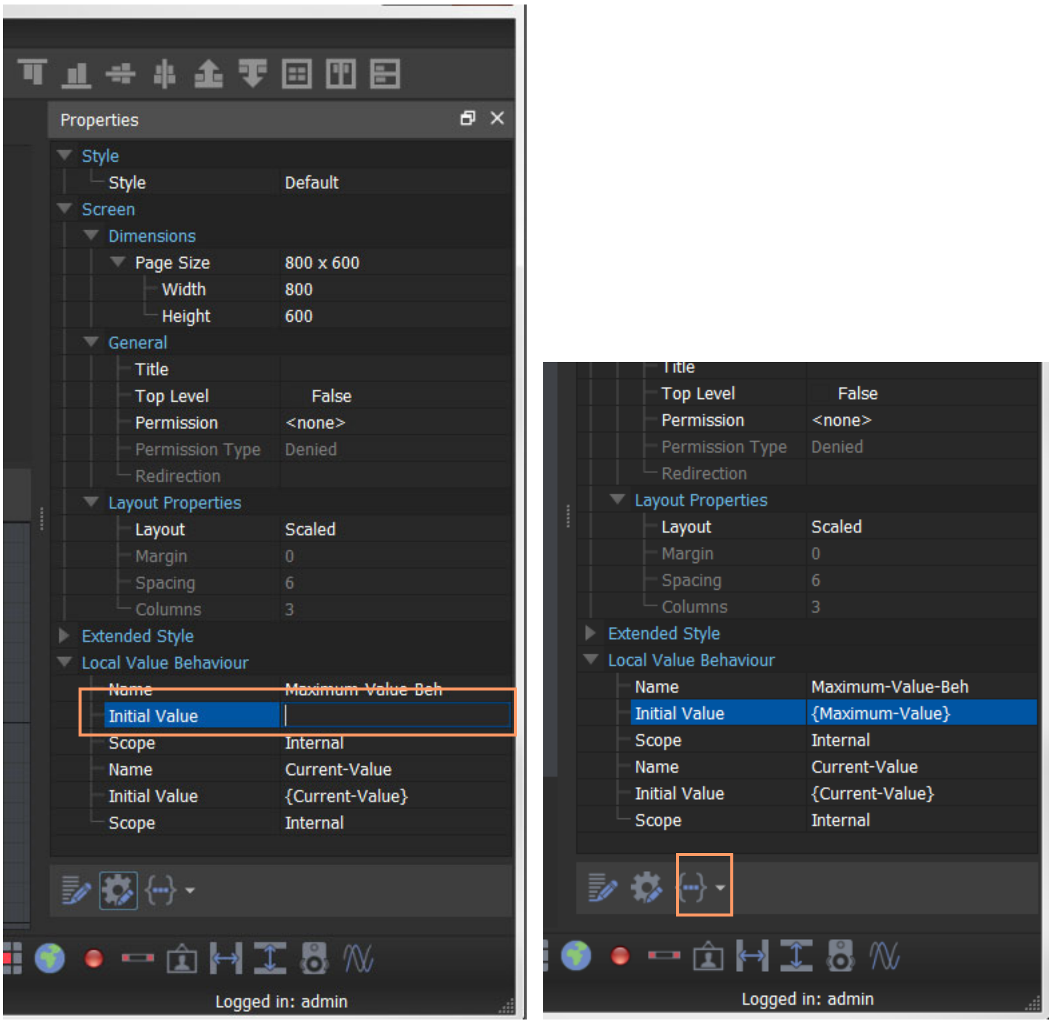
Now edit the **Local Value** Behaviour properties. This can be done in one of two methods, see below:

Behaviour property value editing, Method 1:

8 Click on a **Name** property in the property box.
And enter a new name the **Local Value**, for example, use 'Max Local Value'.
See Figure 4-17a.

9 Click on the **Initial Value** property.
And click the **Set Variable** button. See Figure 4-17b.

A **Select String Variable** dialog is shown. See Figure 4-18.



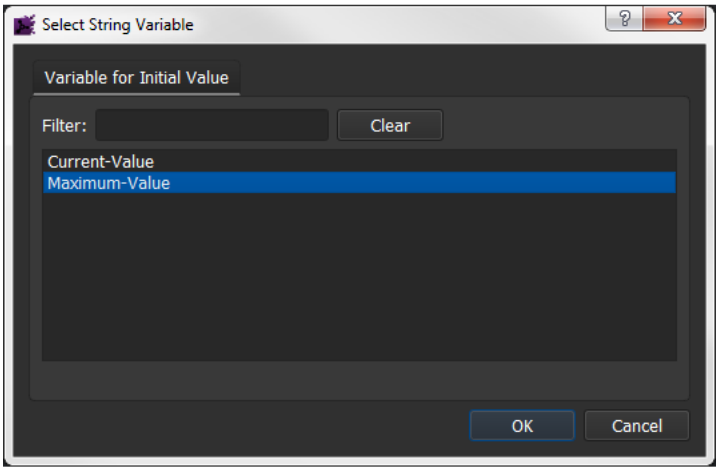
Style	
Style	Default
Screen	
Dimensions	
Page Size	800 x 600
Width	800
Height	600
General	
Title	
Top Level	False
Permission	<none>
Permission Type	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	
Local Value Behaviour	
Name	Maximum-Value-Beh
Initial Value	
Scope	Internal
Name	Current-Value
Initial Value	{Current-Value}
Scope	Internal

Title	
Top Level	False
Permission	<none>
Permission Type	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	
Local Value Behaviour	
Name	Maximum-Value-Beh
Initial Value	{Maximum-Value}
Scope	Internal
Name	Current-Value
Initial Value	{Current-Value}
Scope	Internal

a) Editing **Local Value** Properties

b) **Set Variable** Button

Fig. 4-17: Editing Local Value Properties: a) Properties Box; b) Set Variables



Current-Value
Maximum-Value

Fig. 4-18: Setting a String Property Value

10 For the **Local Value** which shall contain the ‘Maximum Data Capacity’ value in our example,
select the ‘**MaxImum**’ item in the variables list shown in the **Select String Variable** dialog. (Figure 4-18)

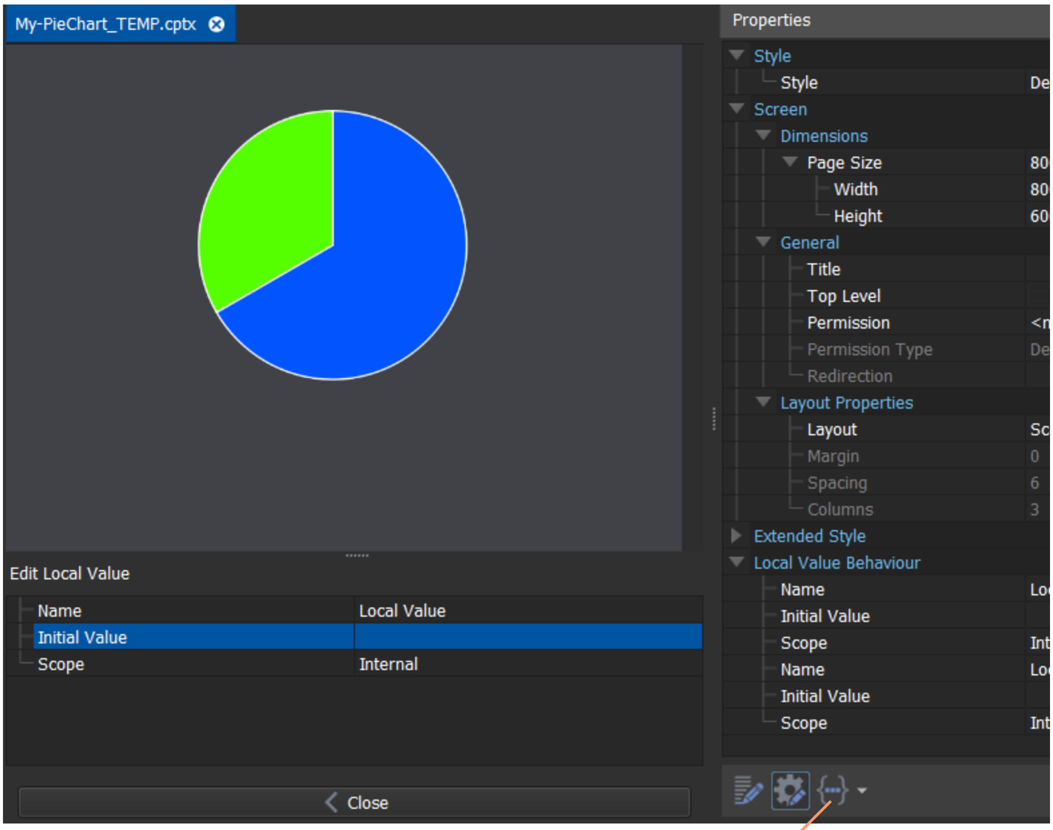
11 Click **OK**.

12 The **Local Value** which shall contain the ‘Maximum Data Capacity’ value is now configured.

Behaviour value property editing, Method 2:

13 Double-click on a **Local Value** Behaviour box in the ‘Behaviours and Bindings’ editor screen (see Figure 4-16 on page 94).

The **Edit Local Value** screen is shown, see Figure 4-19.



Edit Local Value

Name	Local Value
Initial Value	
Scope	Internal

Set Variable button

Fig. 4-19: Edit Local Value Screen

14 Enter a **Name** for the **Local Value**.

15 Click on the **Initial Value** item (as shown in Figure 4-19)
and click the **Set Variable** button.

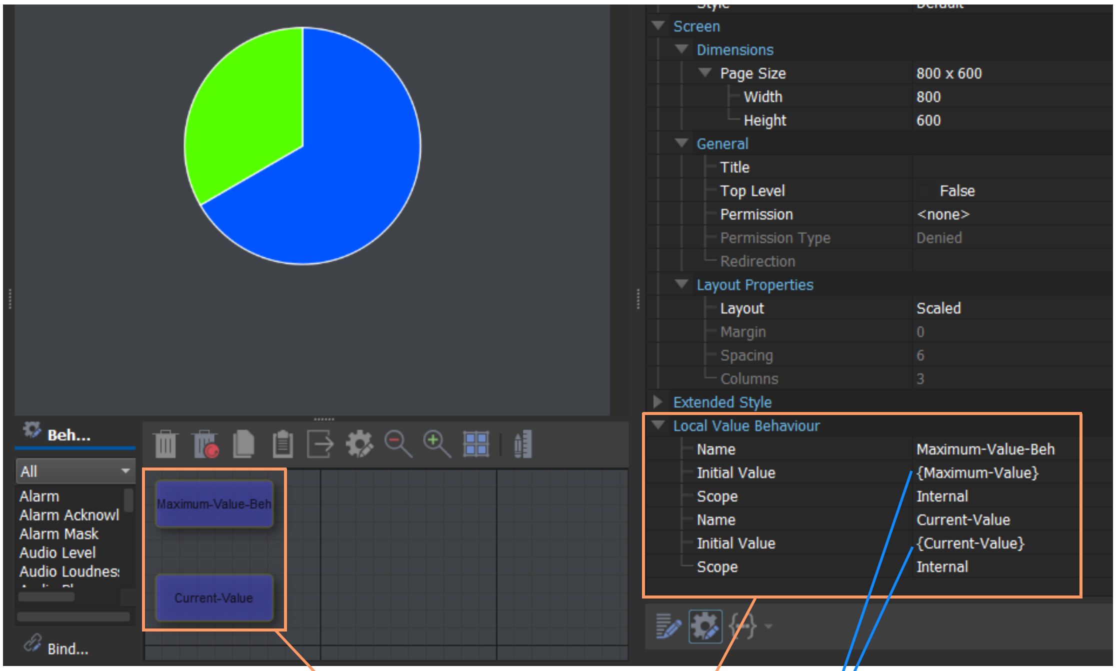
16 The **Select String Variable** dialog is shown (see Figure 4-18 on page 95).

17 For the **Local Value** Behaviour which shall contain the ‘Current Data Rate’ value in our example, select the ‘**Current-Value**’ item in the variables list shown in the **Select String Variable** dialog.

18 Click **OK**.

19 Click **Save**.

The **Local Value** Behaviours are now configured, linked to component variables. See Figure 4-20.



Screen	
Dimensions	
Page Size	800 x 600
Width	800
Height	600
General	
Title	
Top Level	False
Permission	<none>
Permission Type	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	
Local Value Behaviour	
Name	Maximum-Value-Beh
Initial Value	{Maximum-Value}
Scope	Internal
Name	Current-Value
Initial Value	{Current-Value}
Scope	Internal

Local Value Behaviour boxes

Local Value Properties

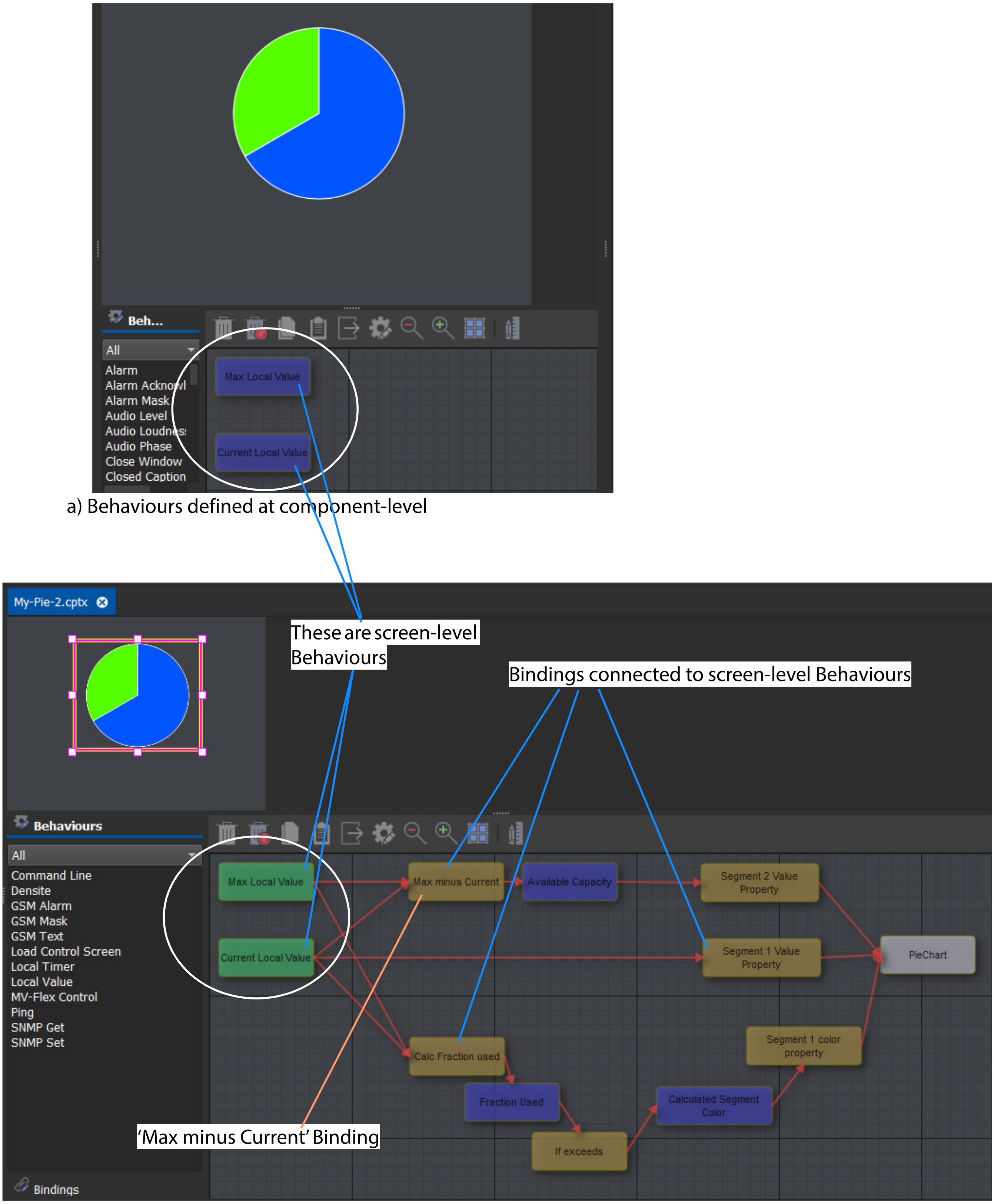
Component Variables, **{variable name}**

Fig. 4-20: Configured Local Value Behaviours

There are now Behaviours defined at ‘component-level’ on the component and at ‘widget-level’. See Figure 4-21a and b.

The next step is to connect up our widget-level items.

Connect Widget-Level Bindings to Component-Level Behaviours



a) Behaviours defined at component-level

These are screen-level Behaviours

Bindings connected to screen-level Behaviours

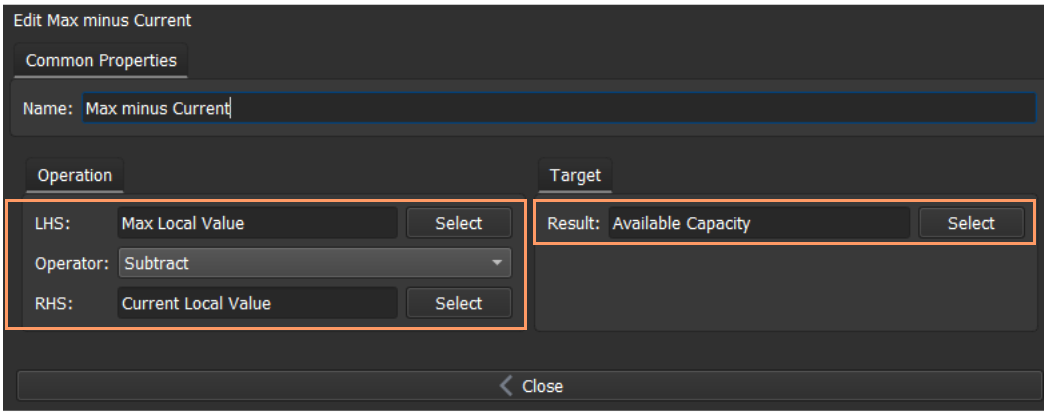
'Max minus Current' Binding

b) Behaviours defined at widget-level

Fig. 4-21: Component with:
a) Component-Level Behaviours.
b) Widget-Level Behaviours and Bindings.

1 Double-click on the 'Max minus Current' Binding (see Figure 4-21b) and edit the resulting dialog.

2 Change values to those shown in Figure 4-22a.



Edit Max minus Current

Common Properties

Name: Max minus Current

Operation

LHS: Max Local Value

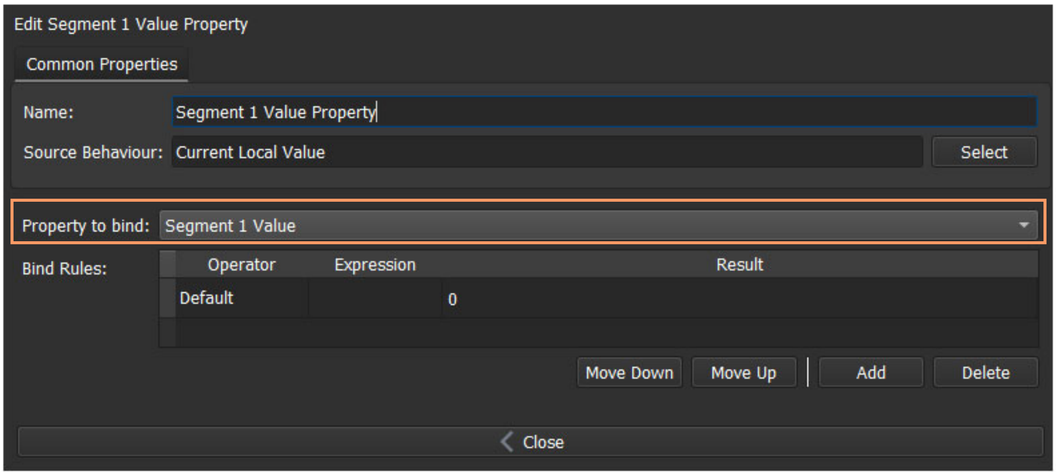
Operator: Subtract

RHS: Current Local Value

Target

Result: Available Capacity

a) 'Max minus Current' Binding Settings



Edit Segment 1 Value Property

Common Properties

Name: Segment 1 Value Property

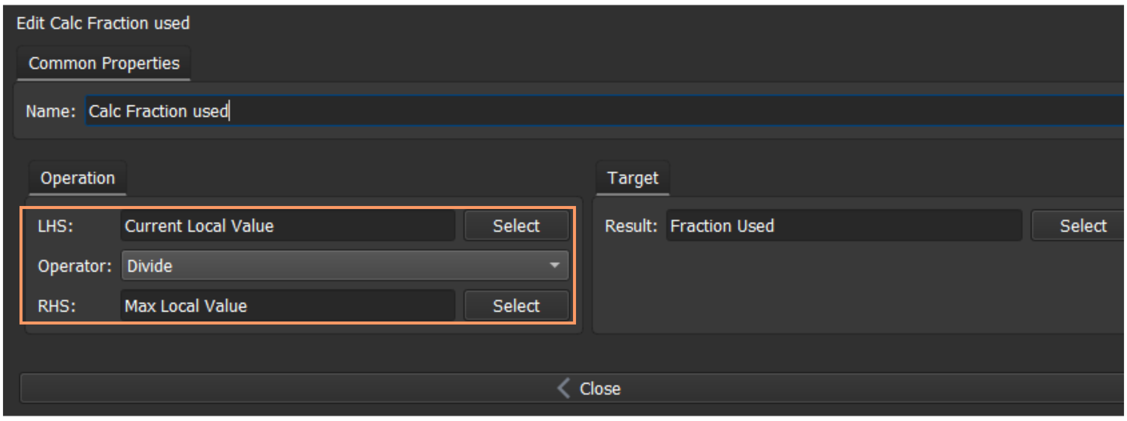
Source Behaviour: Current Local Value

Property to bind: Segment 1 Value

Bind Rules:

Operator	Expression	Result
Default		0

b) 'Segment 1 Value Property' Binding Settings



Edit Calc Fraction used

Common Properties

Name: Calc Fraction used

Operation

LHS: Current Local Value

Operator: Divide

RHS: Max Local Value

Target

Result: Fraction Used

c) 'Calc Fraction Used' Binding Settings

Fig. 4-22: Binding Settings:

a) 'Max minus Current' Binding.

b) 'Segment 1 Value Property' Binding.

c) 'Calc Fraction used' Binding.

3 Double-click on the 'Segment 1 Value property' Binding (see Figure 4-21b) and edit the resulting dialog.
Change values to those shown in Figure 4-22b.

4 Double-click on the 'Calc Fraction used' Binding (see Figure 4-21b) and edit the resulting dialog.
Change values to those shown in Figure 4-22c.

5 Click **Save**.

This has connected the widget-level items to component-level items. The resulting Behaviours and Bindings editor screen is shown in Figure 4-23.

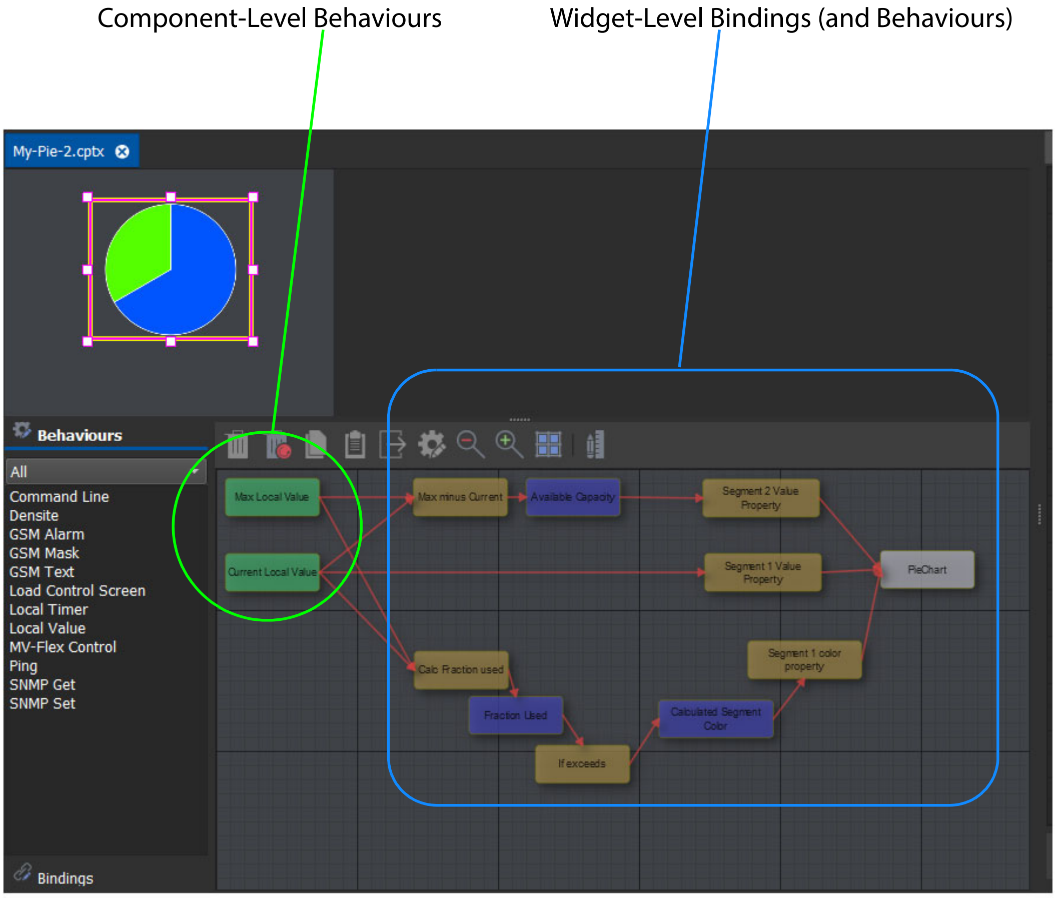


Fig. 4-23: Widget-Level Bindings Connected to Component-Level Behaviours

The component is now ready for use on a screen.

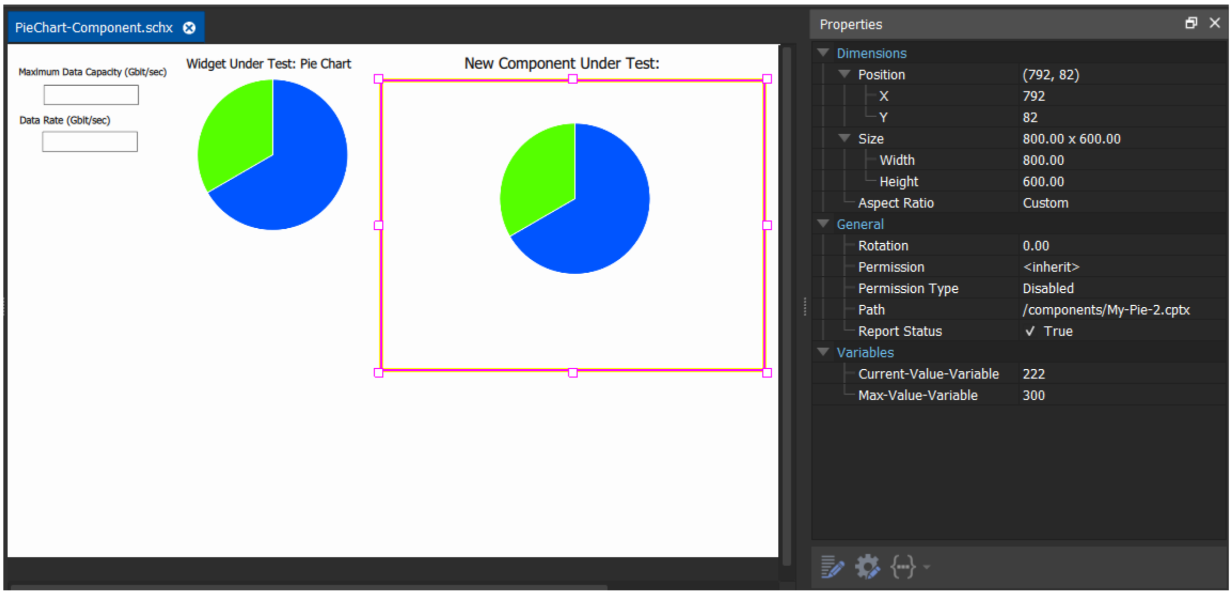
Instantiate the Component on a Screen

The new **Pie Chart** component is now ready for using on a screen. To exercise it, we shall add it to the test bench screen used in Exercising the Extra Functionality, on page 61.

1 Open the test bench screen.

2 To place a component on the screen, either:

- Drag the new **Pie Chart** *component* onto the screen schematic from the **Project View**, alongside the existing **Pie Chart** widget. See Figure 4-24.


PieChart-Component.schx
Maximum Data Capacity (Gbit/sec)
Data Rate (Gbit/sec)
Widget Under Test: Pie Chart
New Component Under Test:
Properties

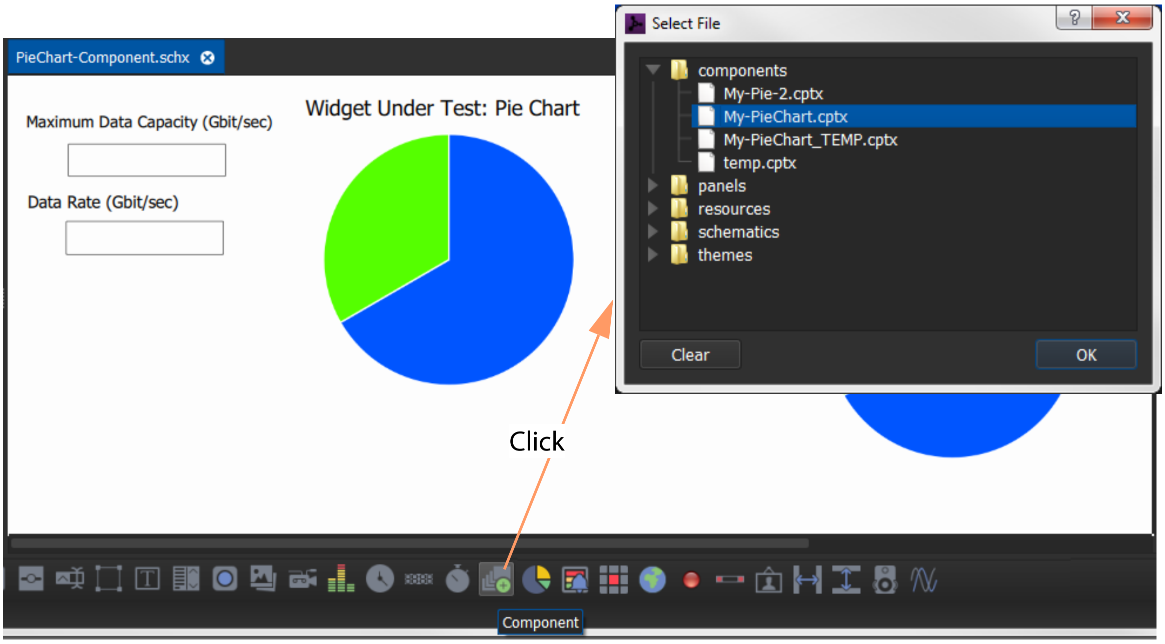
Property	Value
Dimensions	
Position	(792, 82)
X	792
Y	82
Size	800.00 x 600.00
Width	800.00
Height	600.00
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/My-Pie-2.cptx
Report Status	✓ True
Variables	
Current-Value-Variable	222
Max-Value-Variable	300

Fig. 4-24: New Pie Chart Component Used on a Screen

Or

- Click on the Component widget icon (see Figure 4-25), select the required component from the **Select File** dialog, and click **OK**.

Select File dialog



- components
  - My-Pie-2.cptx
  - My-PieChart.cptx
  - My-PieChart_TEMP.cptx
  - temp.cptx
- panels
- resources
- schematics
- themes

Click

Fig. 4-25: Component Widget Icon

3 Open the 'Behaviours and Bindings' graphical editor. (Click the **Edit** Behaviours button.)

4 Select the **Pie Chart** component.

5 Add two **Property** Bindings to the graphical editor screen. See Figure 4-26.

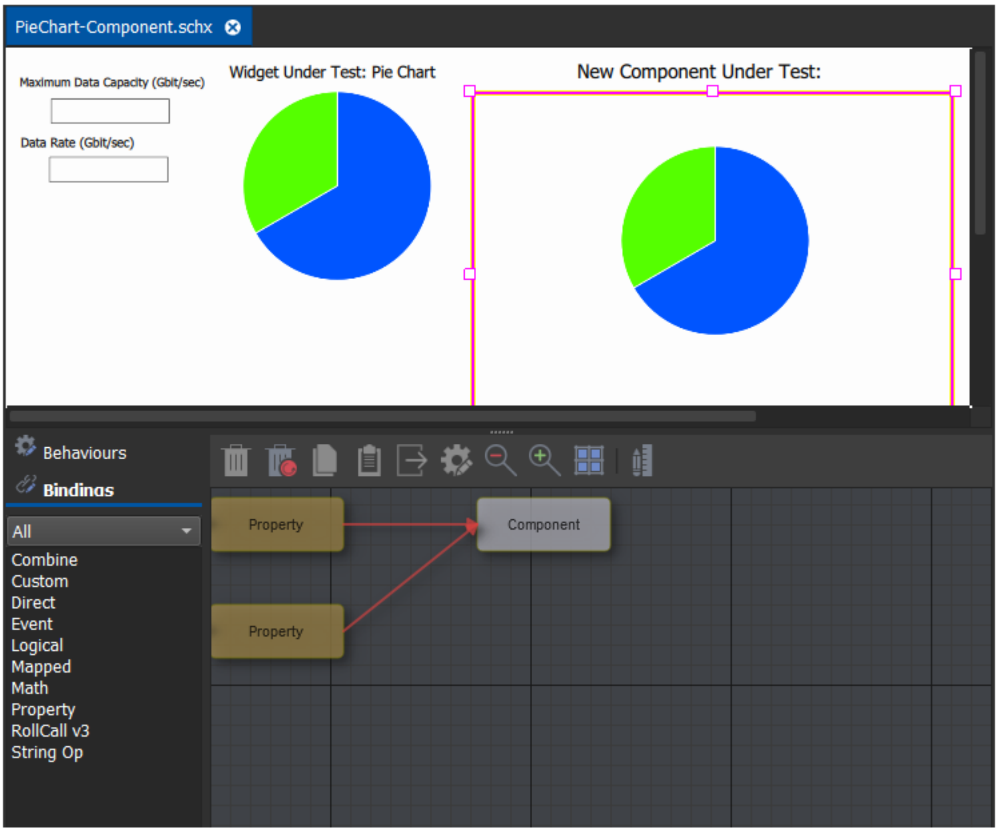


Fig. 4-26: Two Property Bindings Added

6 Double-click on one **Property** Binding box and set:

- 'Source Behaviour' to Max Data Capacity
- 'Property to Bind' to Max-Value-Variable (see Figure 4-27)

And then click **Add**.

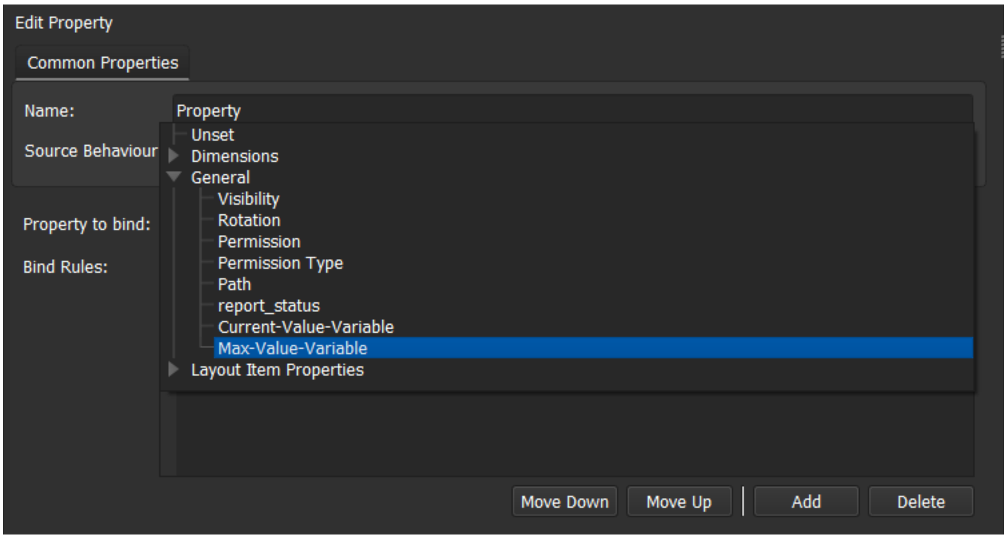


Fig. 4-27: Setting 'Property to Bind'

7 Double-click on the other Binding box to open the Edit Binding screen.

8 Set:

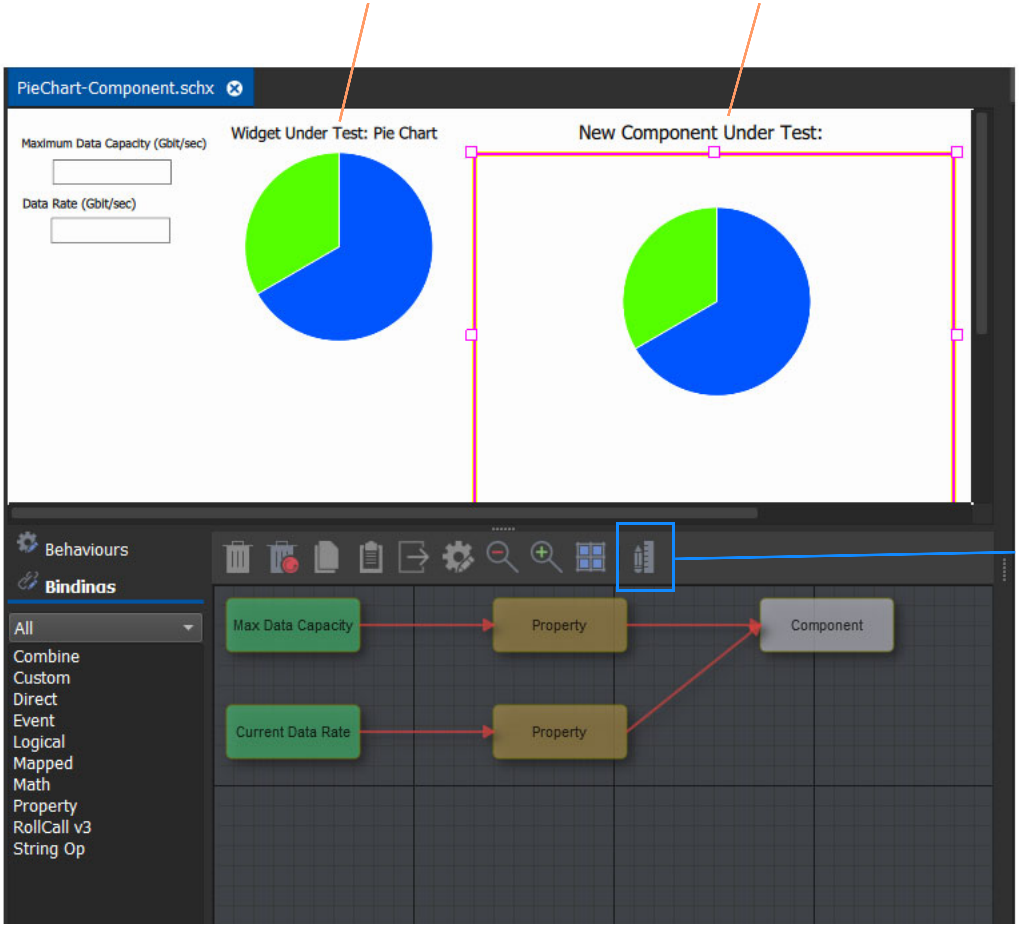
- 'Source Behaviour' to Current Data Rate
- 'Property to Bind' to Current-Value-Variable

And click **Add**.

9 Click **Save**.

The two component variables have now been connected to (bound to) screen-level Behaviours on our test bench screen. See Figure 4-28.

(The previous **Pie Chart** *widget* and our new **Pie Chart** *component* are shown side-by-side.)



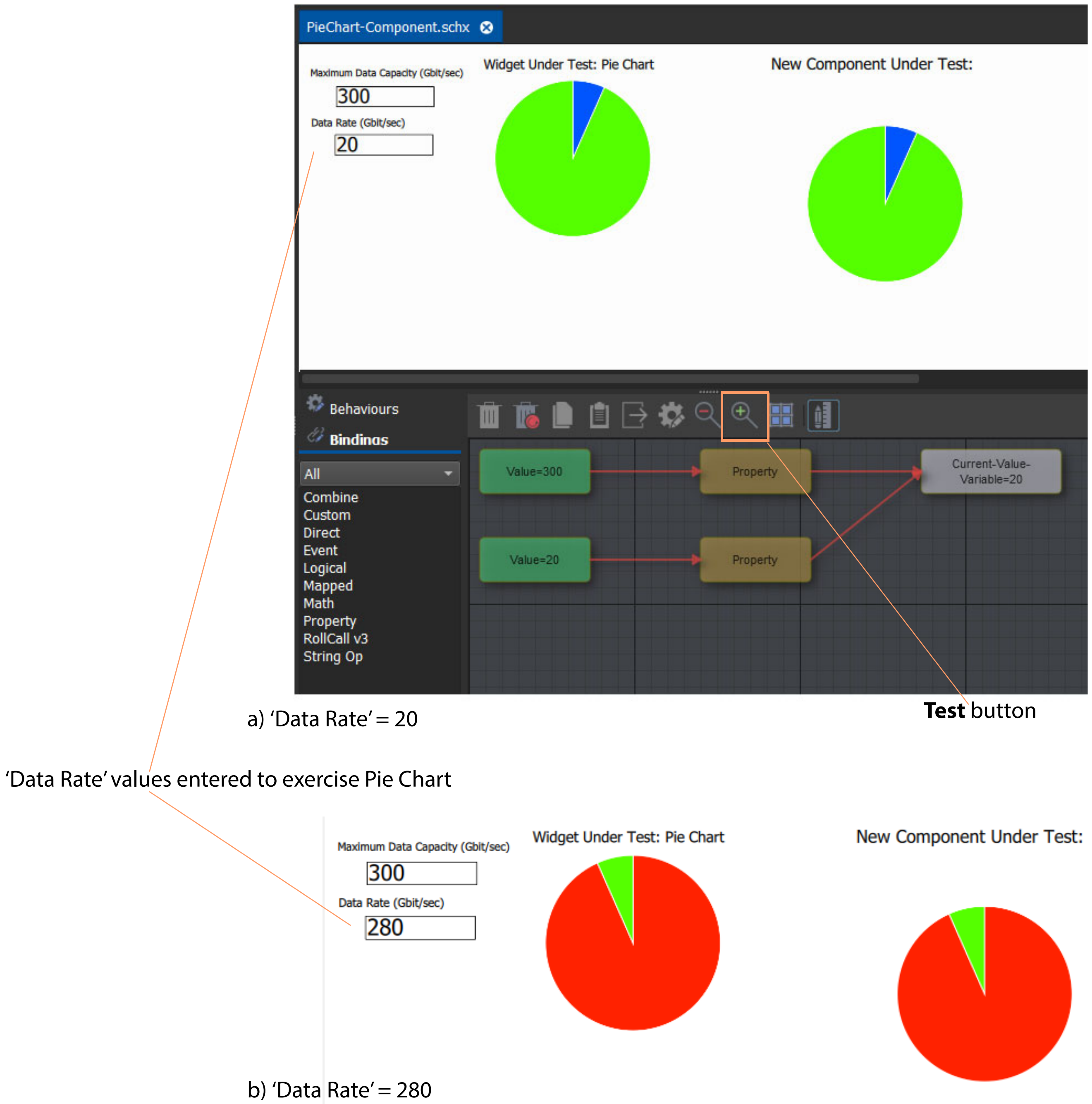
Test button

Fig. 4-28: Test Bench Screen - Showing Component Bindings

Exercise the Component

1 Click the **Test** button to exercise the test bench screen.

2 Enter values in the **Text Edit** widgets to exercise the previous **Pie Chart** widget and our new **Pie Chart** component side by side. See Figure 4-29.



PieChart-Component.schx

Maximum Data Capacity (Gbit/sec) 300

Data Rate (Gbit/sec) 20

Widget Under Test: Pie Chart
New Component Under Test:
Behaviours
Bindings
All
Combine
Custom
Direct
Event
Logical
Mapped
Math
Property
RollCall v3
String Op
Value=300
Property
Current-Value-Variable=20
Value=20
Property

a) 'Data Rate' = 20

Test button

'Data Rate' values entered to exercise Pie Chart

Maximum Data Capacity (Gbit/sec) 300

Data Rate (Gbit/sec) 280

Widget Under Test: Pie Chart
New Component Under Test:

b) 'Data Rate' = 280

Fig. 4-29: Exercising Pie Chart Component: a) 'Data Rate' = 20; b) 'Data Rate' = 280

Using Component Variables

Linking Component Variables to Widget Property Values

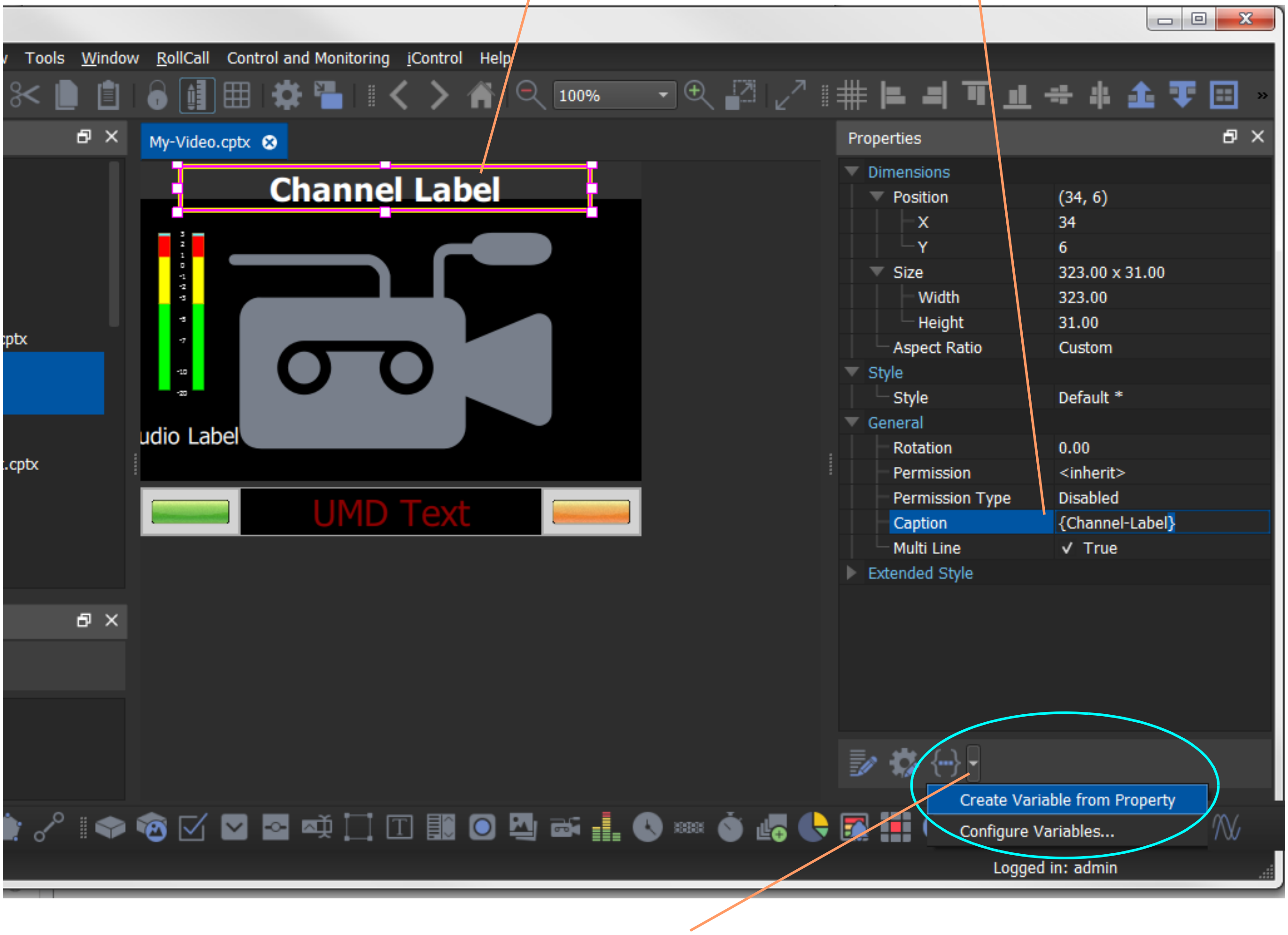
To link widget property values (for example, **Label** widget caption property values) to component variables:

1 Open the 'My-Video' component in Orbit in Design mode.

2 Select the 'Channel Label' Label widget (Figure 4-30a) and select its 'Caption' property value in the Properties box (Figure 4-30b).

a) Select 'Label' widget to see the widget's properties.

b) Select the widget's 'Caption' property value



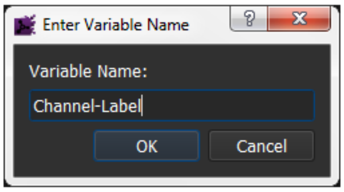
Dimensions	
Position	(34, 6)
X	34
Y	6
Size	323.00 x 31.00
Width	323.00
Height	31.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{Channel-Label}
Multi Line	✓ True
Extended Style	

c) Click down-arrow and select 'Create Variable from Property'

Fig. 4-30: Label Widget Caption Property Value:

3 Click the small down-arrow at the **Set Variable** icon and select 'Create Variable from Property'. (Figure 4-30c).

4 Enter a variable name in the **Enter Variable Name** dialog (see Figure 4-31) and click **OK**.



Variable Name: Channel-Label

Fig. 4-31: Enter Variable Name Dialog

Now deal with the Audio Label widget:

5 Select the ‘Audio Label’ widget
and select its ‘Caption’ property value in the Properties box.

6 Click the Set Variable down-arrow icon
and select ‘Create Variable from Property’.

7 Enter a variable name (‘Audio-Label’) in the **Enter Variable Name** dialog and click OK.

8 Click **Save**.

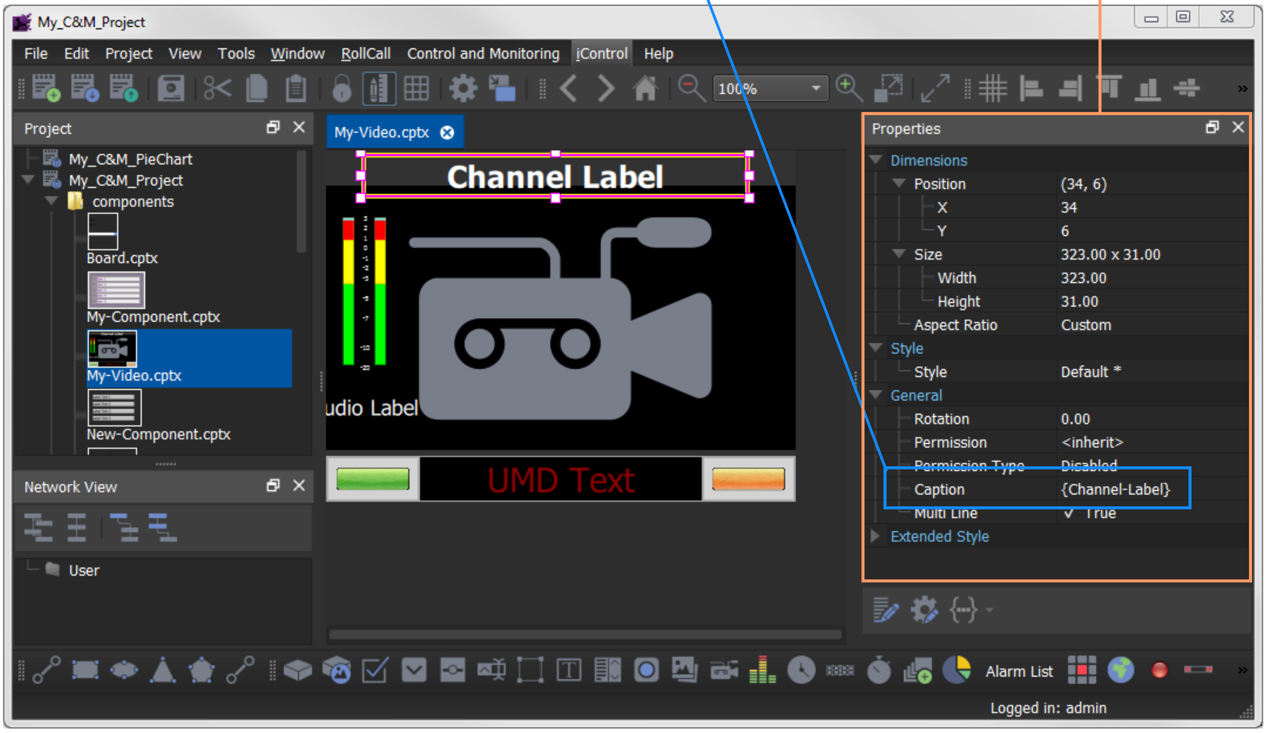
Widget caption property values are now linked to component variables:

- ‘Channel Label’ widget caption linked to variable ‘Channel-Label’; ‘and
- ‘Audio Label’ widget caption linked to ‘Audio-Label’.

This can be verified by selecting the widget and inspecting its Caption property value in the **Properties** box; for property values set to a variable, the value shows the variable name within curly brackets. See Figure 4-32.

Caption property value set to component variable, {Channel-Label}

Properties box



Dimensions	
Position	(34, 6)
X	34
Y	6
Size	323.00 x 31.00
Width	323.00
Height	31.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{Channel-Label}
Multi Line	✓ True
Extended Style	

Fig. 4-32: Widget Property Value Set to a {Variable}

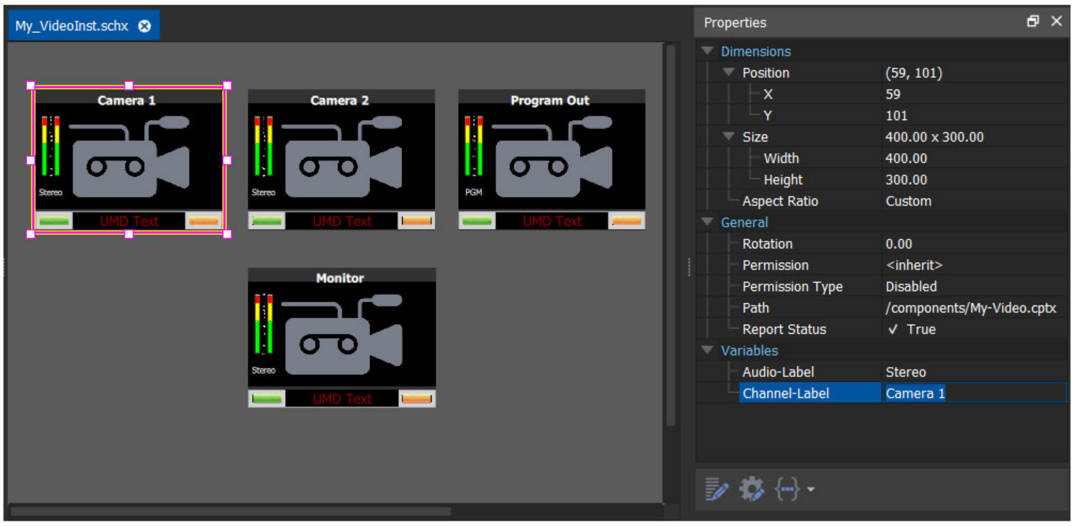
Setting Component Property Values at Screen-Level

Return to our screen with several instances of our 'My-Video' component, see Figure 4-3.

1 Select one component on the screen.

2 Under the 'Variables' heading in the **Properties** box:

- Select the 'Channel-Label' property and change its value to 'Camera 1'. See Figure 4-33.
- Edit the 'Audio-Label' property value.



Properties

Dimensions	
Position	(59, 101)
X	59
Y	101
Size	400.00 x 300.00
Width	400.00
Height	300.00
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/My-Video.cptx
Report Status	✓ True
Variables	
Audio-Label	Stereo
Channel-Label	Camera 1

Fig. 4-33: Changing Value of a Component Variable at the 'Screen-Level'

3 Repeat these steps for each component, setting different label values for each.

Each of the components on the screen can now have different settings for their 'Audio Label' and 'Channel Label' widgets. See Figure 4-33.

Component with Multiple Variables

Creating New Component

1 Right-click on the 'components' item in the **Project View** pane of the Orbit MapView screen.
And select **New Component**.

2 Enter a component **Name** in the dialog and click OK.

A new, blank component sheet is opened in the graphical editor.

3 Check that you are in 'Design mode' in Orbit.

4 Add widgets to the component sheet.
The example shown in Figure 4-34 contains four 'Label' widgets.

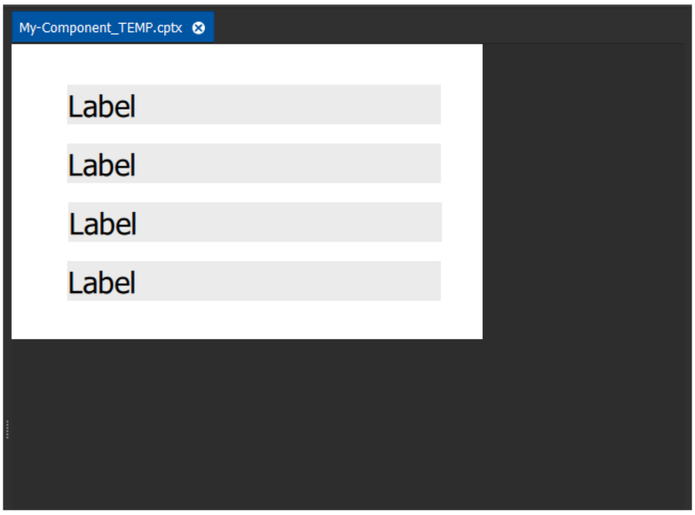


Fig. 4-34: 'New-Component'

5 Click **Save**.

Creating Multiple Component Variables

The component can be used on other screens. However, the label text is fixed at this point. To make the label text editable each time the component is used in a screen, the label text must be linked to a component variable.

To link label caption text to component variables:

1 Open the component in Orbit.

2 Right-click on the component *background* and select 'Variables'. See Figure 4-35.

A 'Variables' dialog is shown.

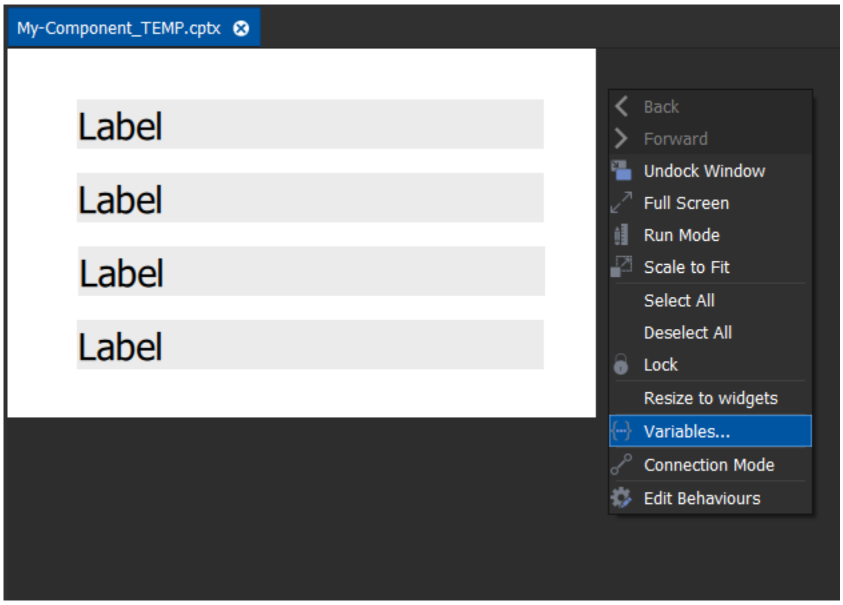
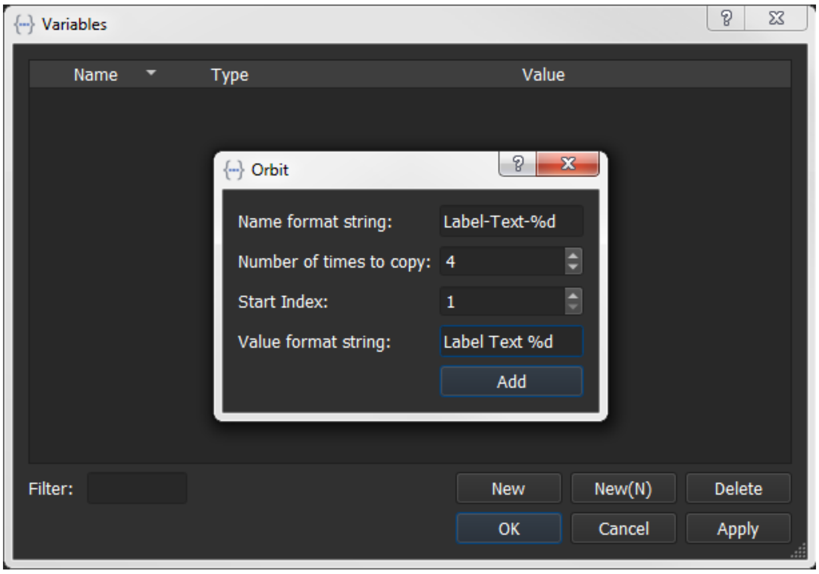


Fig. 4-35: New-Component: Right-click on Component Background and Select 'Variables'

3 In the 'Variables' dialog, click the **New(N)** button.
An Orbit new variable dialog is shown.

4 Set up the new variable dialog as shown in Figure 4-36.
And click **Add**.

Figure 4-37 shows the resulting added component variables (Label-Text-1,... , Label-Text-4).



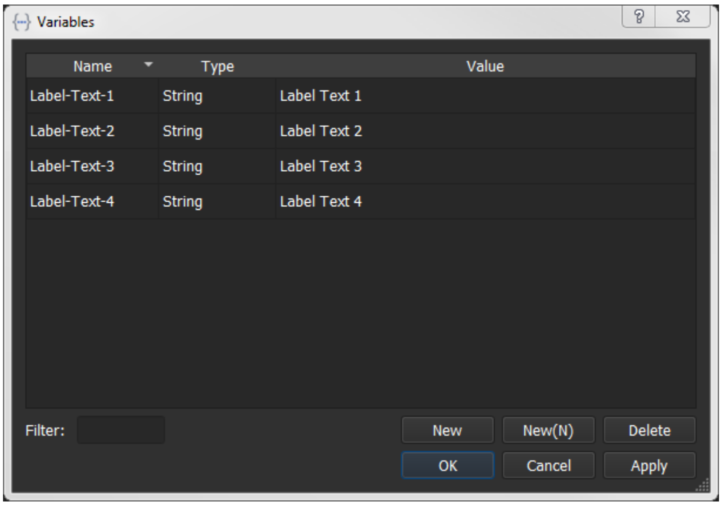
Name format string: Label-Text-%d

Number of times to copy: 4

Start Index: 1

Value format string: Label Text %d

Fig. 4-36: Orbit Variables and New Variable Dialogs



Variables

Name	Type	Value
Label-Text-1	String	Label Text 1
Label-Text-2	String	Label Text 2
Label-Text-3	String	Label Text 3
Label-Text-4	String	Label Text 4

Fig. 4-37: Variables Dialog with Added Component Variables

5 Click **Save**.

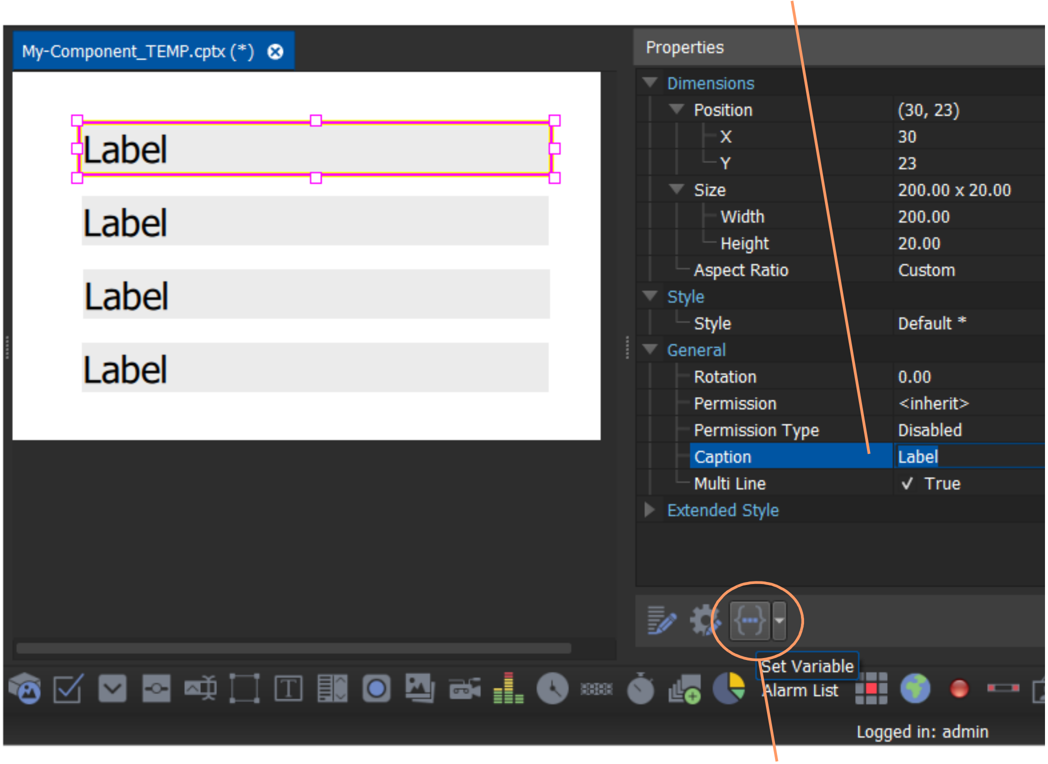
Four new component variables have been added.

Linking Component Variables to Widget Properties

To link these variables to the captions of each of the Label widgets:

1 Select the first **Label** widget on the component; and then select its 'Caption' property value in the **Properties** box. See Figure 4-38.

Caption property selected

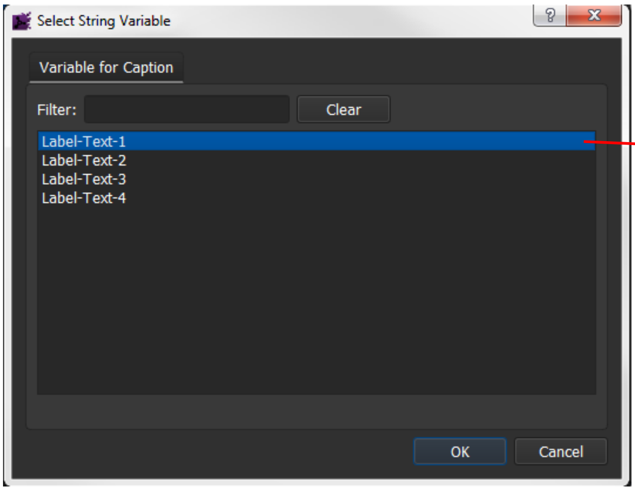


Properties	
Dimensions	
Position	(30, 23)
X	30
Y	23
Size	200.00 x 20.00
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	Label
Multi Line	✓ True
Extended Style	

Set Variable icon

Fig. 4-38: New-Component: Selected Label Widget and Caption Property Value

2 Click the **Set Variable** icon.
The **Select String Variable** dialog is shown. See Figure 4-39.



Label-Text-1
Label-Text-2
Label-Text-3
Label-Text-4

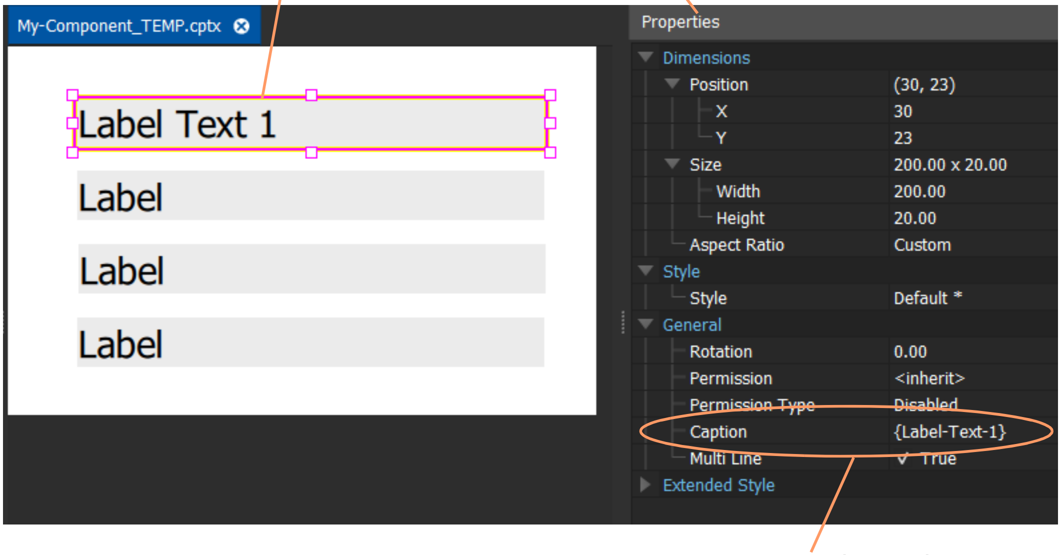
Select first item

Fig. 4-39: Select String Variable Dialog

3 Click on the first item in the list shown. See Figure 4-39.
And click **OK**.

The first **Label** widget's 'Caption' property value is now linked to the first component variable. This is shown in the properties of the **Label** widget, see Figure 4-40.

Selected **Label** widget and its properties



Dimensions	
Position	(30, 23)
X	30
Y	23
Size	200.00 x 20.00
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{Label-Text-1}
Multi Line	✓ True
Extended Style	

Caption property value set to component variable, {Label-Text-1}

Fig. 4-40: New-Component: Label Widget Caption Property Value Linked to a Component Variable

4 Repeat from step 1for the other Label widgets.
Link widget caption property values to component variables:
{Label-Text-2};
{Label-Text-3}; and
{Label-Text-4}.

5 Click **Save**.

The component may now be used on an Orbit MapView screen.

Using a Component with Multiple Variables

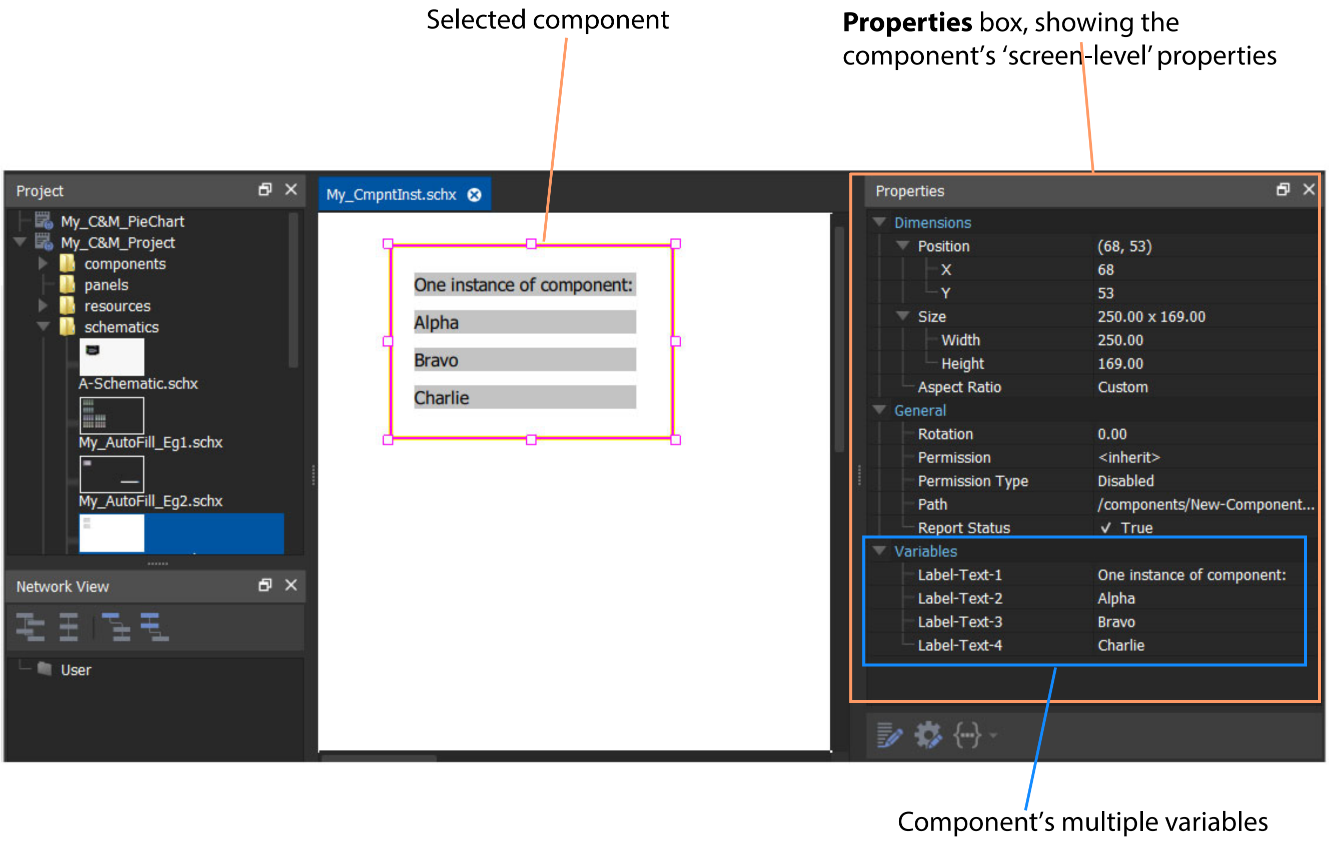
1 Open an Orbit MapView screen.

2 Check that you are in ‘Design mode’ in Orbit.

3 Expand the ‘components’ item in the **Project View** pane.

4 Drag on our newly-made component, ‘New-Component’, from Creating New Component, on page 108.

5 Select the component on the screen.
The screen-level properties of the component are listed in the **Properties** box. See Figure 4-41. This includes the component variables.



Selected component

Properties box, showing the component’s ‘screen-level’ properties

Component’s multiple variables

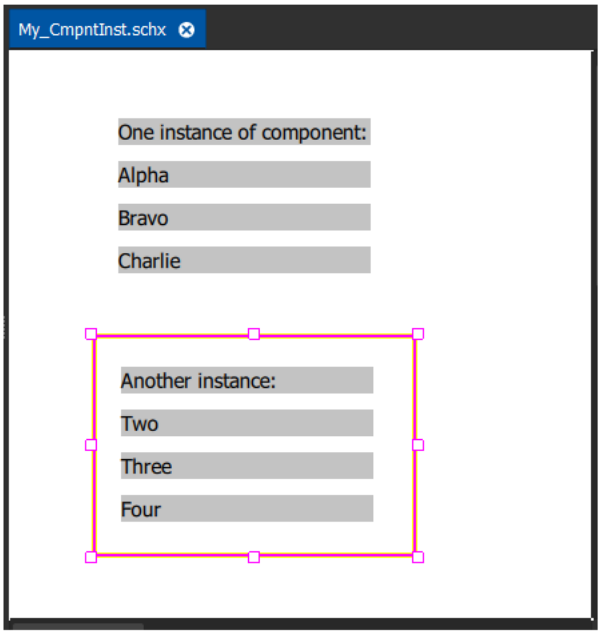
Dimensions	
Position	(68, 53)
X	68
Y	53
Size	250.00 x 169.00
Width	250.00
Height	169.00
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/New-Component...
Report Status	✓ True
Variables	
Label-Text-1	One instance of component:
Label-Text-2	Alpha
Label-Text-3	Bravo
Label-Text-4	Charlie

Fig. 4-41: Component Variables on a Screen

6 The component’s variables may be set in the **Properties** box.
this changes the variables for this instance of the component on this screen.

7 Click **Save**.

Another copy of the same component can be dragged onto the screen and its variables can be independently set. Thus two or more instances of the component are possible. See Figure 4-42.



One instance of component:

Alpha

Bravo

Charlie

Another instance:

Two

Three

Four

Fig. 4-42: Two Instances of a Component on a Screen - with Different Component Variables Values

Binding to Component *Behaviour* Values

Behaviour values within a component can be used when the component is placed on a screen. Bindings at screen-level may bind to component-level Behaviour values. Placing logic on a component keeps the logic contained with the component and avoids unnecessary duplication at screen-level.

Note: Logic defined in a component on a screen is only exercised when the screen is open. It should be used for processing required when the screen is open. For example:

- an action as a result of clicking a button on a screen; or
- logic to determine some state to be shown when the screen is open.

If logic needs to run independently of a screen being open, then implement the logic in a GlobalX file.
(See Orbit Global Files for Server-Side Processing, on page 282.)

For example, use this feature to contain the logic required to decide if a component is correctly configured. A itself component is best-placed to decide whether it is correctly configured. (This is specific to the component and may be a complex decision process.) The result of this decision will be stored in a Behaviour on the component. And the Behaviour value can be used at screen-level, for example, to hide a component when it is not configured.
The example following shows this.

Example - Controlling Component Visibility

This example generates a component which has an address variable. The variable is visible at screen-level. The variable must be configured with a valid RollCall address, otherwise the component should be hidden on screens where it is used.

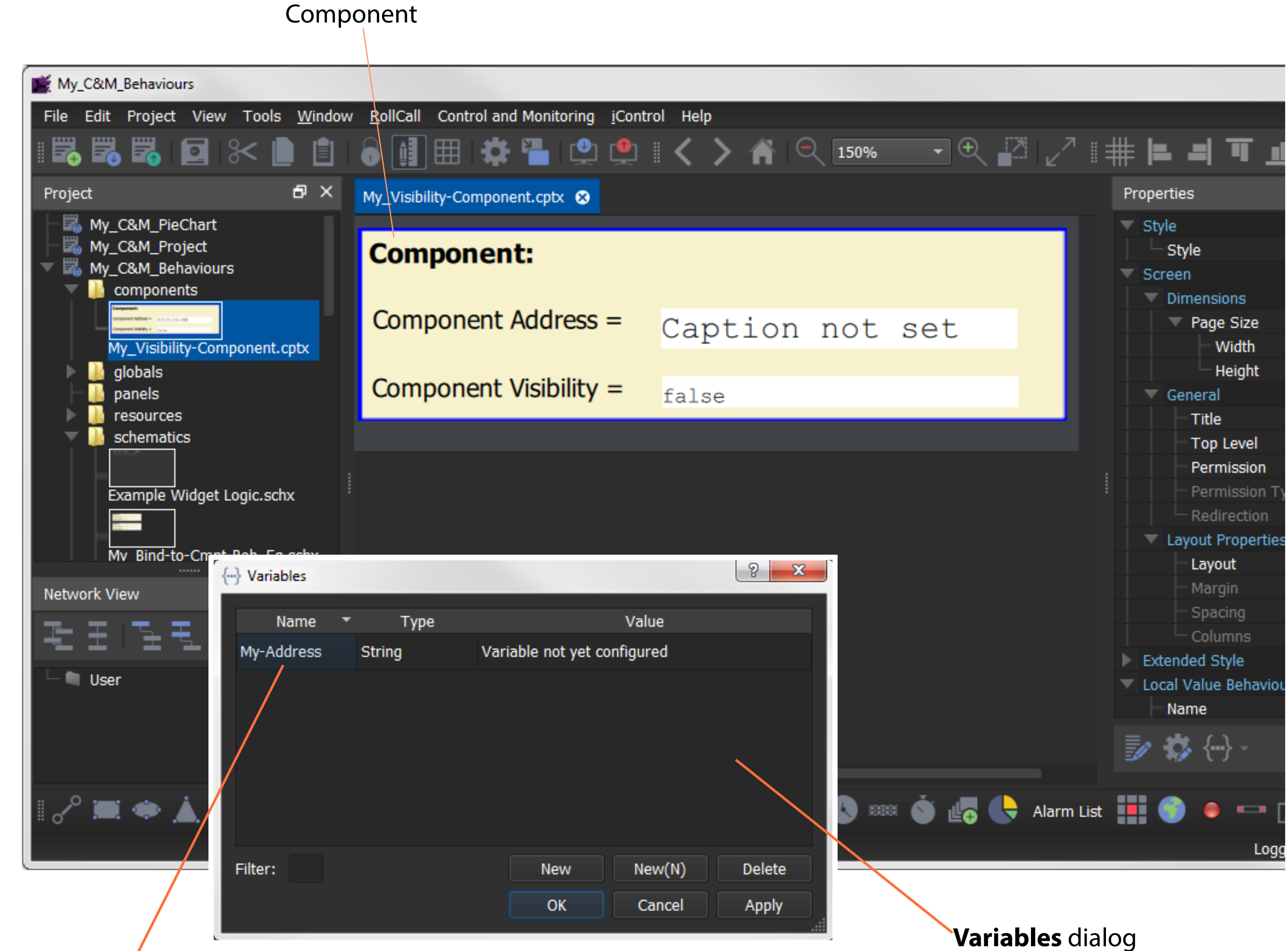
The RollCall address format is NNNN:UU:PP, where N, U and P are hexadecimal digits.

A component is placed onto a screen with a **Component** widget and this widget has a visibility property. A Binding is used to bind to this widget property on the screen and a Behaviour value from *inside* the component is used to control widget visibility.

Create a Component

1 Create a Component (‘My_Visibility-Component’).

2 Right-click on the component background and select ‘Variables...’ .
The **Variables** dialog is shown. See Figure 4-43.



Component

Component variable

Variables dialog

My_Visibility-Component.cptx

Component:

Component Address = Caption not set

Component Visibility = false

Name	Type	Value
My-Address	String	Variable not yet configured

Fig. 4-43: Created Component and its Variables Dialog

3 Click **New** to add a variable (name ‘My-Address’, type ‘String’).

4 Click **OK** to close the dialog.

5 Add a **Local Value** Behaviour, double-click on it and set:

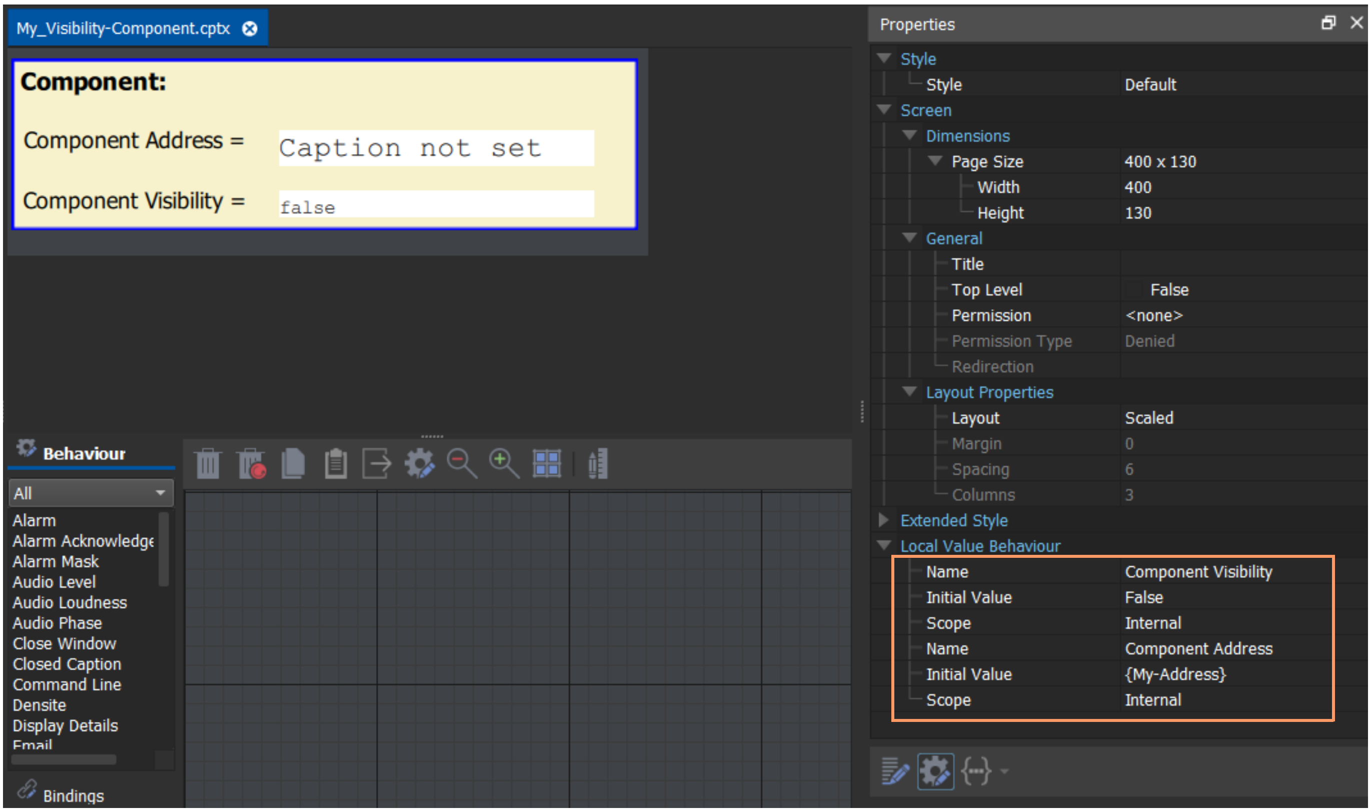
- Name to 'Component Visibility'.
- Initial value to 'False'.

6 Add a **Local Value** Behaviour and set:

- Name to 'Component Address'.
- Initial value to '{My-Address}'.

See Figure 4-44.

:



Component:

Component Address = Caption not set

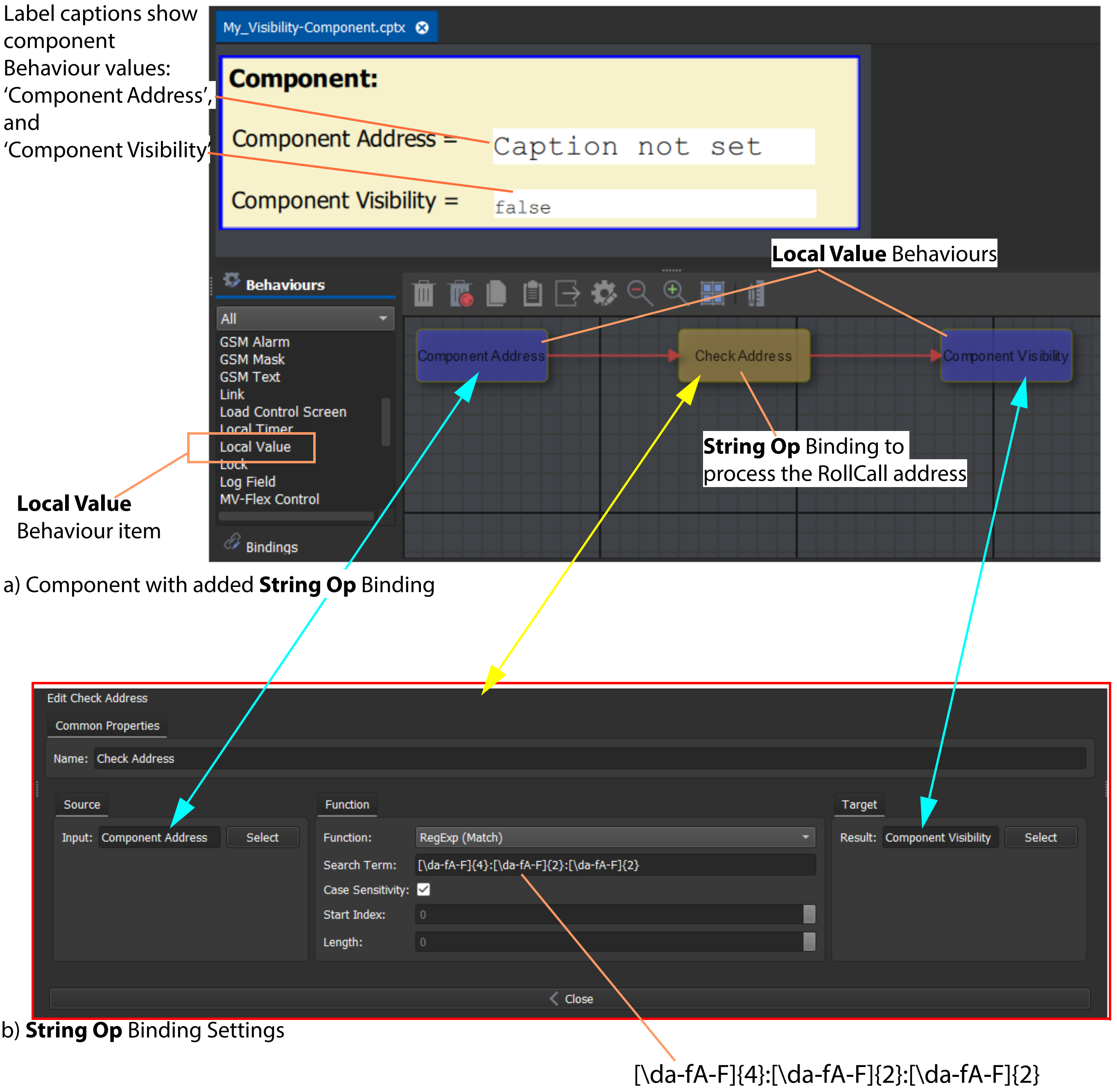
Component Visibility = false

Style	
Style	Default
Screen	
Dimensions	
Page Size	400 x 130
Width	400
Height	130
General	
Title	
Top Level	False
Permission	<none>
Permission Type	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	
Local Value Behaviour	
Name	Component Visibility
Initial Value	False
Scope	Internal
Name	Component Address
Initial Value	{My-Address}
Scope	Internal

Fig. 4-44: Component Variable 'Component Address' set to {My-Address}

7 Add a **String Op** Binding (see Figure 4-45a).
And configure Binding settings as shown in Figure 4-45b).

:


Label captions show component Behaviour values: 'Component Address', and 'Component Visibility'
My_Visibility-Component.cptx
Component:
Component Address =
Caption not set
Component Visibility =
false
Local Value Behaviours
Behaviours
All
GSM Alarm
GSM Mask
GSM Text
Link
Load Control Screen
Local Timer
Local Value
Lock
Log Field
MV-Flex Control
Bindings
Component Address
Check Address
Component Visibility
String Op Binding to process the RollCall address
Local Value Behaviour item
[\da-fA-F]{4}:[\da-fA-F]{2}:[\da-fA-F]{2}

a) Component with added **String Op** Binding

Edit Check Address

Common Properties

Name: Check Address

Source

Input: Component Address [Select]

Function

Function: RegExp (Match)

Search Term: [\da-fA-F]{4}:[\da-fA-F]{2}:[\da-fA-F]{2}

Case Sensitivity: ☑

Start Index: 0

Length: 0

Target

Result: Component Visibility [Select]

Close

b) **String Op** Binding Settings

Fig. 4-45: My Component:
a) Added String Op Binding.
b) String Op Binging Settings.

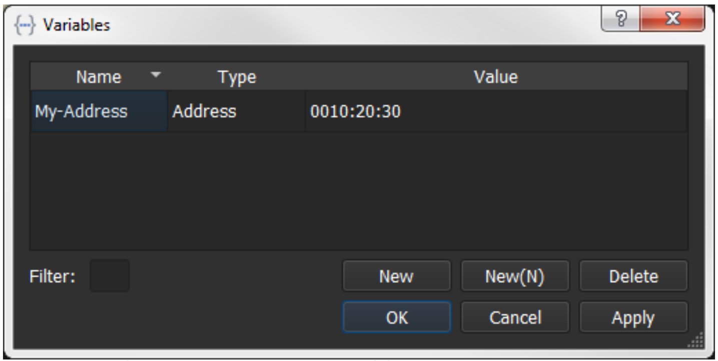
8 Click **Save**.

Exercise the Component for a Valid RollCall Address

Open the component.

1 Right-click in the component's background
and select 'Variables...' to display the **Variables** dialog.

2 In the **Variables** dialog, change the value of 'My-Address' to a valid RollCall address (for example, 0010:20:30).



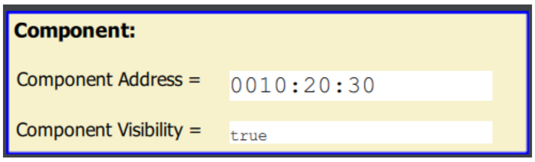
Name	Type	Value
My-Address	Address	0010:20:30

Fig. 4-46: Editing Component Variable

3 Click **OK**.

4 Click on the **Test** button to exercise the component.

The resulting 'Component Visibility' Behaviour value should be 'True' because we have a valid RollCall address.



Component:

Component Address = 0010:20:30

Component Visibility = true

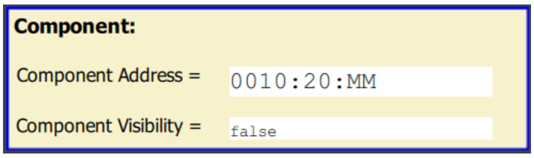
5 Click on the **Test** button to stop the exercise.

Exercise the Component for an Invalid RollCall Address

1 Repeat the steps above but with an invalid RollCall address (for example, 0010:20:MM)

2 Click on the **Test** button to re-exercise the component with the new address value.

The 'Component Visibility' Behaviour value should be 'False' because we have an invalid RollCall address.



Component:

Component Address = 0010:20:MM

Component Visibility = false

3 Click on the **Test** button to stop the exercise.

4 *Do not* save these changes to the component.

Exercising is complete.

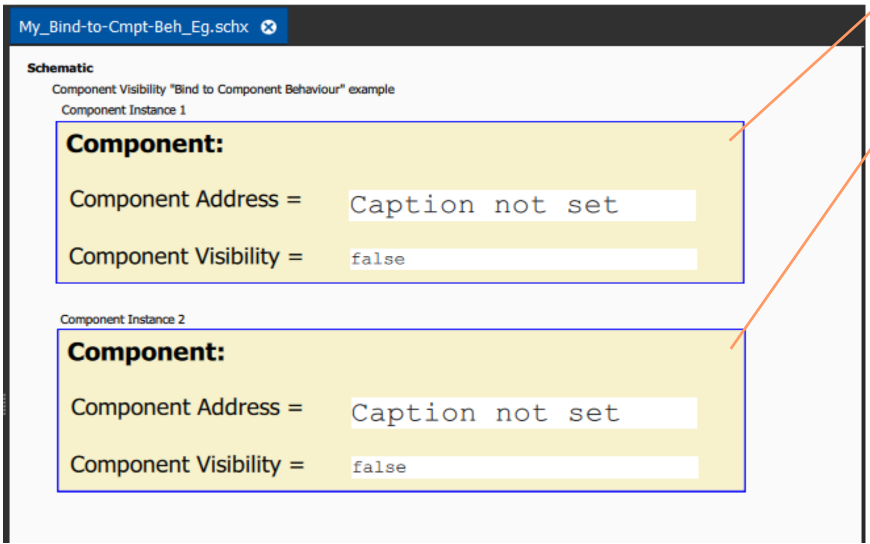
Using the Component on a Screen

Create a Screen and Instantiate the Component

1 Create a screen.

2 Add two instances of the Component. See Figure 4-47.

Component instance 1 and 2



My_Bind-to-Cmpt-Beh_Eg.schx

Schematic
Component Visibility "Bind to Component Behaviour" example
Component Instance 1

Component:
Component Address = Caption not set
Component Visibility = false

Component Instance 2

Component:
Component Address = Caption not set
Component Visibility = false

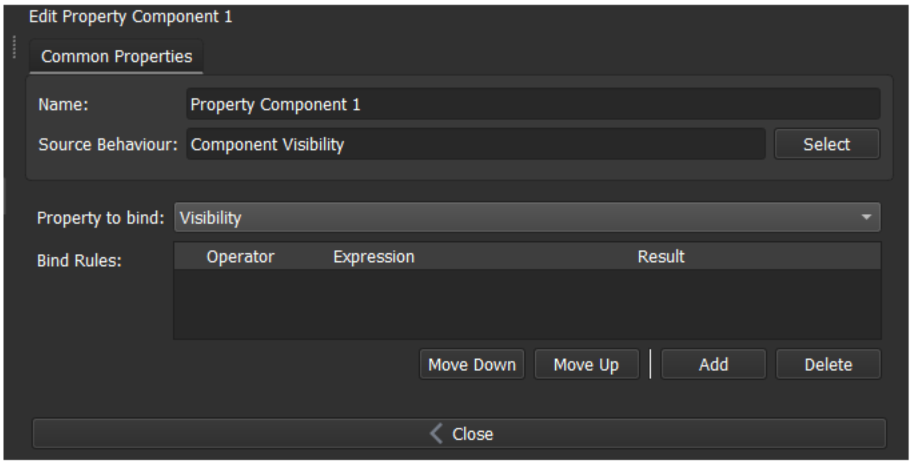
Fig. 4-47: Created Screen

Set Up Component Instance 1

Bind to the Component's Visibility Property

1 Select component instance 1 and show the 'Behaviour and Binding' graphical editor.

2 Add a **Property** Binding and view its properties; see Figure 4-48.



Edit Property Component 1

Common Properties

Name: Property Component 1
Source Behaviour: Component Visibility

Property to bind: Visibility

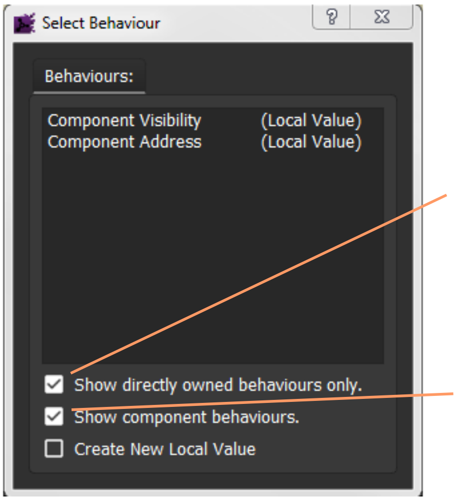
Bind Rules:

Operator	Expression	Result

Fig. 4-48: Edit Property Binding Settings (for Component Instance 1)

To select the **Property** Binding's **Source** Behaviour:

3 Click **Select** for the **Source Behaviour** item.
The **Select Behaviour** dialog is shown. See Figure 4-49.



Component Visibility (Local Value)
Component Address (Local Value)

☑ Show directly owned behaviours only.
☑ Show component behaviours.
□ Create New Local Value

Select to show the Behaviours belonging to the selected Component

Select to show all component Behaviours

Fig. 4-49: Select Source Behaviour Dialog

4 In the **Select Behaviour** dialog:
   - select the **Show Directly Owned Behaviours Only** option; and
   - select the **Show Component Behaviours** option.

5 Click on the 'Component Visibility' Behaviour item in the resulting list in the dialog. See Figure 4-49.
(This is the 'Component Visibility' Behaviour of the selected component.)
The **Select Behaviour** dialog closes.

This has selected the component's 'Component Visibility' Behaviour as Source Behaviour for the **Property** binding.

To select 'Property to Bind', in the Edit Binding Properties box (Figure 4-48 on page 119):

6 Select 'Visibility' as the 'Property to Bind'.

For 'Bind Rules',

7 There should be no Bind Rules. This means that the Source Behaviour's value is used directly.

8 Click **Close**.

Set Up RollCall Address Specifically for Component Instance

To configure the RollCall address parameter specifically on each component instance:

1 Select the first component on the screen schematic.

2 In the **Properties** box, change the value of variable 'My-Address'.
For example, use '0010:20:MM'; see Figure 4-50a. This is an invalid address.

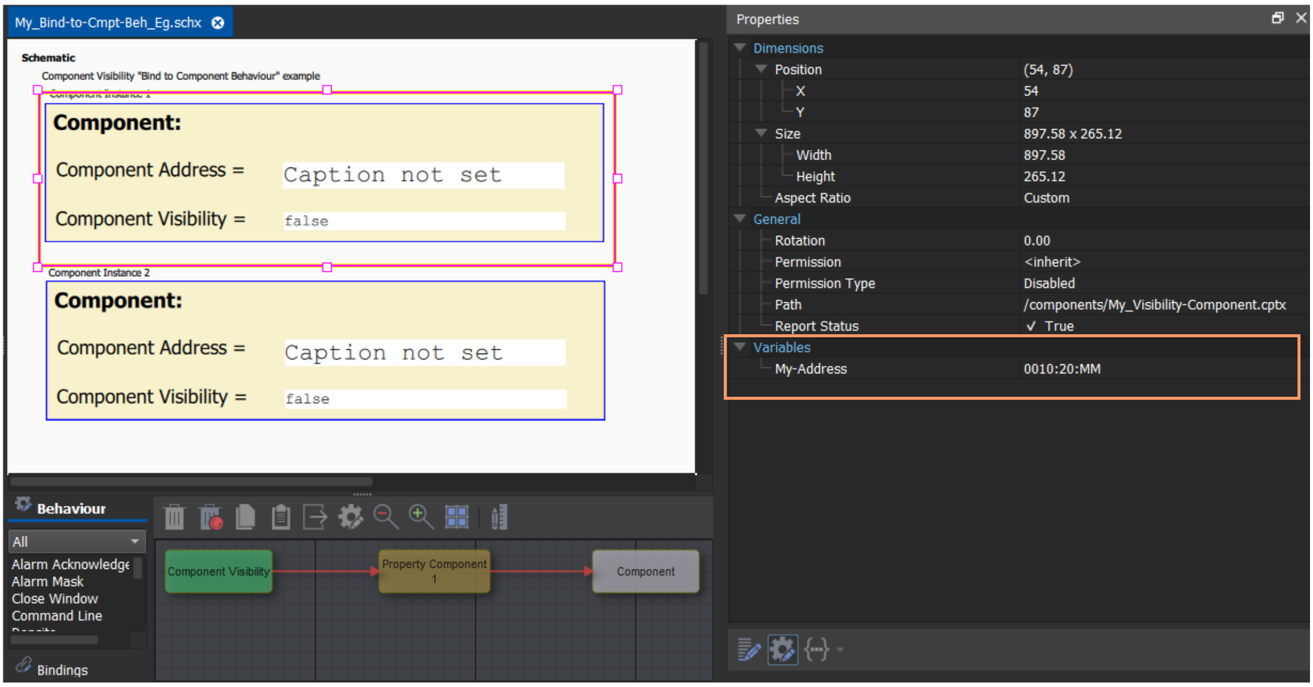
3 Click **Save**.

The component instance now uses a Behaviour value from *within* component instance to control the visibility of the component instance.

Set up Component Instance 2

1 Select the second component instance
and do the same as for instance 1, except use a different variable 'My-Address' value.
(For example, use '0010:20:30'; see Figure 4-50b.)

2 Click **Save**.



My_Bind-to-Cmpt-Beh_Eg.schx

Schematic

Component Visibility "Bind to Component Behaviour" example

Component Instance 1

Component:

Component Address = Caption not set

Component Visibility = false

Component Instance 2

Component:

Component Address = Caption not set

Component Visibility = false

Behaviour

All

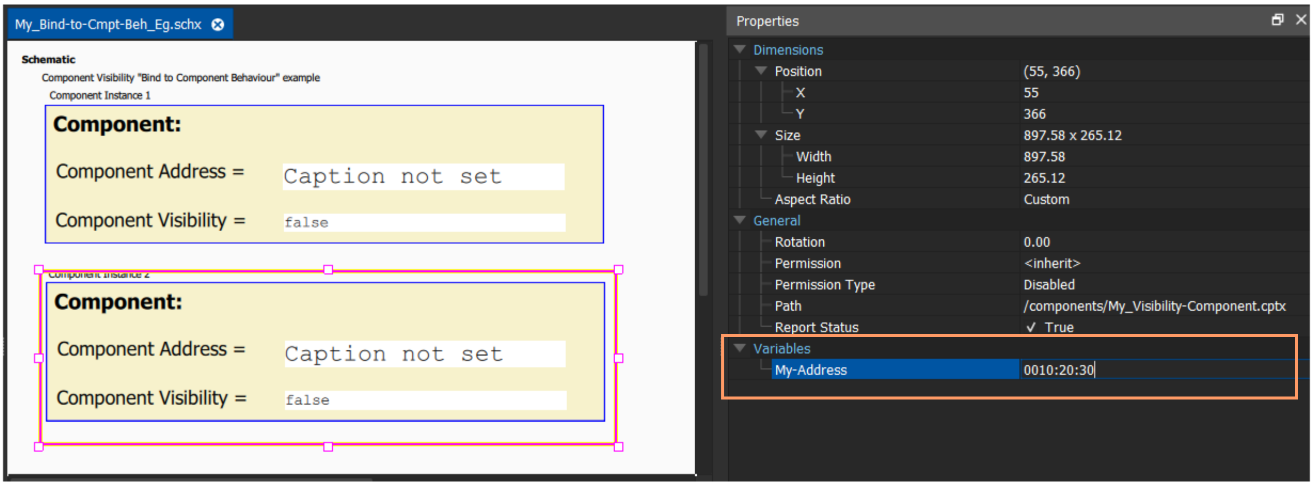
Alarm Acknowledge
Alarm Mask
Close Window
Command Line

Bindings

Properties

Dimensions	
Position	(54, 87)
X	54
Y	87
Size	897.58 x 265.12
Width	897.58
Height	265.12
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/My_Visibility-Component.cptx
Report Status	✓ True
Variables	
My-Address	0010:20:MM

a) Configure RollCall Address on Component Instance 1



My_Bind-to-Cmpt-Beh_Eg.schx

Schematic

Component Visibility "Bind to Component Behaviour" example

Component Instance 1

Component:

Component Address = Caption not set

Component Visibility = false

Component Instance 2

Component:

Component Address = Caption not set

Component Visibility = false

Properties

Dimensions	
Position	(55, 366)
X	55
Y	366
Size	897.58 x 265.12
Width	897.58
Height	265.12
Aspect Ratio	Custom
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Path	/components/My_Visibility-Component.cptx
Report Status	✓ True
Variables	
My-Address	0010:20:30

b) Configure RollCall Address on Component Instance 2

Fig. 4-50: Configuring RollCall Address on Component Instances:
a) Instance 10010:20:MM (an invalid address)
b) Instance 20010:20:30

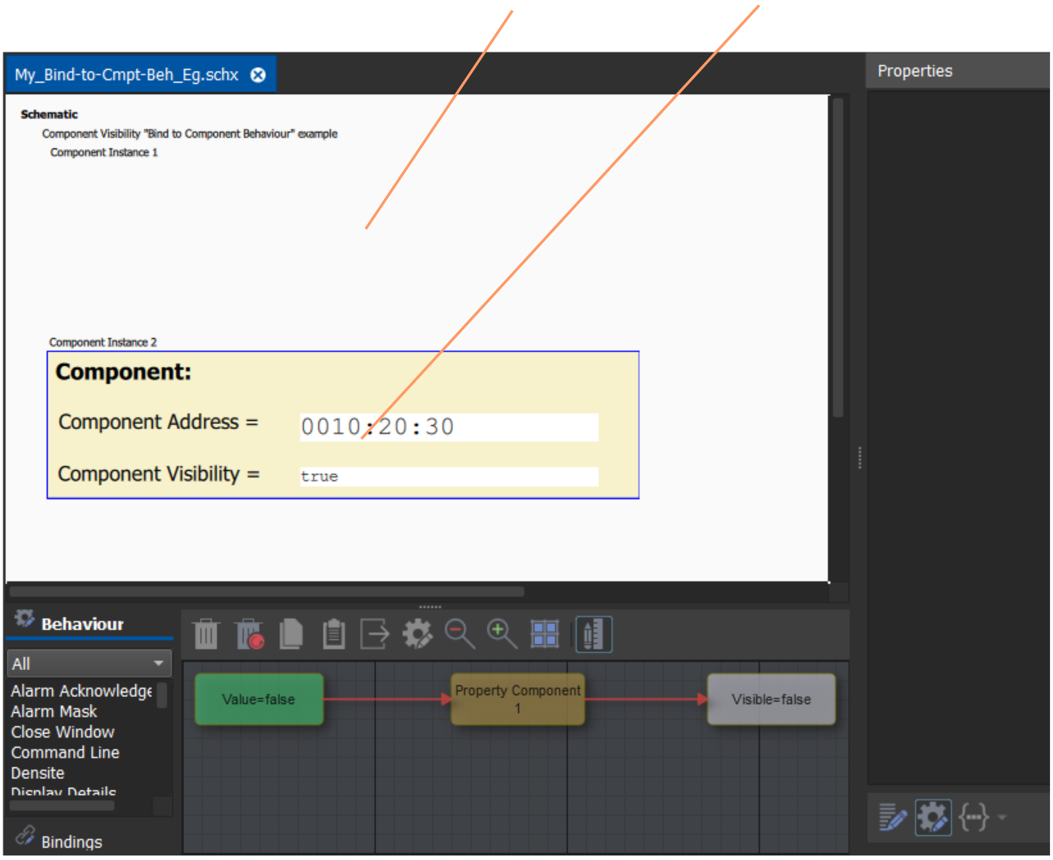
Exercise the Screen

To exercise this screen:

1 Click on the **Test** button.
  The screen is exercised.

2 The settings of Figure 4-50(a and b) result in the first instance of the component not being shown because its RollCall address is invalid. See Figure 4-51.

Instance 1 *not* visible. Instance 2 visible.



Schematic
Component Visibility "Bind to Component Behaviour" example
Component Instance 1
Component Instance 2

Component:

Component Address = 0010:20:30

Component Visibility = true

Fig. 4-51: Exercising Screen.

5 Bindings

Summary

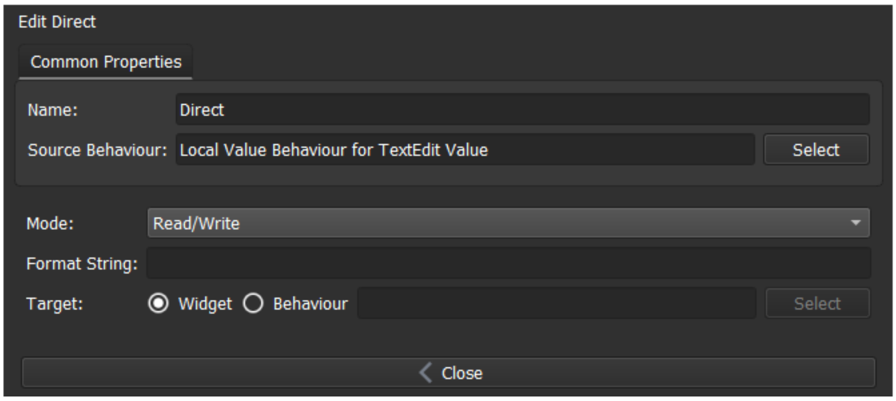
Bindings

This chapter introduces some MapView-specific Bindings which are used in the examples in this document. Other Bindings are described in the ‘Orbit for Multiviewers’ user manual.

Direct Binding

The **Direct** Binding is used to link two values. The **Direct** Binding performs a one-to-one mapping between a source Behaviour's value and a target value of a widget or another Behaviour. It is generally used with widgets that have a 'value' (for example, user interface widgets such as **Text Box**, or **Slider**). Any source value changes may be reflected at the target, and vice versa. (This includes some value attributes, for example, 'visibility', read-only and error states - some of which may be hidden from the user.)

Figure 5-1 shows the Binding's properties.



Edit Direct

Common Properties

Name: Direct

Source Behaviour: Local Value Behaviour for TextEdit Value

Mode: Read/Write

Format String:

Target: ● Widget ○ Behaviour

Fig. 5-1: Direct Binding Edit Properties Dialog

Table 5-1: Direct Binding Properties

Property	Description
Name	Enter a name to identify the Binding.
Source Behaviour	The Behaviour that the **Direct** Binding is to be connected to. For example, a connection to a **Local Value** Behaviour provides a convenient way of getting/setting the value of a user interface/control widget. Widgets may be directly bound to Behaviours such as RollCall-type Behaviours.
Mode	Determines the 'direction' of the Binding connection to be made - in one direction, or two-way between the source Behaviour value and target widget/Behaviour value. The Binding's connection will work:
Read	•from source Behaviour value to target value only;
Write	•from target value to source Behaviour value only;
Read/Write	•in both directions.
Format String	Define a text format string. This can be used to prefix text, and/or format text. Format string examples: •%.2f – Format a numeric value to 2 decimal places. •%s dB – Append text "dB" to the end of a string. •%X – Display a numeric value as hexadecimal string. **Note:** These are standard C-style format strings.

Table 5-1: Direct Binding Properties (continued)

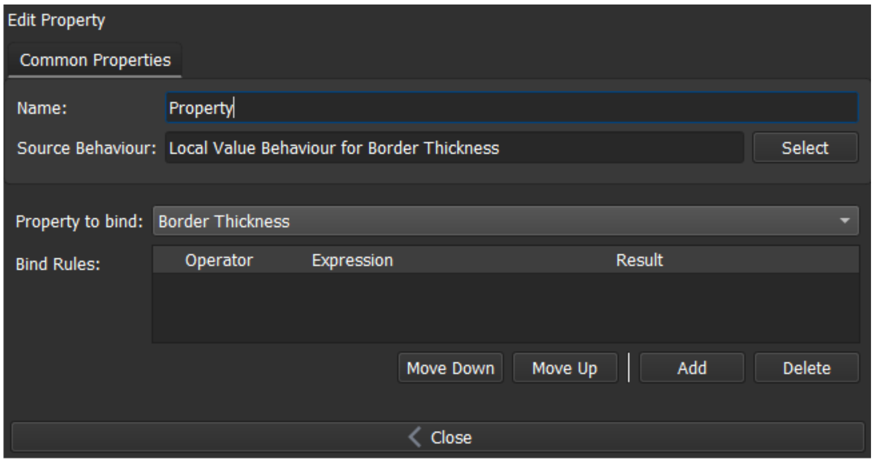
Property	Description
Target	Define the target value for the Binding's source value to be connected.
Radio buttons:	
Widget	Select to use the widget value.
Behaviour	Select to use a Behaviour value.
Select	Button/text box. Active when Behaviour is selected. Select the Behaviour to use as target.

Property Binding

The **Property** Binding is used to update a widget's property value from a source Behaviour value. For example, an on-screen tally lamp color may depend on some alarm value.

The **Property** Binding can be configured with rules to map the source Behaviour value to values or a value range suitable for the target property. (If no rules are present, then the source value is used directly.)

Figure 5-2 shows the Binding's properties which are described in Table 5-2.



Edit Property

Common Properties

Name: Property

Source Behaviour: Local Value Behaviour for Border Thickness

Property to bind: Border Thickness

Bind Rules:

Operator	Expression	Result

Fig. 5-2: Property Binding Edit Properties Dialog

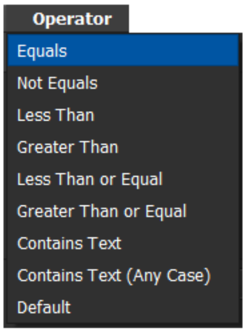
Table 5-2: Property Binding Properties

Property	Description
Name	Enter a name to identify the Binding.
Source Behaviour	Select the source Behaviour that the Binding is to be connected to.
Property to Bind	Select the widget property to bind to. Note: The drop-down list items shown depends on the widget being bound to.
Bind Rules	Processing rules are listed in this Bind Rules section of the table. Use a default rule, to be used if no other rule matches. **Note**: The *first* rule to be satisfied is used. Therefore, the ordering of the rules in the list is critical. **Note**: If the rules table is empty, the value will be copied directly to the target value.

Table columns:

Operator	Drop-down box. Select the operator for the rule to use.

Table 5-2: Property Binding Properties (continued)

Property	Description
	Select operator.  Operator Equals Not Equals Less Than Greater Than Less Than or Equal Greater Than or Equal Contains Text Contains Text (Any Case) Default
Expression	Enter literal, fixed value. This value is compared with the Behaviour value. For example, '49' or 'Warn'.
Result	Enter value to use if the rule evaluates to true. This can be a literal value, a color or a string. For example, '49', or '#00FF00', or 'Warning'. **Note:** In a string value type, "%1" can be used to represent the **Behaviour** value used. (If the Behaviour has more than one value, then %2, %3 etc can also be used.) **Note:** A format specifier can be used (%1,%2.f,%d etc.) For example, to format a number to be 'My Number is: 12' then "My Number is:%1" can be used in the Result field.
Buttons:	
Move Down	Click to move the selected row down the list.
Move Up	Click to move the selected row up the list.
Add	Click to add a new, blank rule row to the list.
Delete	Click to delete the selected row from the list.

Event Binding

The **Event** Binding is used to handle user interface (UI) events (for example, button click events). Upon an event, the Binding will either: execute a target Behaviour's action; or set a target Behaviour's value to a new value. It can be used for the following purposes when a UI control is clicked by a user:

- Execute a target Behaviour action when a certain condition is met.
- Set a Behaviour value to a specific value, or to a value copied from a source Behaviour.

And it can be used to:

- Execute automatically when a source Behaviour value changes, or is changed to a specific value.

Conditions can be configured in the **Event** Binding to qualify execution of the **Event** Binding. This enables an action to be executed dependent on a current value, or if a source Behaviour value changes to a specific value.

Note: Dependent on what a Behaviour does and how it is intended to work, there may not be actions. Therefore these do not execute.

An **Event** Binding may be added to a UI widget, for example to a **Button** widget.

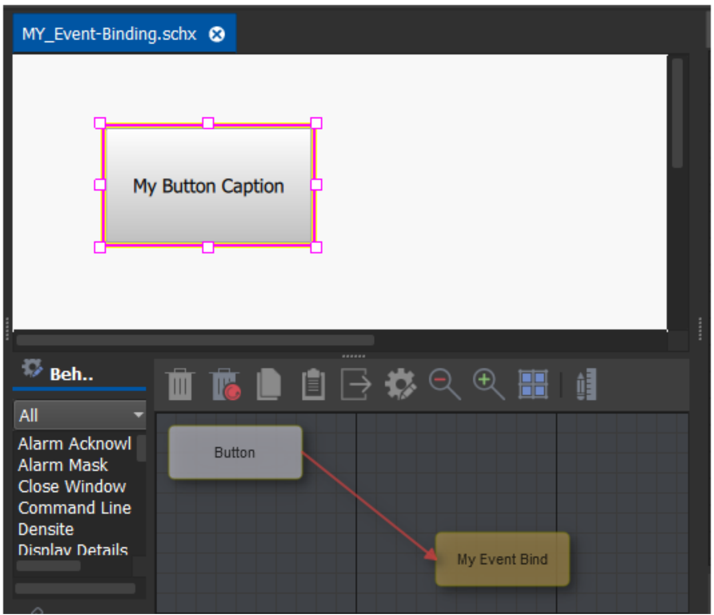
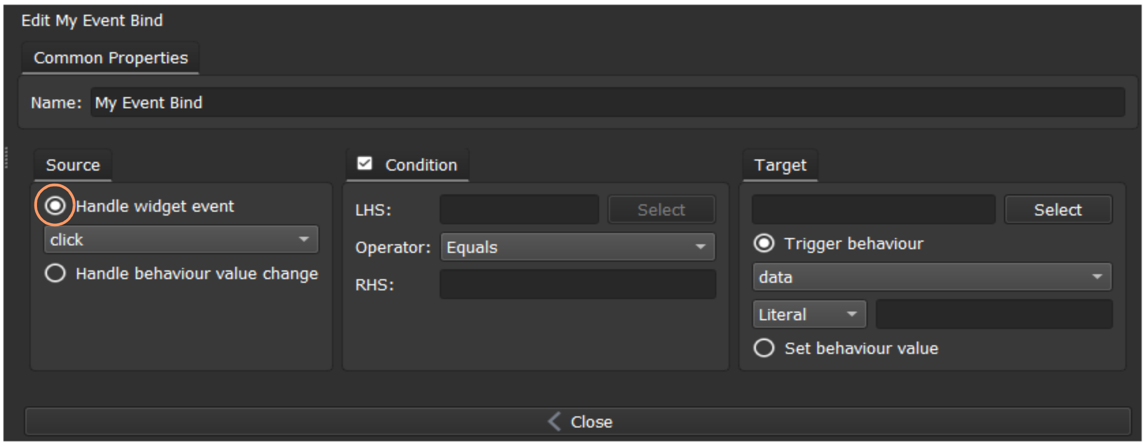


Fig. 5-3: Button Widget and Event Binding

The **Event** Binding settings screen is shown in Figure 5-4.



Edit My Event Bind

Common Properties

Name: My Event Bind

Source

● Handle widget event

click

○ Handle behaviour value change

☑ Condition

LHS:

Operator: Equals

RHS:

Target

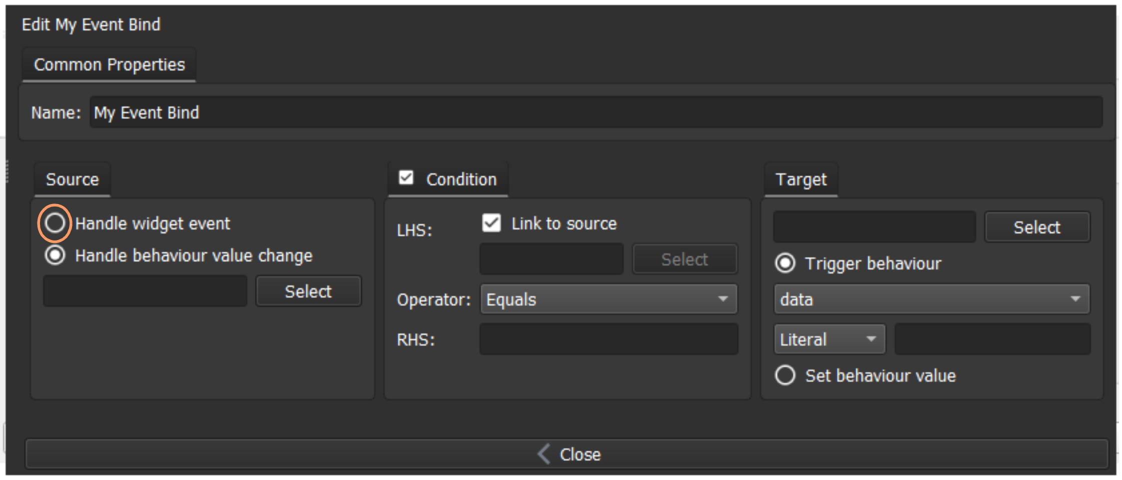
● Trigger behaviour

data

Literal

○ Set behaviour value

a) Source= ‘Handle widget event’



Edit My Event Bind

Common Properties

Name: My Event Bind

Source

○ Handle widget event

● Handle behaviour value change

☑ Condition

LHS: ☑ Link to source

Operator: Equals

RHS:

Target

● Trigger behaviour

data

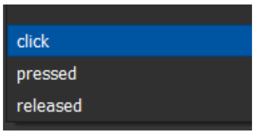
Literal

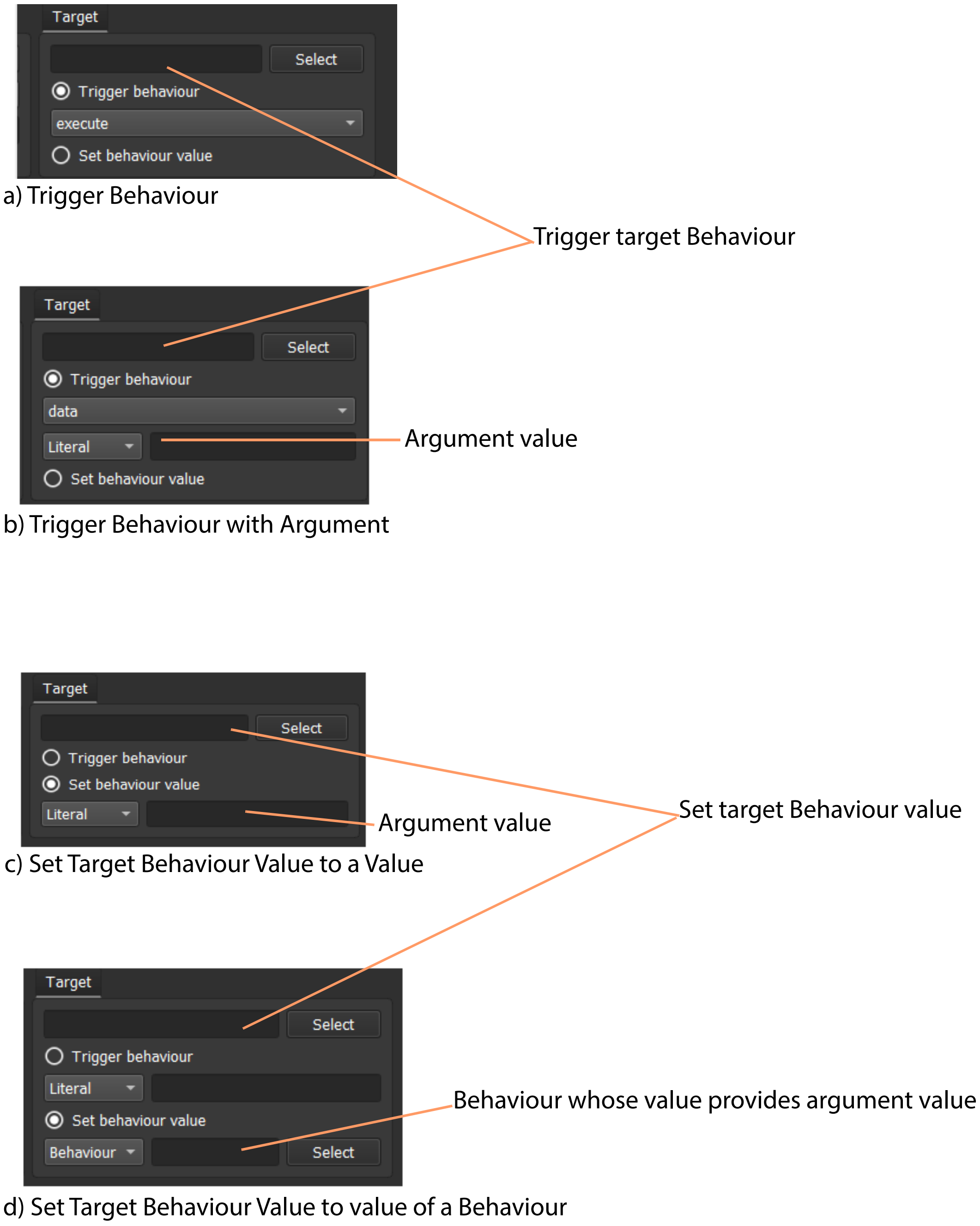
○ Set behaviour value

b) Source= ‘Handle behaviour value change’

Fig. 5-4: Edit Event Binding Settings Dialog Appearance:
a) Handle Widget event selected.
b) Handle Behaviour value change selected.

Table 5-3: Event Binding Properties

Property	Description
Name	Enter a name to identify the Binding.
Source	Radio buttons.
Handle widget event	Select to react to a user interface widget event. The specific event is selectable via a drop-down menu. Options:  • **Click** - when widget is clicked on. • **Pressed** - when (button) is pressed. • **Released** - when (button press) is released.
Handle Behaviour value change	Select to react to a Behaviour value. The specific Behaviour is selected via a **Select Behaviour** dialog.
Condition	Check box. Select to apply a condition to the Event handling.
LHS	Set up the left hand side of the condition expression.
Link to source	Check box. Select to use the source Behaviour value. Text box. (Link to source not selected) Select Behaviour to use for the condition expression.
Operator	Select operator for the condition expression. Options are similar to those used by a Logical Binding (see Logical Binding, on page 143).
RHS	Enter a value to use in right hand side of the condition expression.
Target	Select a target Behaviour via a **Select Behaviour** dialog. Radio buttons:
Trigger Behaviour	Select this to trigger a Behaviour with an argument. Options: • **Execute** - trigger the target Behaviour. (See Figure 5-5a.) • **Data** - set an argument value to trigger the target Behaviour with. The argument can be a value or a Behaviour value. (See Figure 5-5b.) See Behaviour Arguments, on page 132 for information about arguments to Behaviours.
Set Behaviour value	Select this to set the target Behaviour to a value. (See Figure 5-5c.) Options: • **Literal** - set a value to use. (See Figure 5-5c.) • **Behaviour** - set a Behaviour value to use. (See Figure 5-5d.)



Target

Select

● Trigger behaviour

execute

○ Set behaviour value

a) Trigger Behaviour

Trigger target Behaviour

Target

Select

● Trigger behaviour

data

Literal

○ Set behaviour value

Argument value

b) Trigger Behaviour with Argument

Target

Select

○ Trigger behaviour

● Set behaviour value

Literal

Argument value

Set target Behaviour value

c) Set Target Behaviour Value to a Value

Target

Select

○ Trigger behaviour

Literal

● Set behaviour value

Behaviour

Select

Behaviour whose value provides argument value

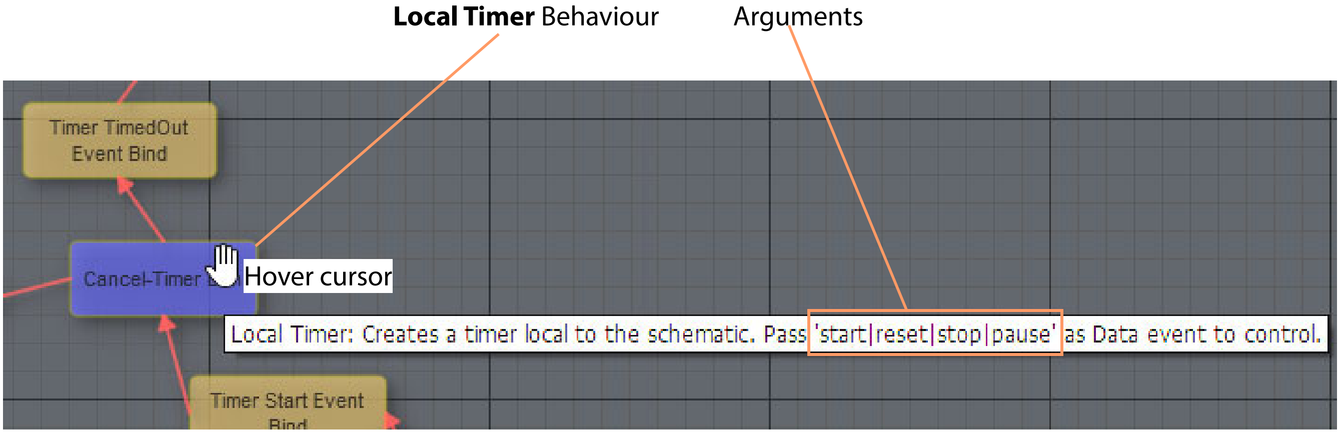
d) Set Target Behaviour Value to value of a Behaviour

Fig. 5-5: Event Binding Settings - Target

Behaviour Arguments

A target Behaviour may be passed an argument when it is executed. This will control its action. To find out if a Behaviour has arguments, in the 'Behaviour and Binding' graphical editor:

- Hover the cursor over the Behaviour.
  Information about the Behaviour is shown. For example, see Figure 5-6.



Local Timer Behaviour

Arguments

Hover cursor

Local Timer: Creates a timer local to the schematic. Pass 'start|reset|stop|pause' as Data event to control.

Fig. 5-6: Behaviour Information and Arguments.

String Op Binding

The **String Op** Binding permits string functions to be performed on Behaviour values. (For example, to extract part of a log field, or to test for a specific value.) The result may be part of the input text string, a Boolean value or an integer value. The result is stored in a target Behaviour. If input values changes, the Binding will keep the target Behaviour value updated.

The resulting Behaviour value may be bound to a widget or used by other bindings to form more complex functions.

Figure 5-7 summarizes the **String Op** Binding settings and Figure 5-8 lists available string operations.

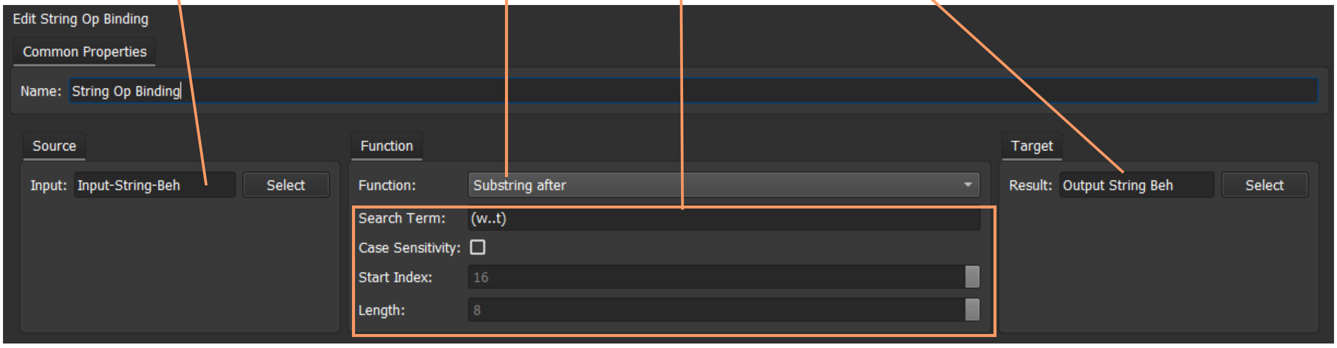
Select the Behaviour value to provide the input text string.

Select function of the string operator:

- **Search Term:** Set a search string.
  (The usage depends on the function selected.)
- **Case Sensitivity:** Check box. Select to enable case-sensivity for the selected function.
- **Start Index:** Start character position parameter for some functions.
- **Length:** Sets a length parameter for same functions.

String operator function parameters.

Select Behaviour to put result value into.



Name: String Op Binding

Input: Input-String-Beh

Function: Substring after

Search Term: (w..t)

Case Sensitivity: ☐

Start Index: 16

Length: 8

Result: Output String Beh

Fig. 5-7: String Op Binding Settings

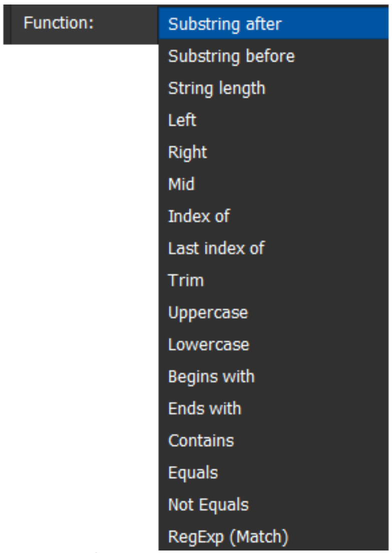


Fig. 5-8: String Functions

Table 5-4 gives some examples of each string operator function.

Table 5-4: String Operator Function Examples

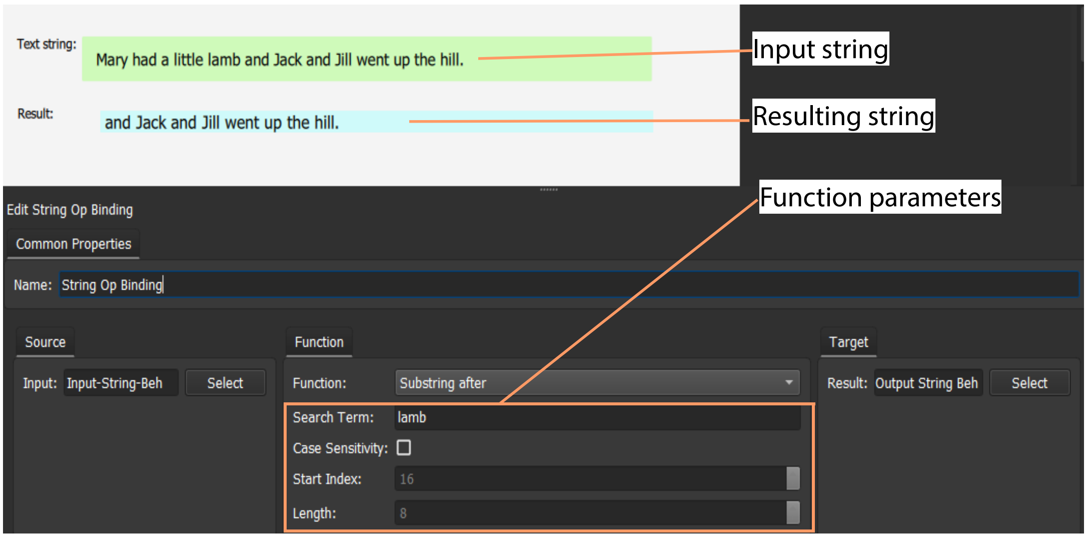
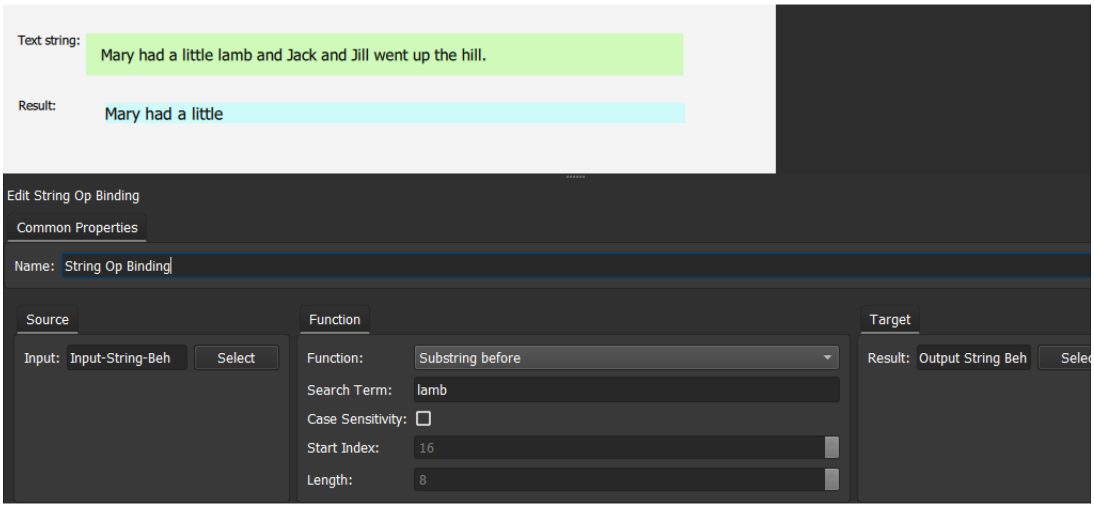
Operation	Example
Substring after	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: and Jack and Jill went up the hill. Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Substring after Search Term: lamb Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Select
Substring before	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: Mary had a little Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Substring before Search Term: lamb Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Selec

Table 5-4: String Operator Function Examples (continued)

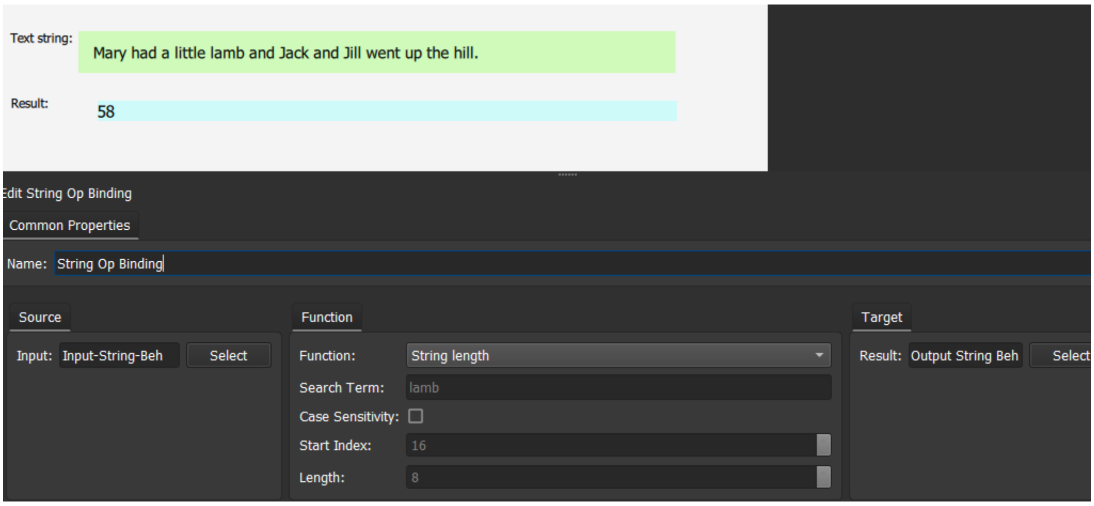
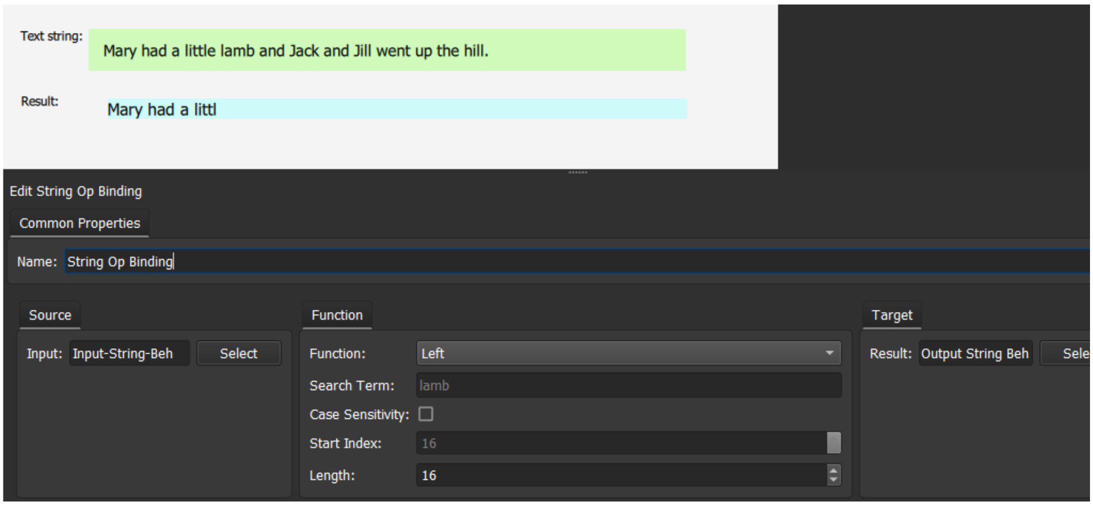
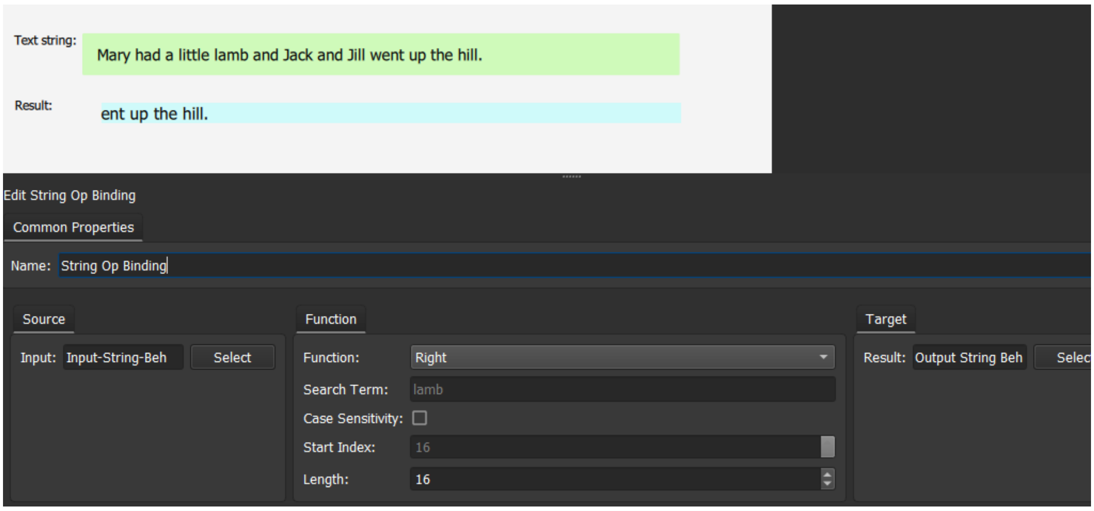
Operation	Example
String length	Result is an integer type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: 58 Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: String length Search Term: lamb Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Select
Left	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: Mary had a littl Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Left Search Term: lamb Case Sensitivity: □ Start Index: 16 Length: 16 Target Result: Output String Beh Sele
Right	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: ent up the hill. Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Right Search Term: lamb Case Sensitivity: □ Start Index: 16 Length: 16 Target Result: Output String Beh Selec

Table 5-4: String Operator Function Examples (continued)

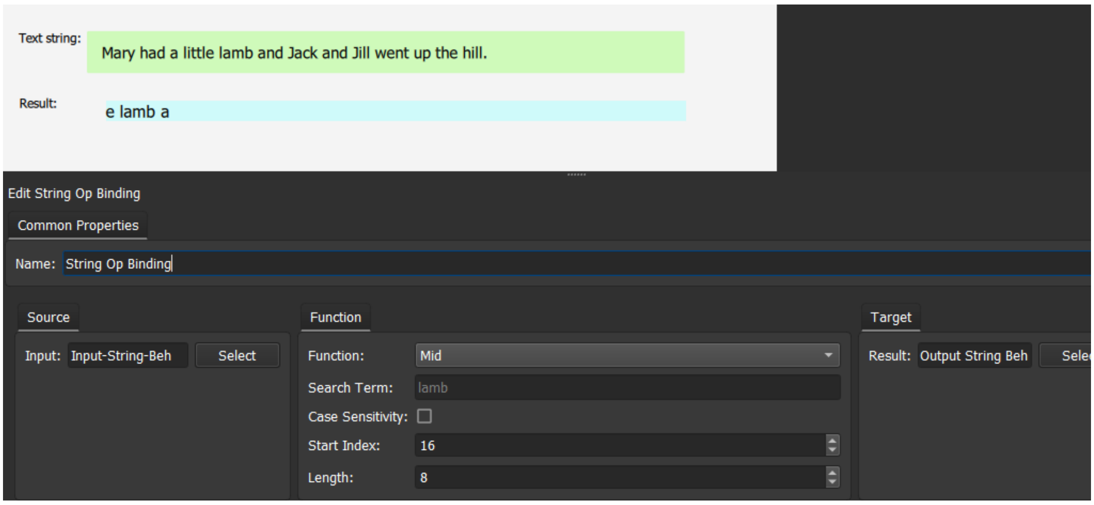
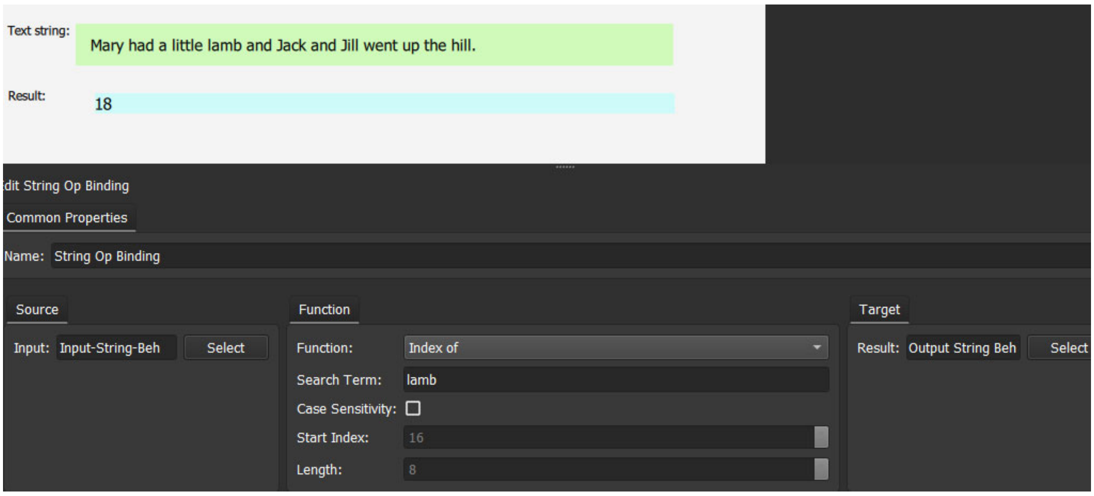
Operation	Example
Mid	Result is a string type. **Example 1:**  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: e lamb a Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Mid Search Term: lamb Case Sensitivity: ☐ Start Index: 16 Length: 8 Target Result: Output String Beh Sele **Example 2:** Parse “1000:AA:00”: • Start = 5, Length = 2; • Result = “AA”
Index of	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: 18 dit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Index of Search Term: lamb Case Sensitivity: ☐ Start Index: 16 Length: 8 Target Result: Output String Beh Select

Table 5-4: String Operator Function Examples (continued)

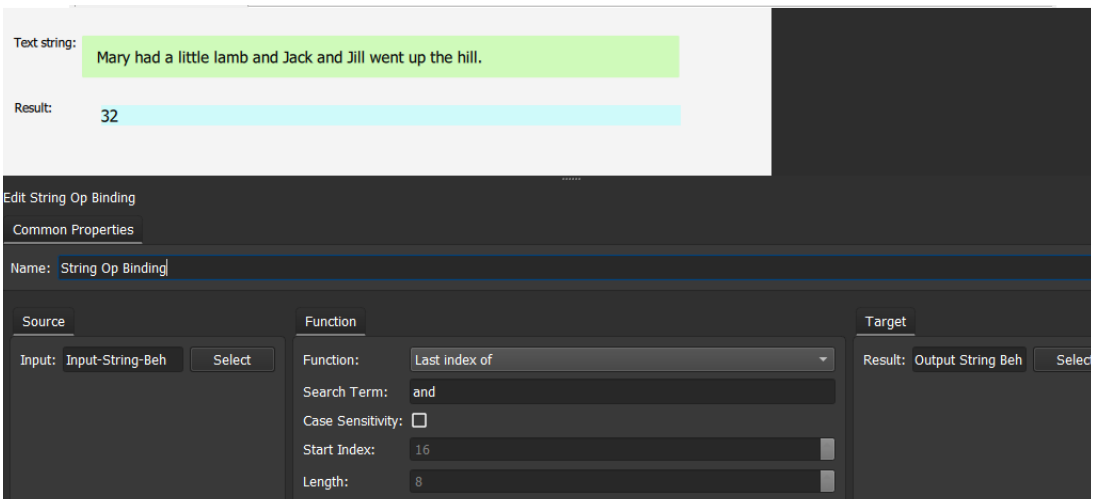
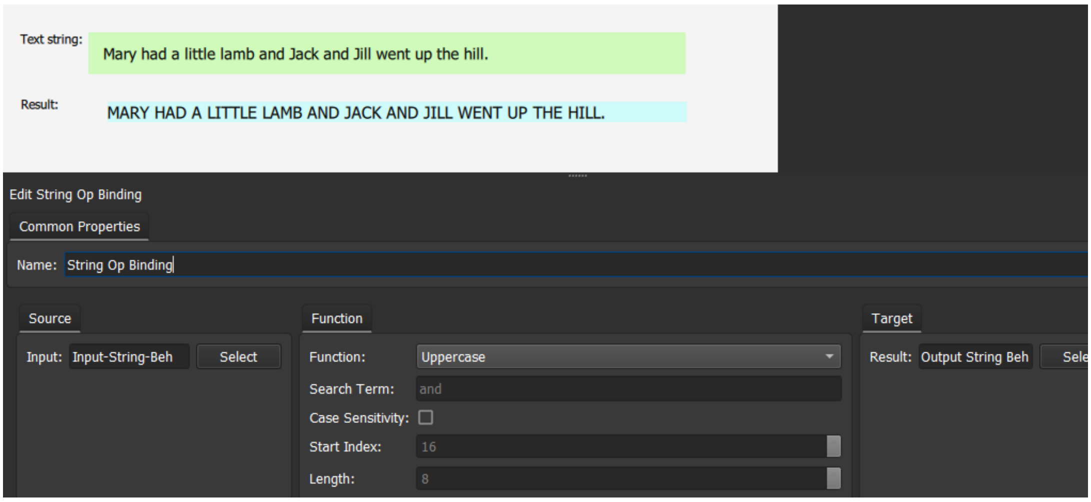
Operation	Example
Last Index of	Result is an integer type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: 32 Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Last index of Search Term: and Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Selec
Trim	Result is a string type. Removes white space from beginning and end of a string. **Example:** • Input string = " 100 " • Result = "100"
Uppercase	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: MARY HAD A LITTLE LAMB AND JACK AND JILL WENT UP THE HILL. Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Uppercase Search Term: and Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Sele

Table 5-4: String Operator Function Examples (continued)

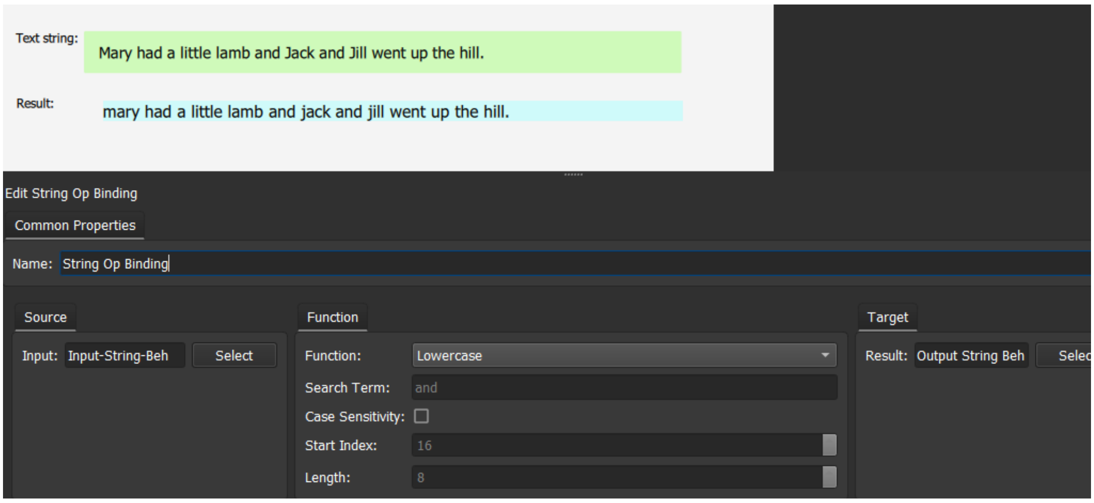
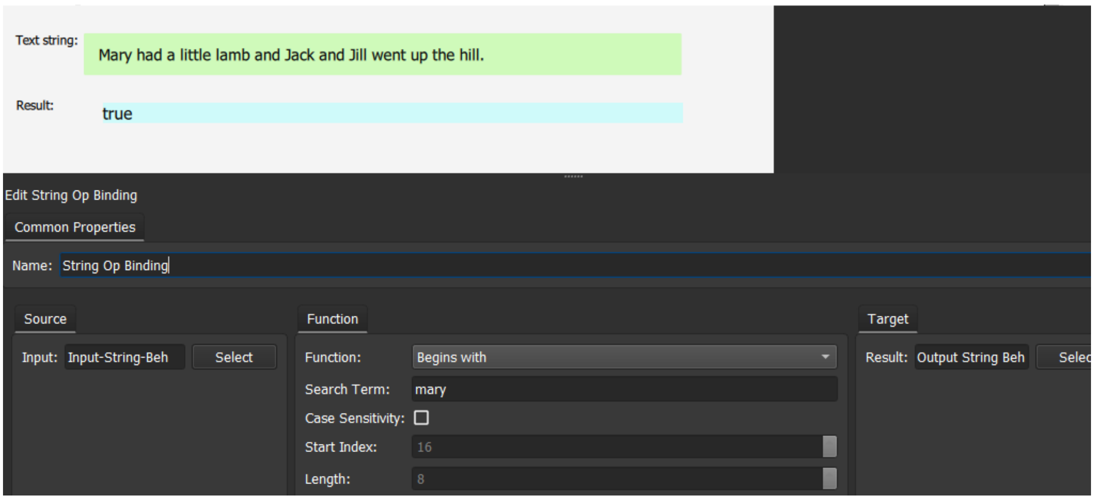
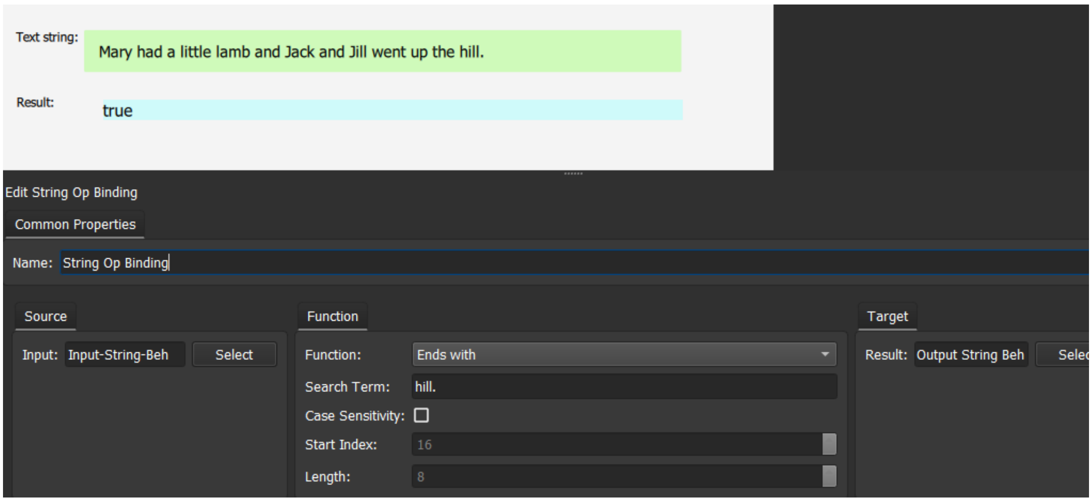
Operation	Example
Lowercase	Result is a string type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: mary had a little lamb and jack and jill went up the hill. Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Function Function: Lowercase Search Term: and Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh
Begins with	Result is a Boolean type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Function Function: Begins with Search Term: mary Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh
End with	Result is a Boolean type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Function Function: Ends with Search Term: hill. Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh

Table 5-4: String Operator Function Examples (continued)

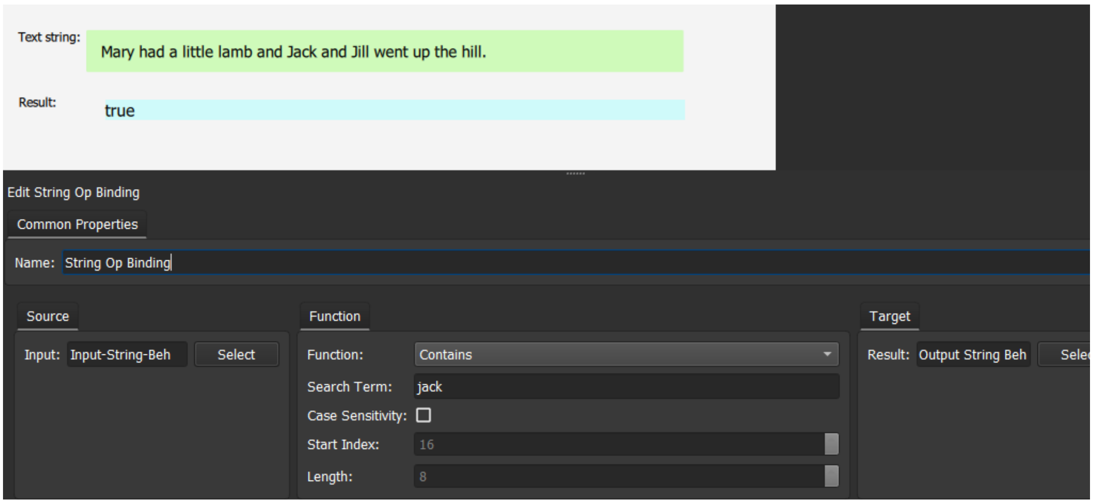
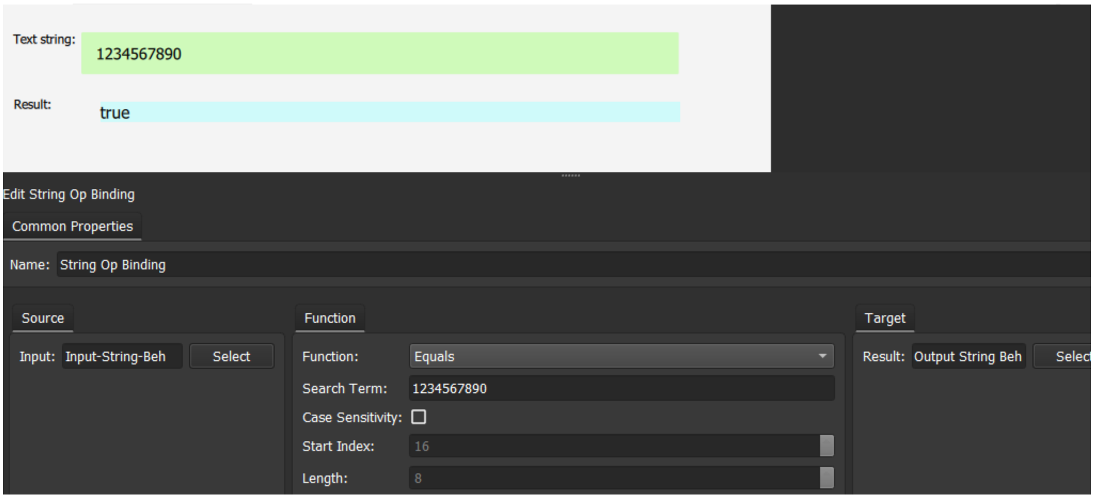
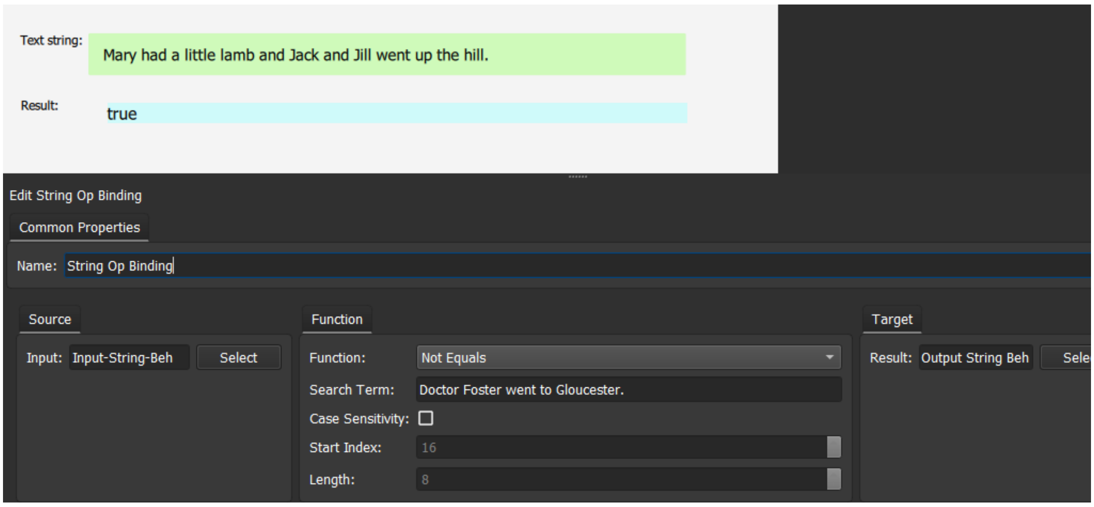
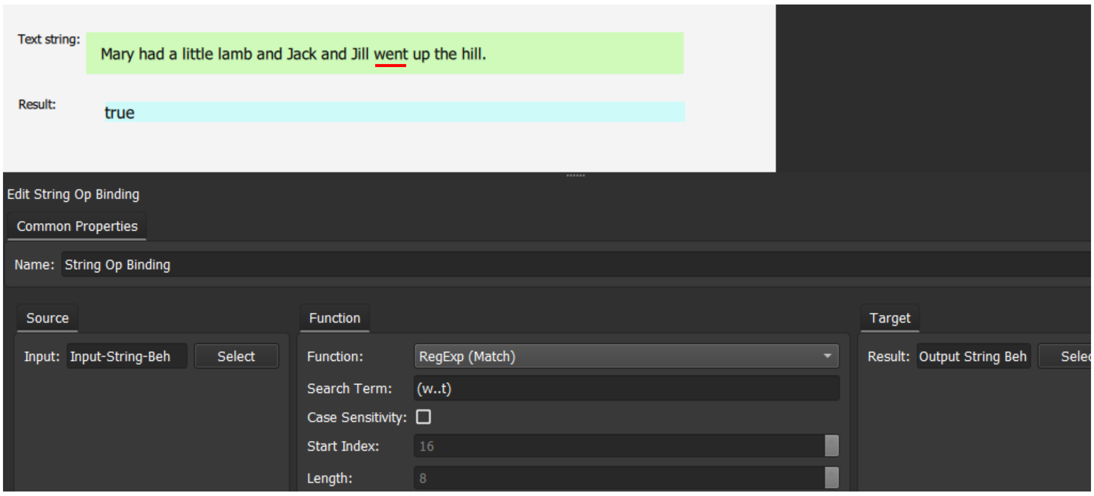
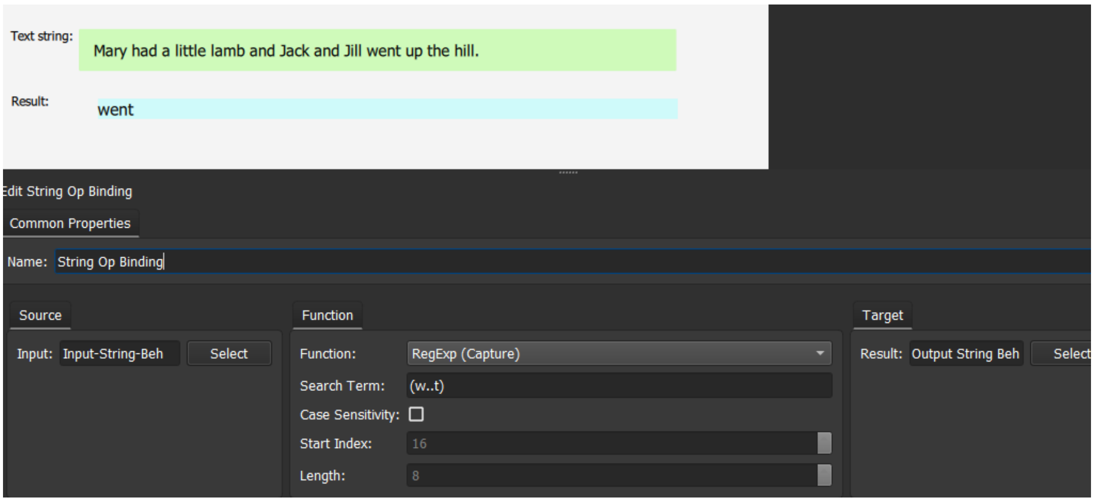
Operation	Example
Contains	Result is a Boolean type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Contains Search Term: jack Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Sele
Equals	Result is a Boolean type.  Text string: 1234567890 Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Equals Search Term: 1234567890 Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Selec
Not Equals	Result is a Boolean type.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: Not Equals Search Term: Doctor Foster went to Gloucester. Case Sensitivity: □ Start Index: 16 Length: 8 Target Result: Output String Beh Sele

Table 5-4: String Operator Function Examples (continued)

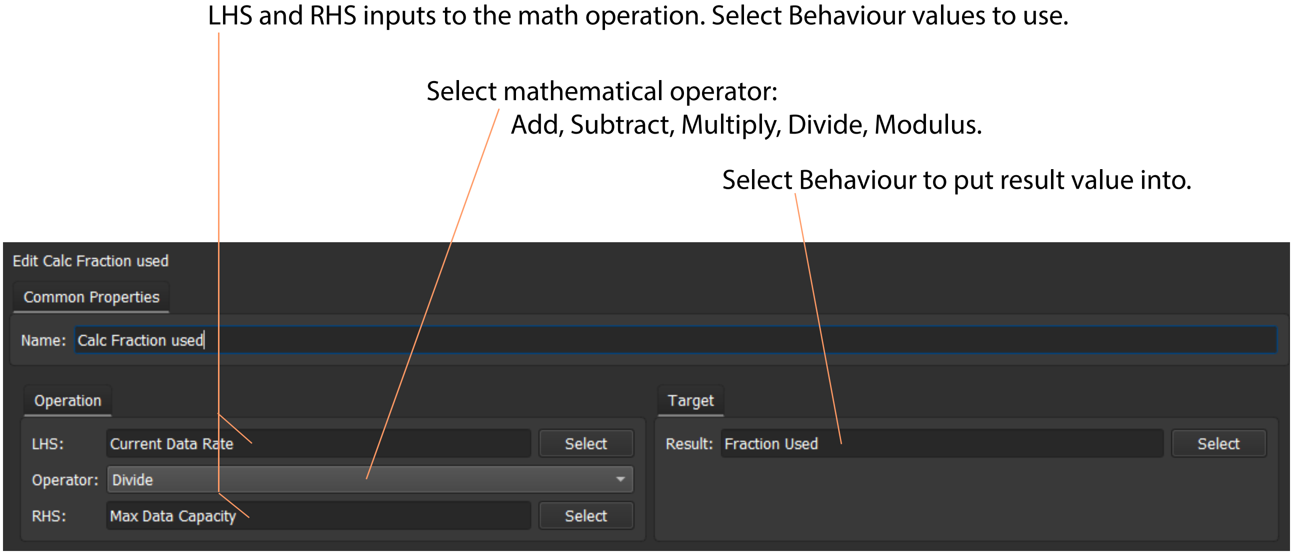
Operation	Example
RegExp (Match)	Result is a Boolean value. The function uses regular expressions to match text in input string. **Example 1:** Match 'w' and 't' which are separated by any two characters. E.g. 'went'.  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: true Edit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: RegExp (Match) Search Term: (w..t) Case Sensitivity: ☐ Start Index: 16 Length: 8 Target Result: Output String Beh Sele **Example 2:** Does input string contain a numeric value? • Input string = "FAIL: 10". • Search term = \d+ • Result = True
RegExp (Capture)	Result is a string value. The function uses regular expressions to capture text from the input string. **Example 1:**  Text string: Mary had a little lamb and Jack and Jill went up the hill. Result: went dit String Op Binding Common Properties Name: String Op Binding Source Input: Input-String-Beh Select Function Function: RegExp (Capture) Search Term: (w..t) Case Sensitivity: ☐ Start Index: 16 Length: 8 Target Result: Output String Beh Select **Example 2:** Get numeric value if input string contain a numeric value: • Input string = "FAIL: 10". • Search term = (\d+) • Result = "10"

Math Binding

The **Math** Binding is a general purpose binding which accepts two input Orbit Behaviour values (LHS and RHS) and performs a mathematical operation on them. The resulting value is stored in a target Behaviour.

If either of the input values change, the Binding will keep the target Behaviour value updated.

Many **Math** Bindings can be chained together with intermediate **Local Value** Behaviours to form a chain of operations. Figure 5-9 summarizes the **Math** Binding settings.



LHS and RHS inputs to the math operation. Select Behaviour values to use.

Select mathematical operator:
Add, Subtract, Multiply, Divide, Modulus.

Select Behaviour to put result value into.

Edit Calc Fraction used

Common Properties

Name: Calc Fraction used

Operation

LHS: Current Data Rate

Operator: Divide

RHS: Max Data Capacity

Target

Result: Fraction Used

Fig. 5-9: Math Binding Settings

Note: The **Math** Binding automatically extracts numeric values from a string that starts with a number. This is useful.

Hence, it is possible to use some strings directly (for example, from a Log Field). A string may contain '10.4 Gbits/sec' and the **Math** Binding will extract the 10.4 value.

Mapped Binding

The **Mapped** Binding is a general-purpose binding which accepts a single input Behaviour value and performs a mapping operation. The resulting value is stored in a target Behaviour. This Binding type is useful when available values are not in some particular format/range for subsequent use (for example, when converting a list of status codes 1, 2, 3 etc. into a list of colors "red", "green", "yellow" etc.).

The mapping operation is defined as an ordered list of rules. The first rule to match will be used. There may be a default rule if none of the rules match. If no rules match, and no default exists the mapping will do nothing (no-op), and the target Behaviour value will not be changed.

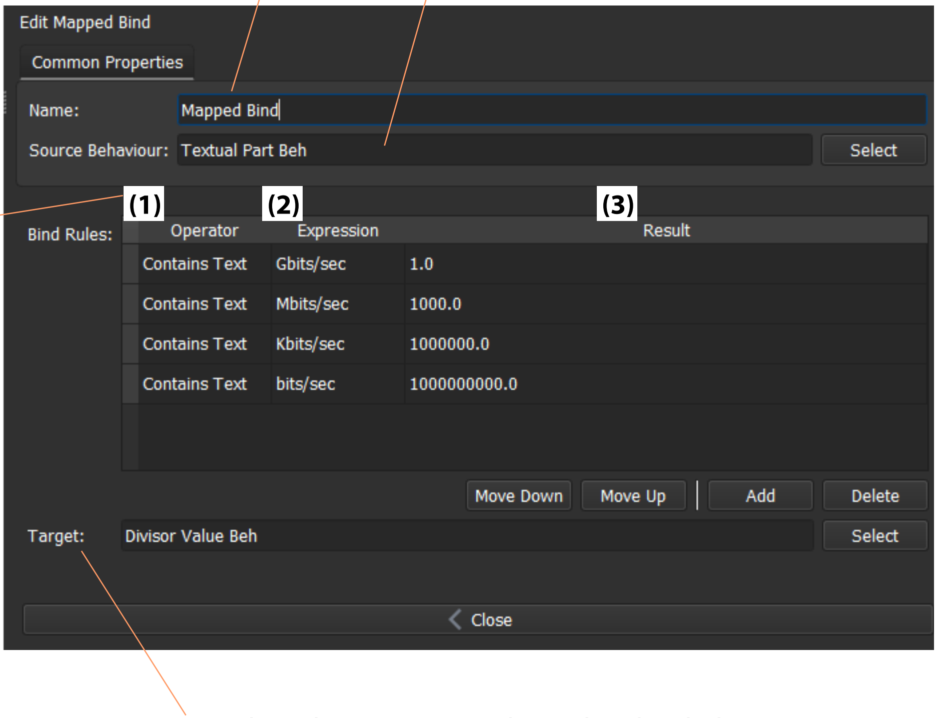
Figure 5-10 summarizes the **Mapped** Binding settings.

Name: Identifies Binding

Select Behaviour value to use.

Mapped operation is defined with Bind Rules:

- **(1) Operator** – Rule used when determining whether there is a match.
  Operators: Equals, Not Equals, Less than, Greater than, Less than equal, Greater than equal, Contains text, Contains text (any case).
- **(2) Expression** – Comparison value used to compare input against.
- **(3) Result –** Resulting value from the bind rules.



Edit Mapped Bind

Common Properties

Name: Mapped Bind

Source Behaviour: Textual Part Beh

Bind Rules:

Operator	Expression	Result
Contains Text	Gbits/sec	1.0
Contains Text	Mbits/sec	1000.0
Contains Text	Kbits/sec	1000000.0
Contains Text	bits/sec	1000000000.0

Target: Divisor Value Beh

Target - The Behaviour value to be updated with the result.

Note: A 'No-Op' result does not update the target.

Fig. 5-10: Mapped Binding Settings

Note: A **Property** Binding can perform a mapping when binding to a widget. A **Property** Binding with no rules will pass the source value.

Note: When mapping, it is good practice to explicitly use a separate **Mapped** binding, storing the resultant value in a **Local Value** Behaviour.

Logical Binding

The **Logical** Binding is a general-purpose binding designed to compare Behaviour values against either a literal value, or another Behaviour value. The resulting value is stored in a target Behaviour.

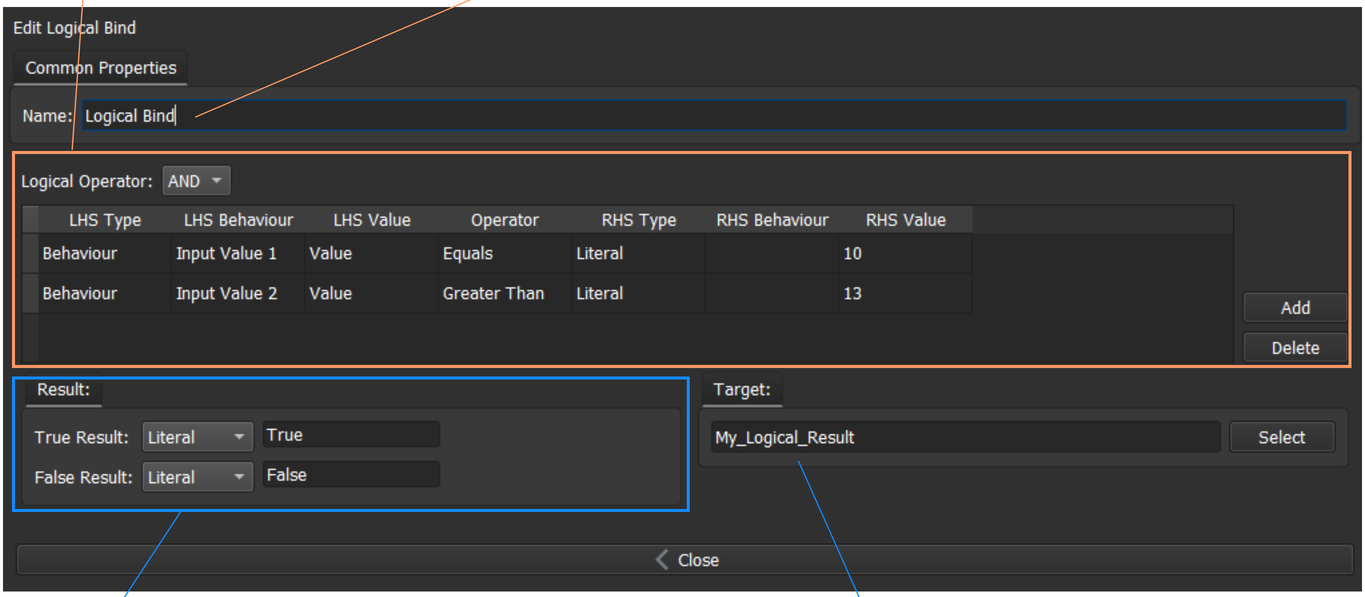
Note: Use the **Logical** Binding to compare two Behaviour values.
Use a **String Op** Binding to compare a single string with a literal.

The logical expression used in a **Logical** Binding allows many comparisons to be compounded with an 'AND' or 'OR' operator in an **Expressions Table**, which defines all expressions that are evaluated by the Binding. The **Expressions Table** will always evaluate to a Boolean True or False value. The overall evaluated value is used to generate a resulting Binding value.

Figure 5-11 summarizes the **Logical** Binding settings and Figure 5-12.

Expressions Table, see Figure 5-12.

Name: Identifies Binding



Edit Logical Bind

Common Properties

Name: Logical Bind

Logical Operator: AND

LHS Type	LHS Behaviour	LHS Value	Operator	RHS Type	RHS Behaviour	RHS Value
Behaviour	Input Value 1	Value	Equals	Literal		10
Behaviour	Input Value 2	Value	Greater Than	Literal		13

Result:

True Result: Literal True

False Result: Literal False

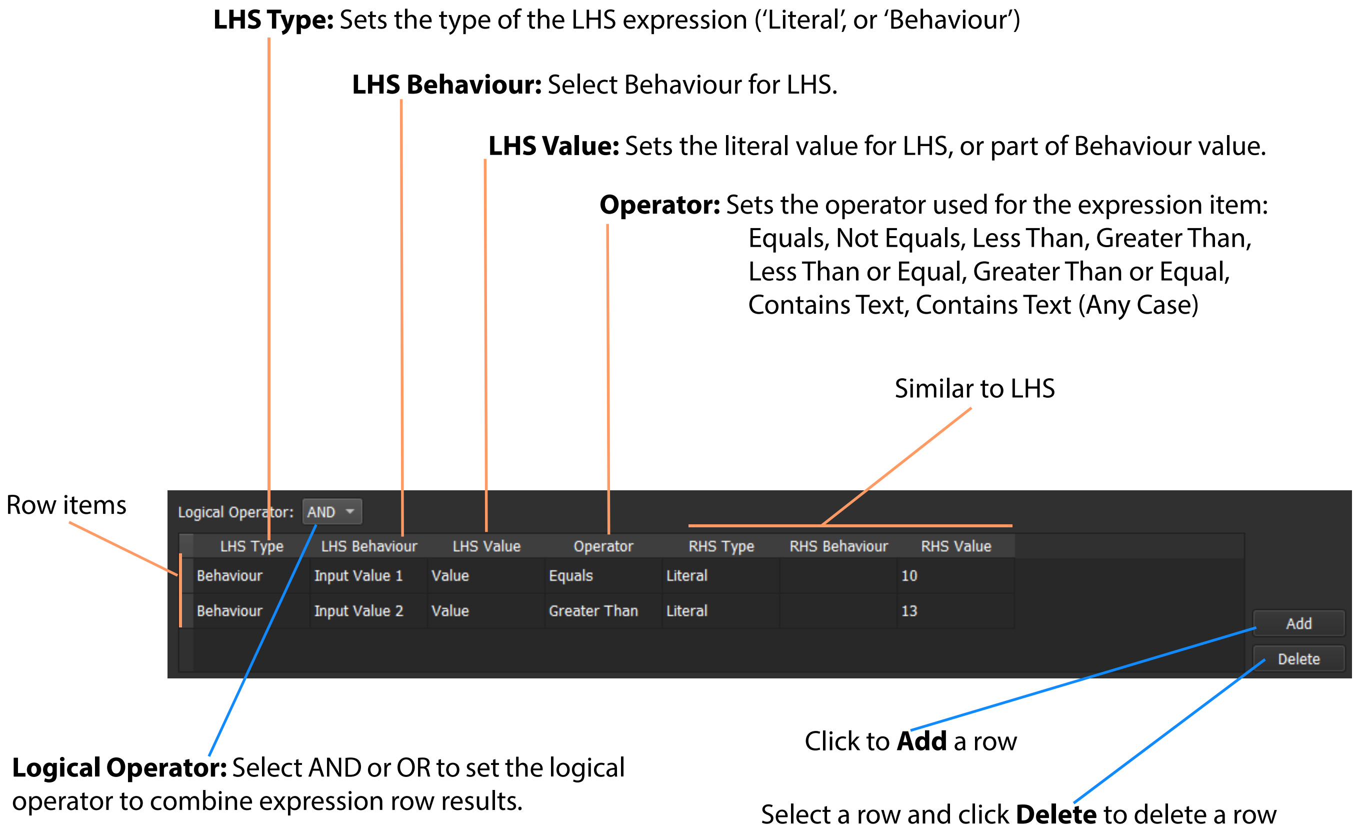
Target:

My_Logical_Result

Result: From evaluated expressions table (a True or False value), select values for the final Binding result.

Target: Select Behaviour value to update with final Binding result value.

Fig. 5-11: Logical Binding Settings



LHS Type: Sets the type of the LHS expression ('Literal', or 'Behaviour')

LHS Behaviour: Select Behaviour for LHS.

LHS Value: Sets the literal value for LHS, or part of Behaviour value.

Operator: Sets the operator used for the expression item:
Equals, Not Equals, Less Than, Greater Than,
Less Than or Equal, Greater Than or Equal,
Contains Text, Contains Text (Any Case)

Similar to LHS

Row items

Logical Operator: AND

LHS Type	LHS Behaviour	LHS Value	Operator	RHS Type	RHS Behaviour	RHS Value
Behaviour	Input Value 1	Value	Equals	Literal		10
Behaviour	Input Value 2	Value	Greater Than	Literal		13

Logical Operator: Select AND or OR to set the logical operator to combine expression row results.

Click to **Add** a row

Select a row and click **Delete** to delete a row

Fig. 5-12: Expressions Table

Table 5-5 lists some example expressions that are possible.

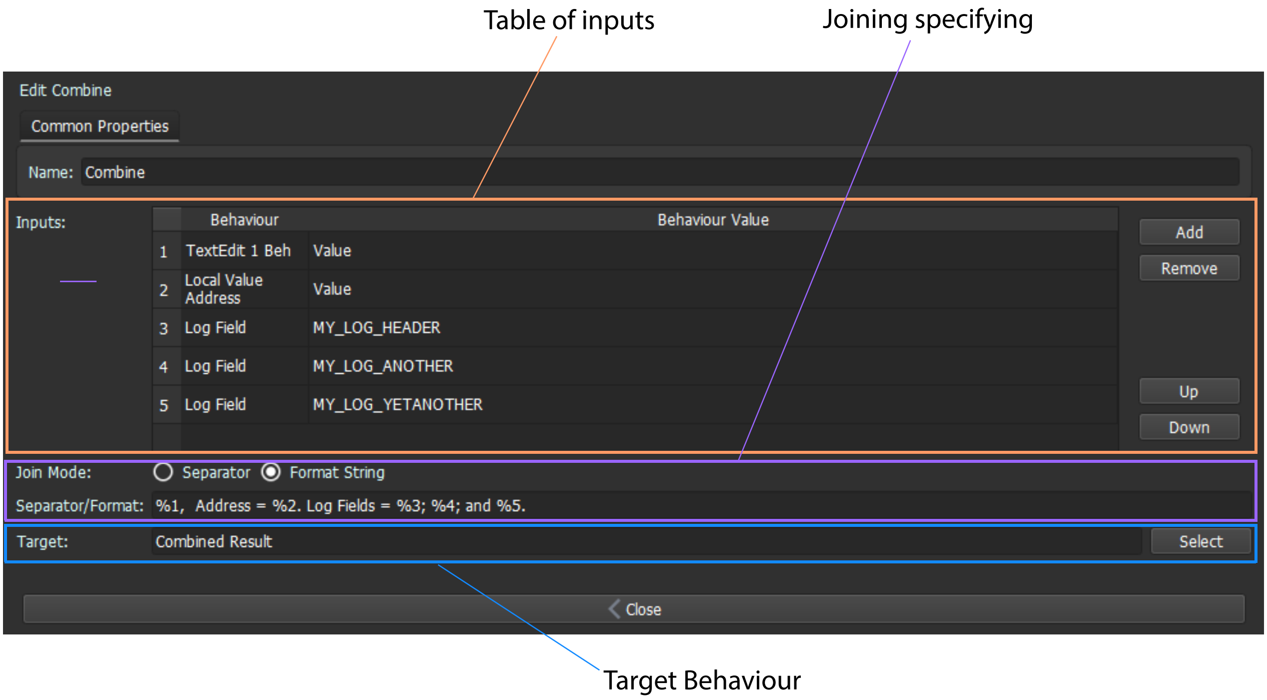
Table 5-5: Example Expressions

Example Expression	Expression Table Row Item		Logical Row Operator	Result	
	LHS	RHS		True	False
If (A=L2) **T=L1**	Behavior A.	Literal value L2.	-	L2	No-op
If (A=L1 AND B=L2) **T=B** **else T=A**	Two rows in the expressions table: 1) LHS set to Behaviour A, RHS set to L1. 2) LHS set to Behaviour B, RHS set to L2.		AND	B	A
If (A=B) **T=L1** **else T=L2**	Behavior A.	Behavior B.	-	L1	L2
If (A=B OR A>=L1) **T=L2**	Two rows in the expressions table: 1) LHS is set to Behaviour A, RHS set to B. 2) LHS is set to Behaviour A, RHS set to L1.		OR	L2	No-Op
Where:	•A and B are input Behaviour values. •T is a target Behaviour. •L1 and L2 are literal values.				

Combine Binding

The **Combine** Binding concatenates values together to form a new value. Source value changes are reflected in the target value. The Binding can concatenate strings, for example, to form a formatted command line string from two or more source values.

Figure 5-13 shows the Binding properties screen.



Name: Combine

Inputs:

	Behaviour	Behaviour Value
1	TextEdit 1 Beh	Value
2	Local Value Address	Value
3	Log Field	MY_LOG_HEADER
4	Log Field	MY_LOG_ANOTHER
5	Log Field	MY_LOG_YETANOTHER

Join Mode: ○ Separator ● Format String

Separator/Format: %1, Address = %2. Log Fields = %3; %4; and %5.

Target: Combined Result

Fig. 5-13: Combine Binding Properties

Table 5-6 describes each property item.

Table 5-6: Combine Binding Properties

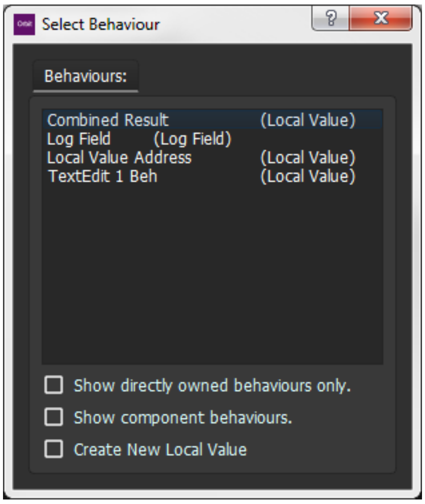
Property	Description
Name	Enter a name to identify the Binding.
Inputs	
Buttons:	
Add	Click to add a Behaviour to the bottom of the list.  Select Behaviour Behaviours: Combined Result (Local Value) Log Field (Log Field) Local Value Address (Local Value) TextEdit 1 Beh (Local Value) □ Show directly owned behaviours only. □ Show component behaviours. □ Create New Local Value

Table 5-6: Combine Binding Properties (continued)

Property	Description
Remove	Click to remove selected item from list.
Up	Click to move selected item up list.
Down	Click to move selected item down list.
Columns:	
Behaviour	Lists an input Behaviour.
Behaviour Value	Select the Behaviour value to use from drop-down list (in cases where there is more than one value associated with the added Behaviour).
Join Mode	Radio buttons. Select the mode to combine the input values:
Separator	• Select to combine separated by a fixed text string.
Format String	Select to format the combined output. A string is formed from the listed Behaviour values with a format specified by the **Separator/Format format** string.
Separator/Format	Text box. Specify the separator or the format string to use. Example: For three inputs: “Quick”, ”Brown”, and “Fox”. • Join Mode: **Separator** _SEP_ Output is: Quick_SEP_Brown_SEP_Fox •Join Mode: **Format String** `First= % 1;second= % 2;and third= % 3.` Output is: First= Quick; second= Brown; and third= Fox. The **Format String** should contain placeholders%1,%2,..., corresponding to the values of items in the list, which are inserted at the placeholder positions.
Target	Select a target Behaviour via a **Select Behaviour** dialog.

6 Behaviours

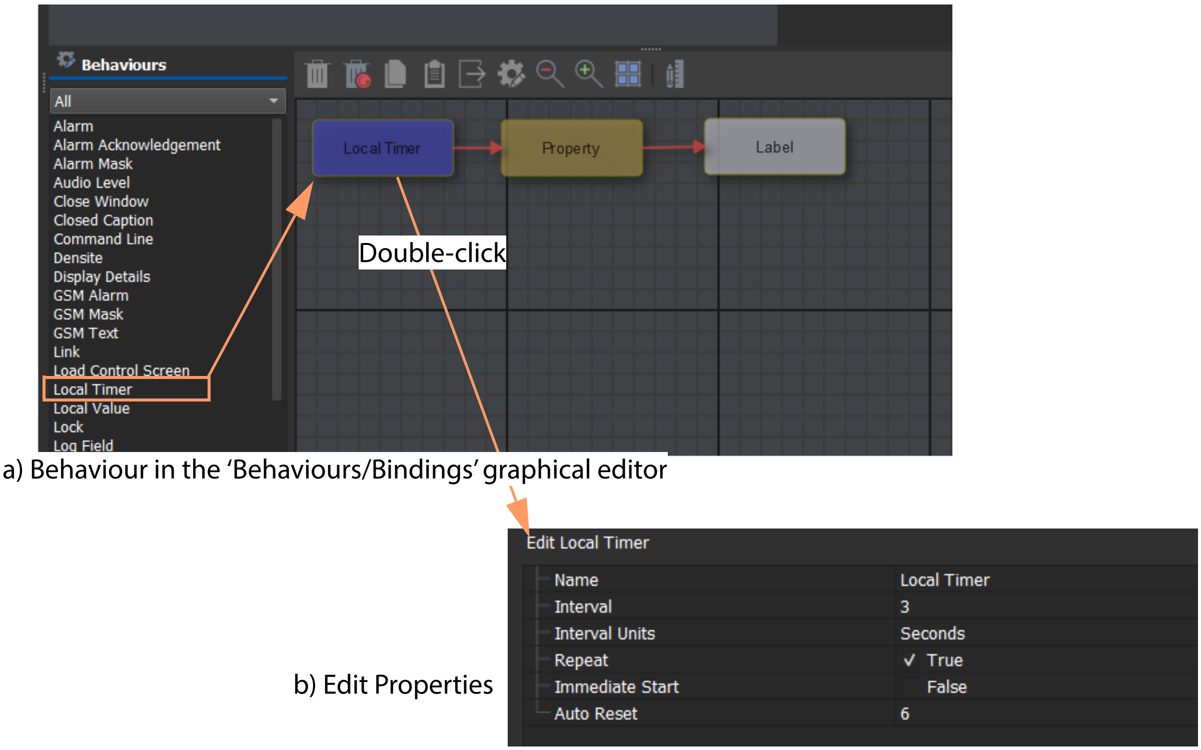
Summary

Behaviours

This chapter introduces some MapView-specific Behaviours used in examples in this document. Other Behaviours are described in the 'Orbit for Multiviewers' user manual.

Local Timer Behaviour

A **Local Timer** Behaviour implements a timer and a counter-timer on a MapView screen.



a) Behaviour in the 'Behaviours/Bindings' graphical editor

b) Edit Properties

Edit Local Timer

Name	Local Timer
Interval	3
Interval Units	Seconds
Repeat	✓ True
Immediate Start	False
Auto Reset	6

Fig. 6-1: Local Timer Behaviour:
a) in 'Behaviour and Binding' graphical Editor.
b) Edit Properties.

Table 6-1: Local Timer Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
Interval	Text box. Enter a timer duration.
Interval Units	Drop down box. Select duration time units: seconds, or milliseconds.
Repeat	Check box. • Select to enable repeated timer operation. For example, with the interval set to 1 second, a timer repeatedly times out after 1 second and the time value counts seconds. • Deselect to enable one-shot timer operation. The timer times out once, after the interval time.

Table 6-1: Local Timer Behaviour Properties (continued)

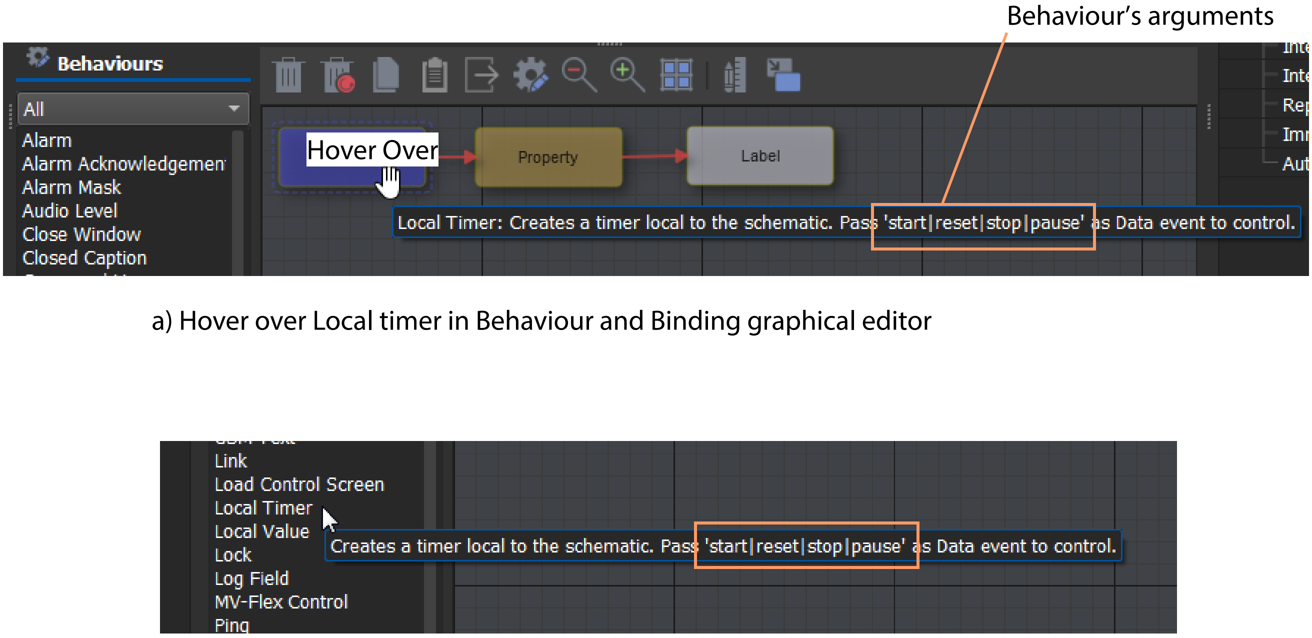
Property	**Description**
Immediate Start	Check box. Select to run the timer from the beginning.
Auto Reset	Text box. (Property is active when **Repeat** is selected.) Set timer value at which timer value resets automatically (and continues counting). For example, if **Auto Reset** = 6 and **Interval** = 1, then the timer value (count) sequence is 0,1,2,3,4,5,0,1,2....

Behaviour Arguments

When the **Local Timer** Behaviour is invoked, it can have certain arguments which determine what it does (for example, 'start' or 'stop').

A Behaviour's (any Behaviour's) arguments are shown when the cursor is either:

- hovered over the Behaviour in the 'Behaviour and Bindings' graphical editor - see Figure 6-2a for the **Local Timer** Behaviour; or
- hovered over the Behaviour name in the Behaviour list - see Figure 6-2b.



Local Timer: Creates a timer local to the schematic. Pass 'start|reset|stop|pause' as Data event to control.

a) Hover over Local timer in Behaviour and Binding graphical editor

Creates a timer local to the schematic. Pass 'start|reset|stop|pause' as Data event to control.

b) Hover over Local timer in Behaviour list

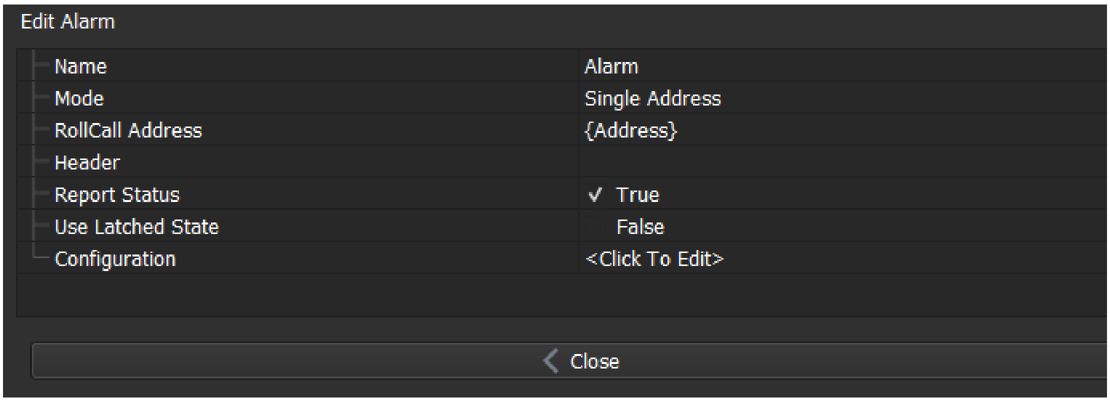
Fig. 6-2: Local Timer Behaviour's Arguments (Hover Cursor over Behaviour)

Table 6-2: Local Timer Behaviour Arguments

Argument	**Description**
start	Start the timer.
reset	Reset timer, timer value set to zero. If the timer is running, the timer value continues to count on from zero.
stop	Stop the timer. A subsequent 'start' will set the timer value to 0 and start the timer.
pause	Pause the timer and the timer value is held. A subsequent 'start' will continue the timer and the timer value sequence.

Alarm Behaviour

An **Alarm** Behaviour connects to an alarm state or to a remote log field giving a Orbit MapView screen access to the alarm state or log field value via Orbit services.



Edit Alarm

Name	Alarm
Mode	Single Address
RollCall Address	{Address}
Header	
Report Status	✓ True
Use Latched State	False
Configuration	<Click To Edit>

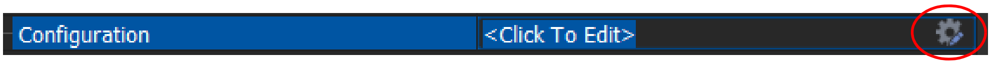
Fig. 6-3: Alarm Behaviour Properties

Table 6-3: Alarm Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
Mode	Drop-down box. Single or multiple addresses mode. • **Single Address** - one RollCall address is monitored. • **Multiple Addresses/Headers** - multiple RollCall addresses and Log Field headers can be monitored.
RollCall Address	Text box. (Grayed-out for 'Multiple Addresses/Headers' mode) Enter RollCall address. **Note:** A variable can be used, for example, {Address}.
Header	Text box. (Grayed-out for 'Multiple Addresses/Headers' mode) Enter the required Log Field header. For example, MY_LOG_HEADER.
Report Status	Check box. • **Select** - Alarm state contributes to overall screen state. **Note:** Any **Alarm** Behaviour with **Report Status** selected will contribute to the state of a **Link** Behaviour. See Link Behaviour, on page 174, and Link State, on page 175. • **De-select** - Alarm state does not contribute to overall screen state. **Note:** De-selecting **Report Status** is useful where no contribution to a **Link** Behaviour state is needed - for example, in a banner tool bar which might appear on several screens.

Table 6-3: Alarm Behaviour Properties (continued)

Property	Description
Use Latched State	Check box. Select for the Behaviour to use the latched alarm state. (A latched alarm is useful when an alarm occurs briefly.) **Note:** Orbit services provide both a • normal, 'live', non-latched Alarm state; and a • latched Alarm state. (A latched Alarm state can be reset with the **Reset Latch** Behaviour, see Reset Latch Behaviour, on page 158.)
Configuration	Text box. (Property is active when **Multiple Addresses/Headers** mode is selected.) Click the ⚙ symbol to open the **Alarm Configuration** dialog to enter RollCall addresses and Log Headers. See Alarm Configuration Dialog, on page 152.

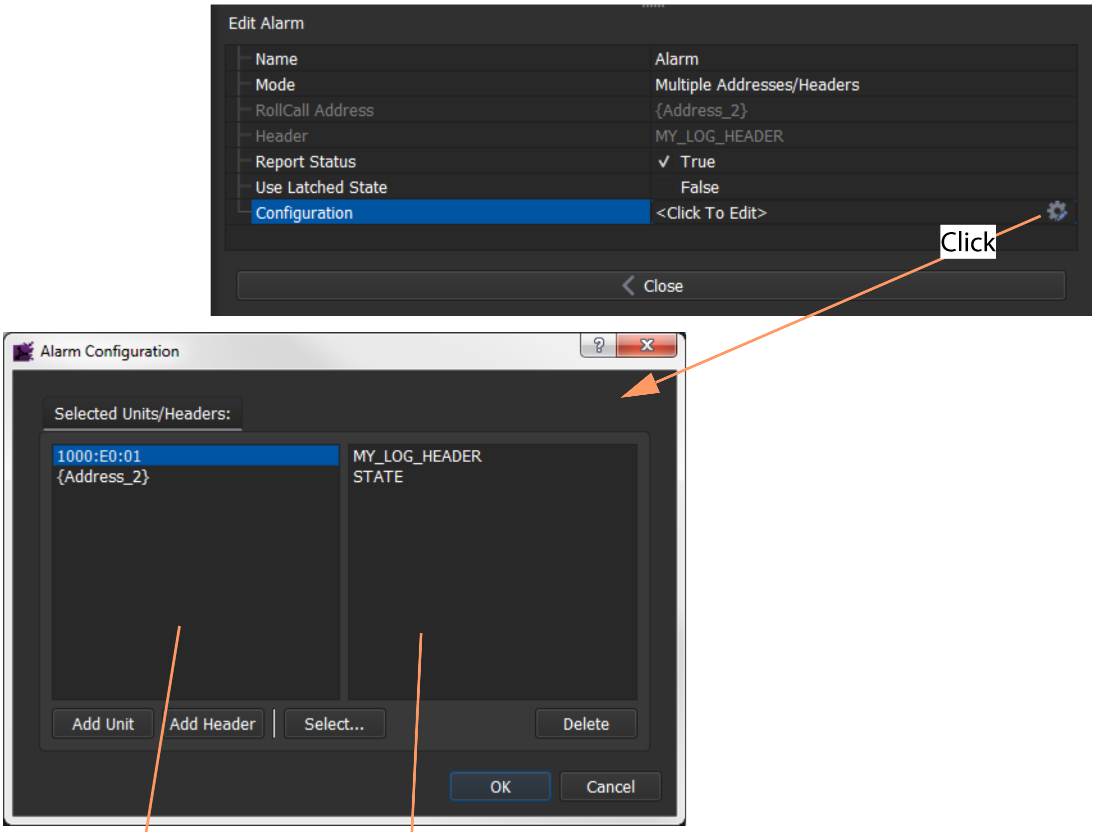


Configuration <Click To Edit>

Alarm Configuration Dialog

A list of RollCall addresses and corresponding Log Headers can be entered. The Alarm Behaviour aggregates all the Log Header values.

Click the ⚙ symbol to open the **Alarm Configuration** dialog.



Edit Alarm

Name	Alarm
Mode	Multiple Addresses/Headers
RollCall Address	{Address_2}
Header	MY_LOG_HEADER
Report Status	✓ True
Use Latched State	False
Configuration	<Click To Edit>

Alarm Configuration

Selected Units/Headers:

1000:E0:01	MY_LOG_HEADER
{Address_2}	STATE

Device/Unit address list

Log Field header list

Fig. 6-4: Alarm Configuration Dialog

To Enter a RollCall Address

In the **Alarm Configuration** dialog:

1. Click **Add Unit**. A new item is added into the list.
2. Double-click on the item and edit it, entering the required RollCall address. This may be a variable, for example {Address_2}.

This has entered a device/unit address into the address list, left-hand side.

The Log Field header list (right-hand side) for this new address is an empty list.
An empty Log Field header list means that *all* Log Field headers will be monitored.

If only specific Log Field headers are to be monitored, then these can be listed. A non-empty Log Field headers list (right-hand side) forms a white list of Log Field headers to be used, listing the Log Field headers to be used.

To Enter a Specific Log Field Header

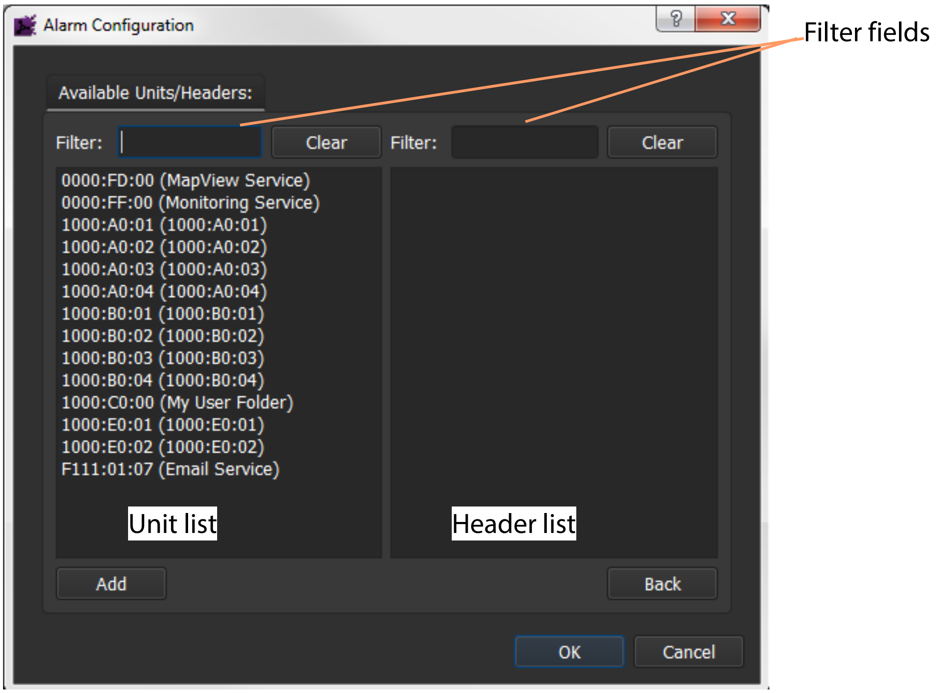
> Note: If a unit address has no specific log field headers listed in the **Alarm Configuration** dialog, then *all* alarms from the unit are used.

1 Click **Add Header**. A new item is added into the list.

2 Double-click on the item and edit it, entering the required Log Field header.
A variable name may be used, for example {MyHeader}.

Alternatively, to enter a Log Field header:

3 Click **Select...** and all accessible units are listed. See Figure 6-5.

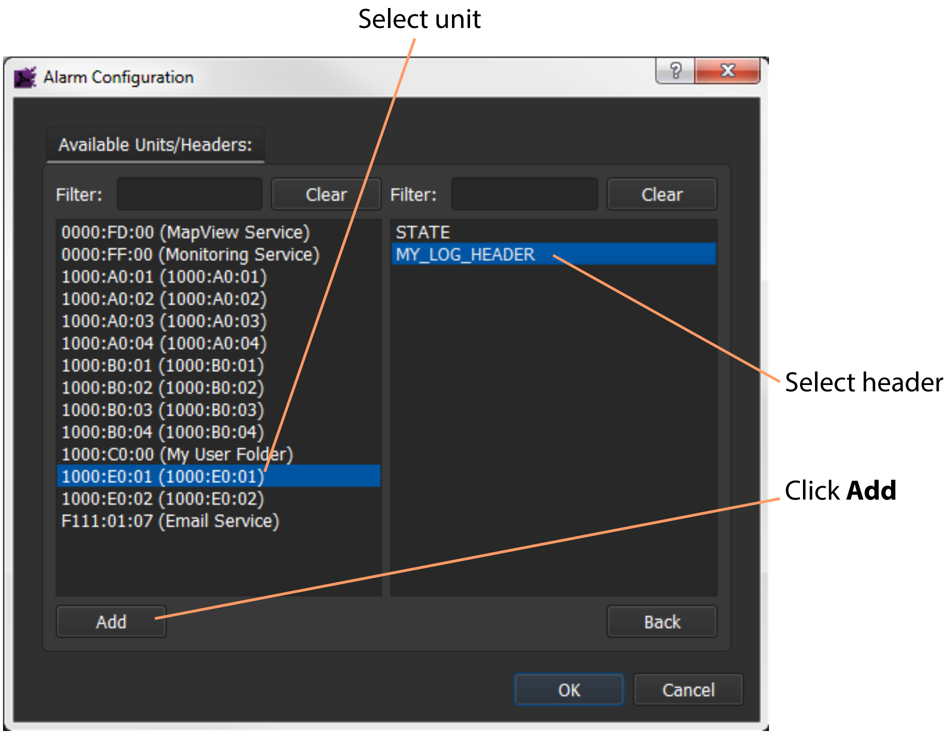


```
0000:FD:00 (MapView Service)
0000:FF:00 (Monitoring Service)
1000:A0:01 (1000:A0:01)
1000:A0:02 (1000:A0:02)
1000:A0:03 (1000:A0:03)
1000:A0:04 (1000:A0:04)
1000:B0:01 (1000:B0:01)
1000:B0:02 (1000:B0:02)
1000:B0:03 (1000:B0:03)
1000:B0:04 (1000:B0:04)
1000:C0:00 (My User Folder)
1000:E0:01 (1000:E0:01)
1000:E0:02 (1000:E0:02)
F111:01:07 (Email Service)
```

Fig. 6-5: Alarm Configuration Dialog - 'Select...'

4 Select the required unit item in the unit list on the left-hand side.
All the possible Log Field headers from the selected unit are then listed on the right-hand side.

Note: Use the **Filter** fields to filter the unit list or header list as required.



Select unit

Select header

Click **Add**

0000:FD:00 (MapView Service)
0000:FF:00 (Monitoring Service)
1000:A0:01 (1000:A0:01)
1000:A0:02 (1000:A0:02)
1000:A0:03 (1000:A0:03)
1000:A0:04 (1000:A0:04)
1000:B0:01 (1000:B0:01)
1000:B0:02 (1000:B0:02)
1000:B0:03 (1000:B0:03)
1000:B0:04 (1000:B0:04)
1000:C0:00 (My User Folder)
1000:E0:01 (1000:E0:01)
1000:E0:02 (1000:E0:02)
F111:01:07 (Email Service)

STATE
MY_LOG_HEADER

Fig. 6-6: Alarm Configuration Dialog

5 Select the required Log Field header.

6 Click **Add**.

7 Click **OK**.

To Delete a Specific Unit Item

1 Select a unit item.

2 Click **Delete**.

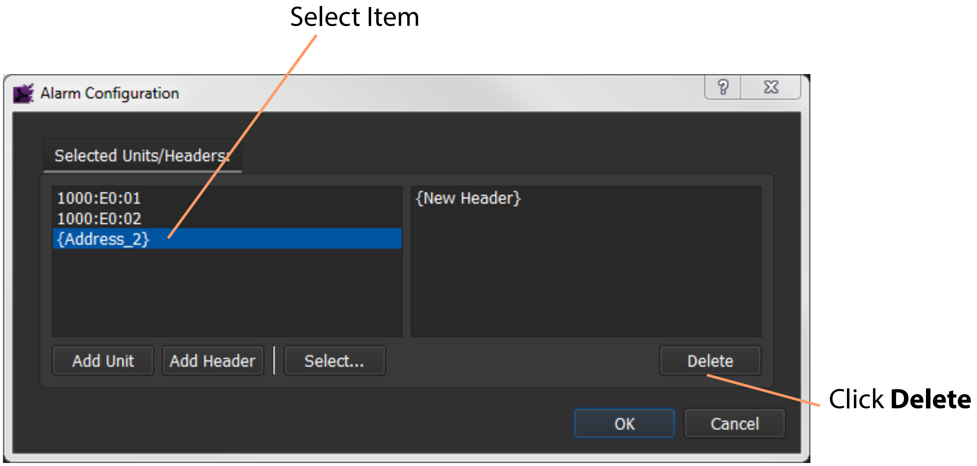
The unit item is deleted.

To Delete a Specific Header Item

1 Select a unit item
and then select a specific header item.

2 Click **Delete**.

The header item is deleted.



Select Item

Click **Delete**

1000:E0:01
1000:E0:02
{Address_2}

{New Header}

Fig. 6-7: Alarm Configuration Dialog

Alarm Acknowledge Behaviour

An **Alarm Acknowledge** Behaviour acknowledges alarms for RollCall addresses and their corresponding Log Fields.

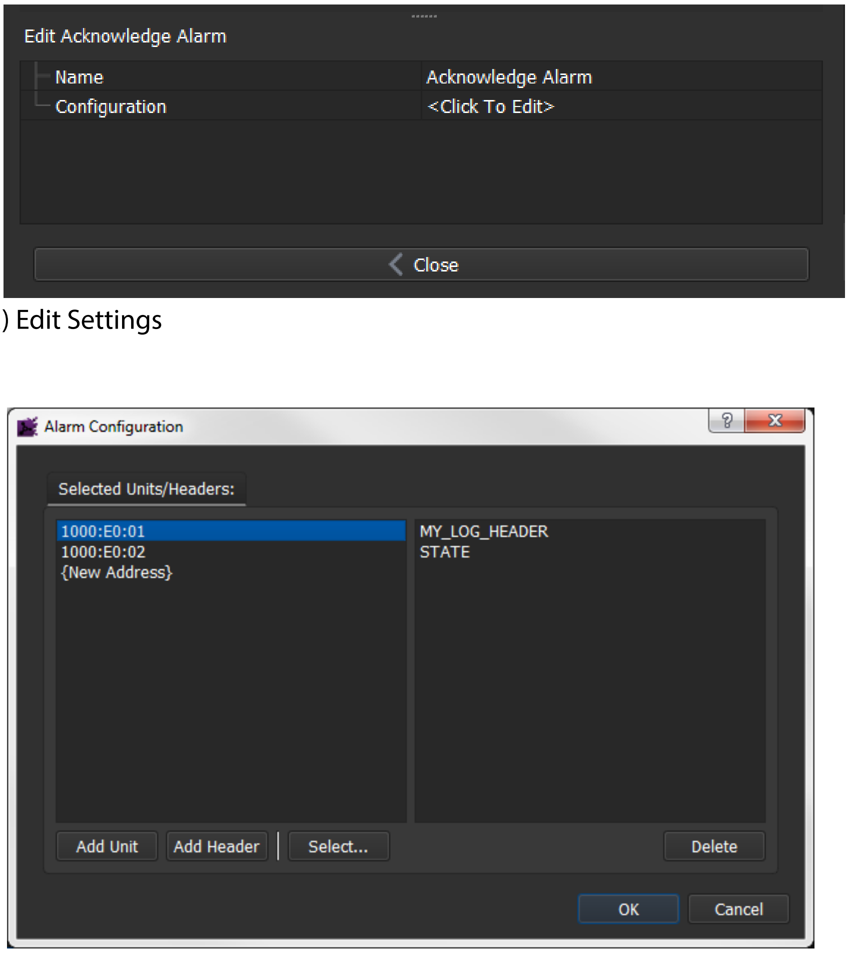
For example:

- an alarm state of **50** (**Warning**) is changed to **49** (**Acknowledged Warning**);
- an alarm state of **100** (**Fail**) is changed to **99** (**Acknowledged Fail**).

The Behaviour is configured for one or more unit addresses/Log Field headers. (The 'Configuration' setting operates like the 'Configuration' setting of the **Alarm** Behaviour, via an **Alarm Configuration** dialog. See Alarm Configuration Dialog, on page 152.)

> Note: An **Alarm Acknowledge** Behaviour acts on specific unit address and log field header alarm(s) in the Orbit services; it does not directly control an **Alarm** Behaviour.

The **Alarm Acknowledge** Behaviour may be given a name to identify it with.



Edit Acknowledge Alarm

Name	Acknowledge Alarm
Configuration	<Click To Edit>

a) Edit Settings

Alarm Configuration

Selected Units/Headers:

1000:E0:01	MY_LOG_HEADER
1000:E0:02	STATE
{New Address}	

b) Define Units/Headers

Fig. 6-8: Alarm Acknowledge Behaviour

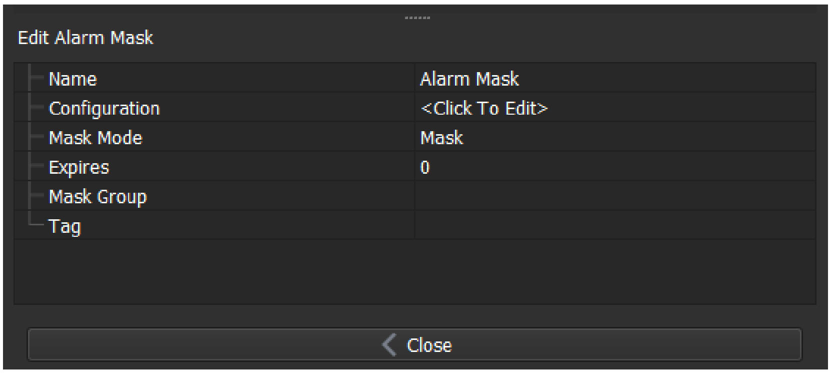
Alarm Mask Behaviour

An **Alarm Mask** Behaviour controls the masking of alarms from unit addresses and corresponding Log Field headers.

Alarm masking is useful in maintenance situations or in concessionary cases (for example, if a unit is intentionally operating temporarily with one power supply, then warning some related power supply warning/failure alarms can be masked).

An **Alarm Mask** Behaviour is configured to mask or to unmask alarms; it is configured for one or more unit(s)/log header(s).

Note: A masked alarm has an alarm state value of 0.
An unmasked alarm has an alarm state value in the range 1 to 100.
(See State Value 0 to 100, on page 17.)



Edit Alarm Mask

Name	Alarm Mask
Configuration	<Click To Edit>
Mask Mode	Mask
Expires	0
Mask Group	
Tag	

Fig. 6-9: Alarm Mask Behaviour

Table 6-4: Alarm Mask Properties

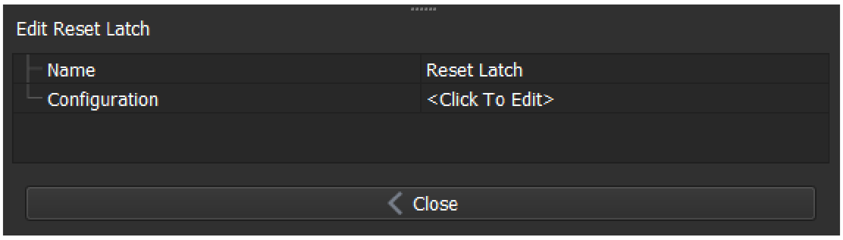
Property	**Description**
Name	Enter a name to identify the Behaviour.
Configuration	Text box. Alarms to be masked are defined here. Specify the unit address and, optionally, the Log Field headers. • Click the ⚙ symbol to open the **Alarm Configuration** dialog to enter RollCall addresses and Log Field headers. See Alarm Configuration Dialog, on page 152.

Table 6-4: Alarm Mask Properties (continued)

Property	Description
Mask Mode	Drop-down box. Select mask mode. • **Mask** - the specified alarms are masked. • **UnMask** - the specified alarms are unmasked. • **Mask Until Green** - mask until the status is 'OK' (Green color indication). • **Mask Until Time** - mask the specified alarms for a period of time. Once the time period has elapsed, the device is included into the calculation. (See main menu 'Tools > Options > Monitoring' on the 'Masking' tab, Tools > Options > Monitoring - Masking tab, on page 14.) • **Mask Group** - Mask a group of units/devices. • **UnMask Group** - Unmask a group of units/devices. • **Apply Invert** - Select to set all 'Warnings' and 'Failures' to show an 'OK' status. • **Remove invert** - Select to revert all "inverted"' alarm states back to show their original state.
Expires	Text field (seconds) Enter the number of seconds over which the mask should be applied. After this period, it is removed automatically. If mask is to be applied indefinitely until it is removed, enter 0.
Mask Group	Text field. Specify which Mask Group to mask/unmask. **Note:** 'Mask Groups' are defined as part of the system services running on the Log Server.
Tag	Text field. Specify which Tagged Mask to operate on. See Network View, on page 21.

Reset Latch Behaviour

A **Reset Latch** Behaviour unlatches one or more latched alarm(s). This operates on the Orbit services and the result of this will be subsequently seen by all **Alarm** Behaviours monitoring a device’s latched alarm state.



Edit Reset Latch

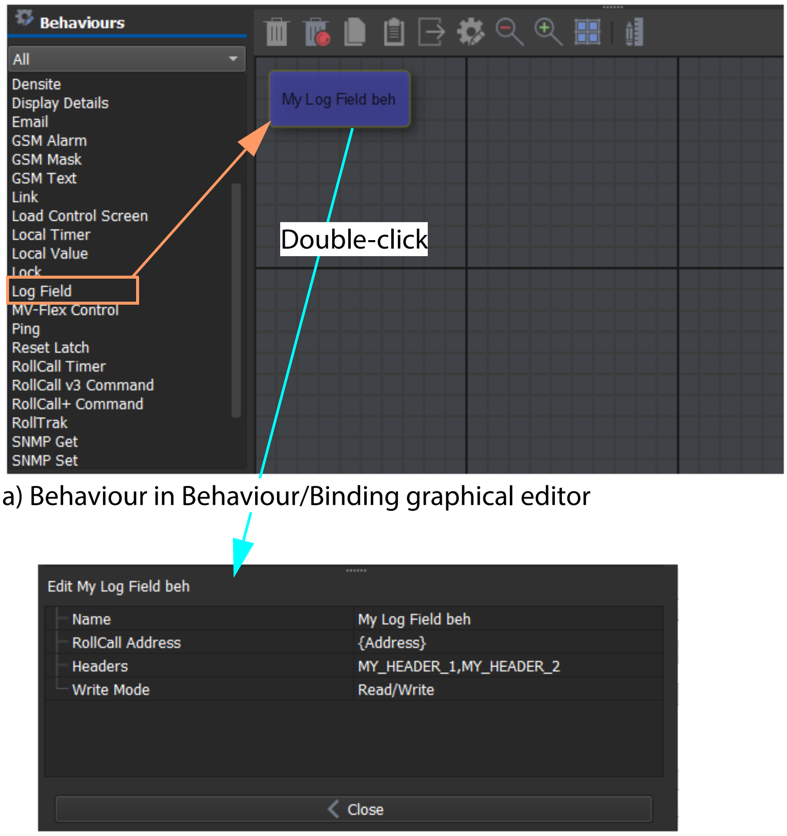
Name	Reset Latch
Configuration	<Click To Edit>

Fig. 6-10: Reset Latch Behaviour

The ‘Configuration’ setting operates like the ‘Configuration’ setting of the Alarm Behaviour via an **Alarm Configuration** dialog. See Alarm Configuration Dialog, on page 152.

Log Field Behaviour

A **Log Field** Behaviour enables a MapView screen to use Log Field messages from RollCall address - i.e. from a card, a unit or an Orbit Monitoring Service (OMS). A Log Field can be read and/or written to by a MapView screen with the **Log Field** Behaviour.



a) Behaviour in Behaviour/Binding graphical editor

Edit My Log Field beh

Name	My Log Field beh
RollCall Address	{Address}
Headers	MY_HEADER_1,MY_HEADER_2
Write Mode	Read/Write

b) Edit Properties

Fig. 6-11: Log Field Behaviour:
a) In Behaviour/Binding Graphical Editor.
b) Edit Properties.

Properties are described in Table 6-5.

Table 6-5: Log Field Behaviour Properties

Property	**Description**
Name	Enter a name to identify the Behaviour.
RollCall Address	Text box. Enter a RollCall address. The Behaviour will look for Log Field headers with this address. **Note:** This may be set to be a variable, for example, {Address}.
Headers	A list of Log Field headers for the Behaviour to expose on the MapView screen. Either: • enter a comma-separated list of Log Field headers; or

Table 6-5: Log Field Behaviour Properties (continued)

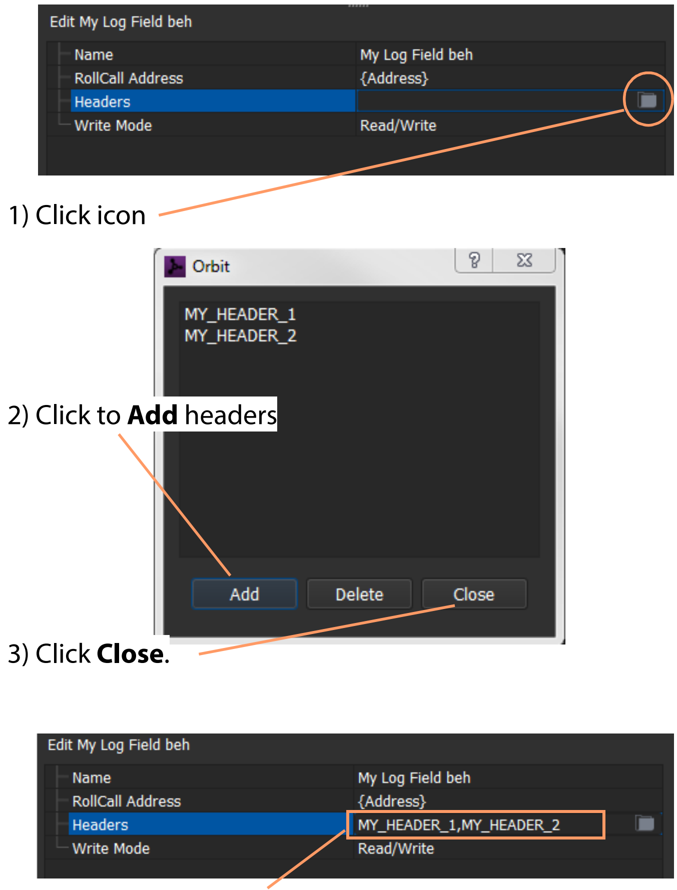
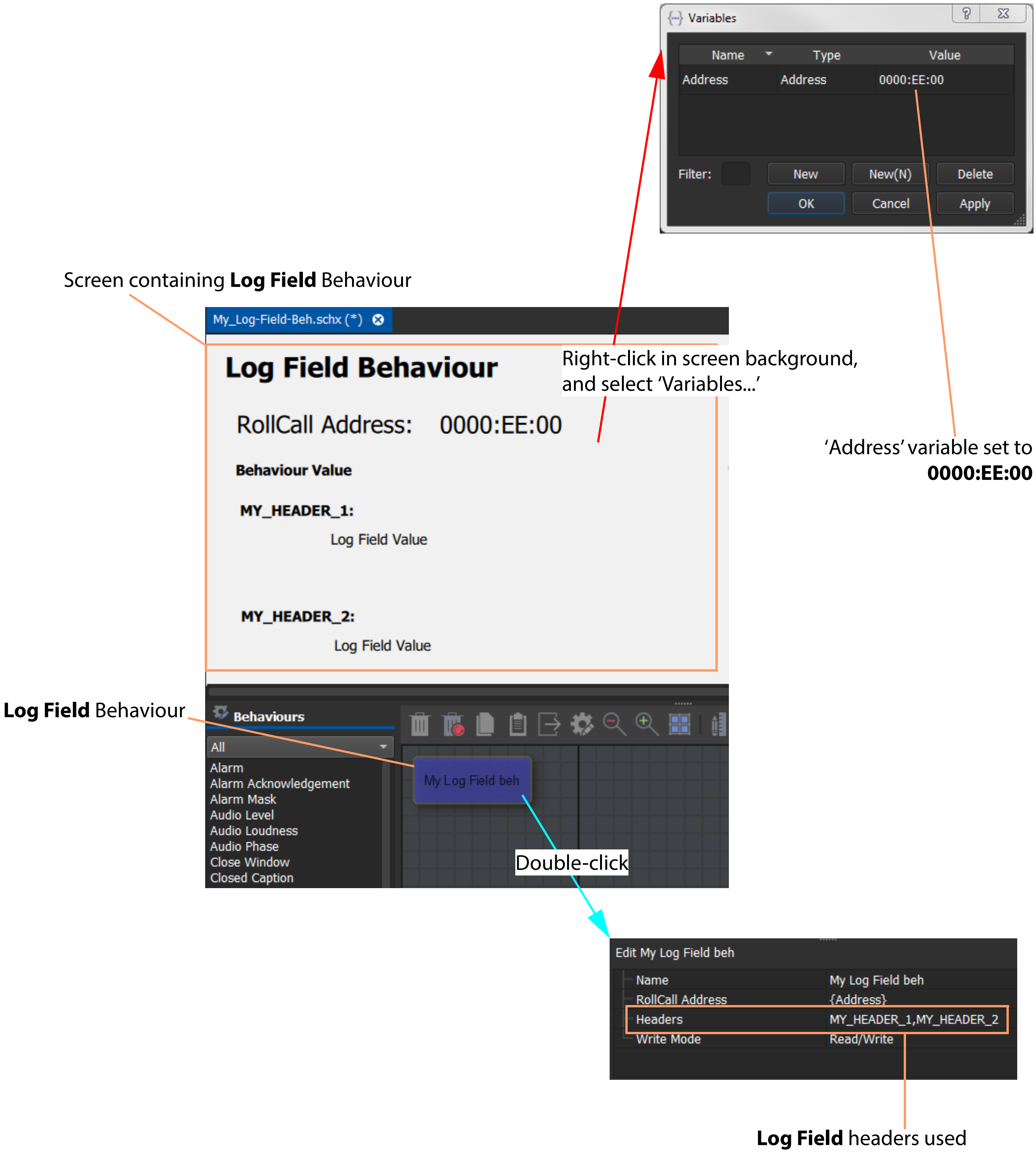
Property	Description
	or • click the icon in the ‘Headers value’ field and add headers into the resulting dialog:  Edit My Log Field beh Name: My Log Field beh RollCall Address: {Address} Headers: Write Mode: Read/Write 1) Click icon Orbit MY_HEADER_1 MY_HEADER_2 2) Click to **Add** headers 3) Click **Close**. Edit My Log Field beh Name: My Log Field beh RollCall Address: {Address} Headers: MY_HEADER_1,MY_HEADER_2 Write Mode: Read/Write 4) Comma-separated list entered as ‘Headers’ property value.
Write Mode	Drop-down box. Specify Behaviour’s read/write capability for the Log Fields it handles. Options: • **Read Only** - Behaviour exposes the Log field values on the MapView screen but will not write a changed value back. • **Write Only** - Behaviour will write a changed value back but will not update the Log Field value if it is changed outside of the Orbit MapView screen. • **Read/Write** - Synchronizes the Behaviour with the Log Field in both directions (into and out of the Orbit MapView screen). See **Note 1**.
Note 1:	When **Write Mode** is set to ‘Write Only’ or ‘Read/Write’ then a **Log Field** Behaviour becomes writeable; any changes that Orbit MapView makes to the Behaviour value will be sent back to the Orbit Monitoring Service and other parts of the system.

Figure 6-12 and Figure 6-13 show a **Log Field** Behaviour instanced on a MapView screen with an accompanying **Label** Binding. And Figure 6-14 shows this being exercised.



Variables

Name	Type	Value
Address	Address	0000:EE:00

Screen containing **Log Field** Behaviour

Right-click in screen background, and select 'Variables...'

'Address' variable set to **0000:EE:00**

Log Field Behaviour

RollCall Address: 0000:EE:00

Behaviour Value

MY_HEADER_1:
Log Field Value

MY_HEADER_2:
Log Field Value

Log Field Behaviour

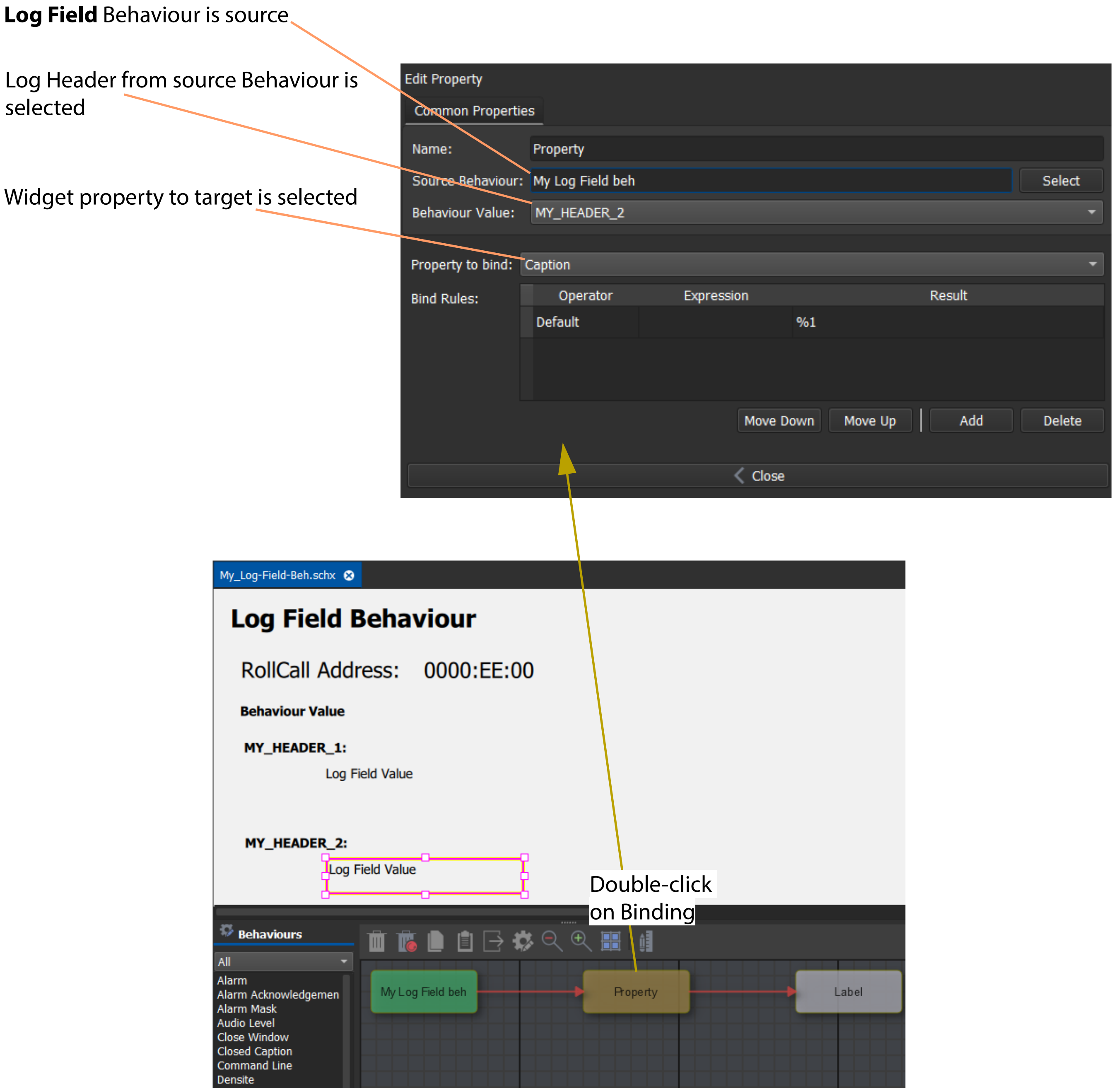
Double-click

Edit My Log Field beh

Name	My Log Field beh
RollCall Address	{Address}
Headers	MY_HEADER_1,MY_HEADER_2
Write Mode	Read/Write

Log Field headers used

Fig. 6-12: Log Field Behaviour on MapView Screen



Log Field Behaviour is source

Log Header from source Behaviour is selected

Widget property to target is selected

Double-click on Binding

Edit Property

Common Properties

Name: Property

Source Behaviour: My Log Field beh

Behaviour Value: MY_HEADER_2

Property to bind: Caption

Bind Rules:

Operator	Expression	Result
Default		%1

My_Log-Field-Beh.schx

Log Field Behaviour

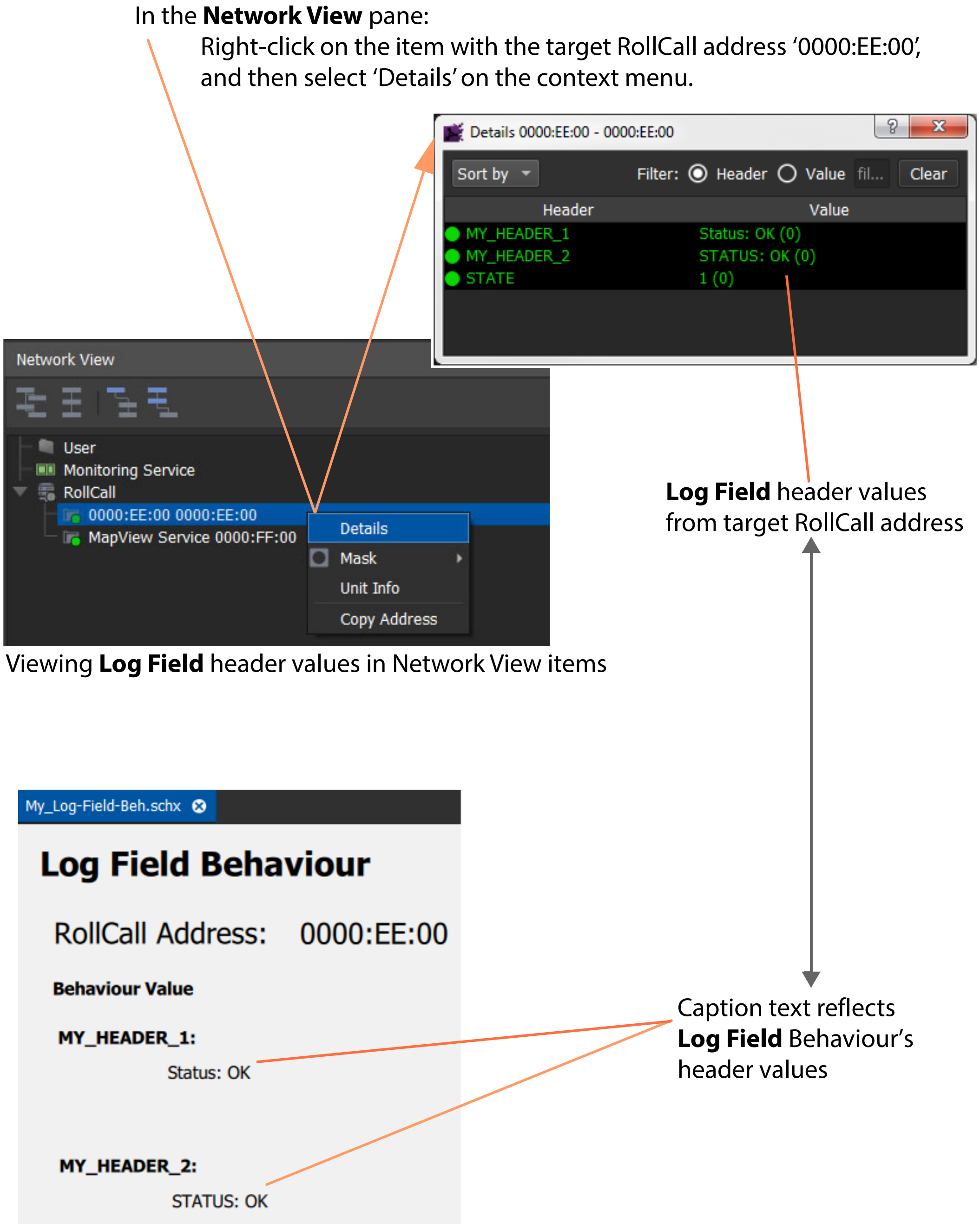
RollCall Address: 0000:EE:00

Behaviour Value

MY_HEADER_1:
Log Field Value

MY_HEADER_2:
Log Field Value

Fig. 6-13: Property Binding for Label Caption (used with Log Field Behaviour)



In the **Network View** pane:
Right-click on the item with the target RollCall address ‘0000:EE:00’, and then select ‘Details’ on the context menu.

Details 0000:EE:00 - 0000:EE:00

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
🟢 MY_HEADER_1	Status: OK (0)
🟢 MY_HEADER_2	STATUS: OK (0)
🟢 STATE	1 (0)

Log Field header values from target RollCall address

a) Viewing **Log Field** header values in Network View items

My_Log-Field-Beh.schx

Log Field Behaviour

RollCall Address: 0000:EE:00

Behaviour Value

MY_HEADER_1:
Status: OK

MY_HEADER_2:
STATUS: OK

Caption text reflects **Log Field** Behaviour’s header values

b) Exercising **Log Field** Behaviour

Fig. 6-14: Log Field Being Exercised

#Hash Field# Syntax

It is quite common to need to read or to write to Log Field values in a MapView screen in a control and monitoring application. This can result in many instances of a **Log Field** Behaviour. Orbit MapView supports a quick way of creating **Log Field** Behaviours; this makes the handling of a large number of Log Field values much easier and less time consuming when designing an Orbit MapView screen.

Specifying Log Field Header

A #HASH_FIELD# syntax may be used in value fields in Orbit (for property values or Behaviour values. The required Log Field header name is written between '#' characters.
For example:

- #MSG#
- #AESREF#

Behind the scenes, Orbit MapView adds the necessary Behaviours and Bindings to be able to monitor the required Log Field. If the Log field changes at MapView screen run time, it will also change on the MapView screen.

Specifying RollCall Address

When using hash fields, the RollCall address to be used will be taken from:

- a screen variable called 'Address';
- if this does not exist,
  it will be taken from a MapView project variable of the same name; and
- if neither exist,
  the hash field's Behaviour is not defined and address 0000:00:00 is used.

It is possible to specify the RollCall address in a hash field. All or part of the address may be specified:

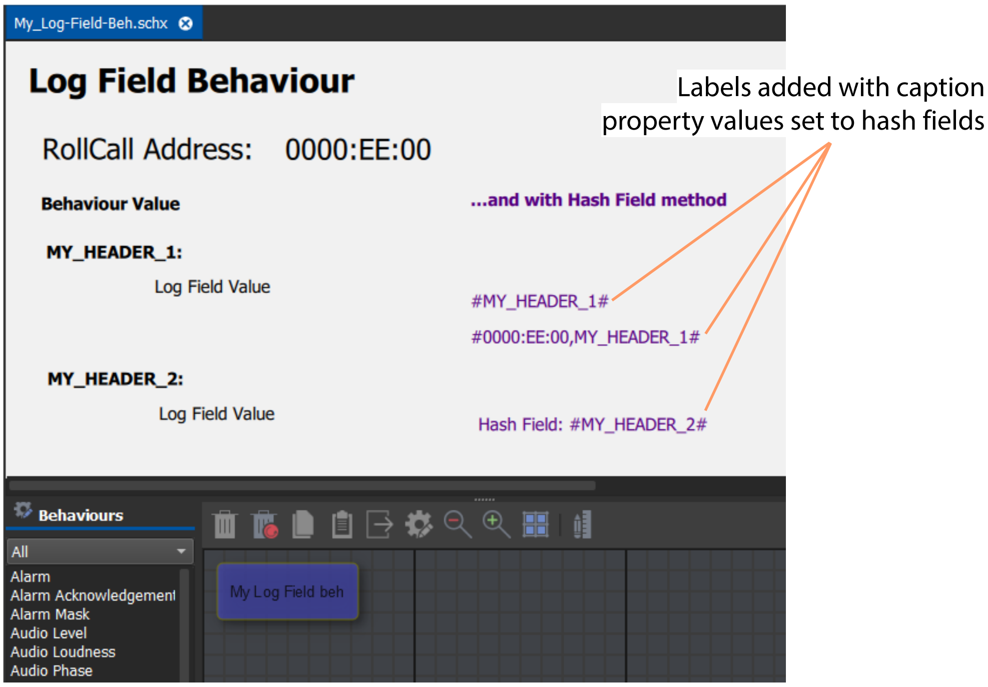
- #0000:AA:00,MSG# - RollCall address '0000.AA.00' is used for the hash field's associated Log Field Behaviour.

To just specify *part* of the address:

- #0000:AA:??,MSG# - With screen/project variable 'Address' set to '1000:00:02', then RollCall address of '0000:AA:02' is used in the hash field Behaviour.

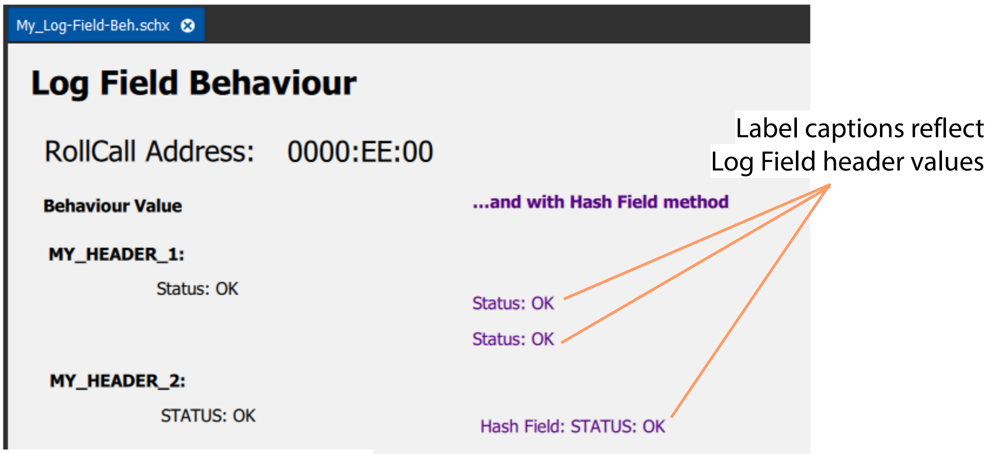
Example

Figure 6-15 shows the hash fields being used, when added to the example shown in Figure 6-14 on page 163.



Labels added with caption property values set to hash fields

a) Using the #Hash Field# Method for Accessing Log Field Header Values



Label captions reflect Log Field header values

b) Exercising #Hash Field# Method

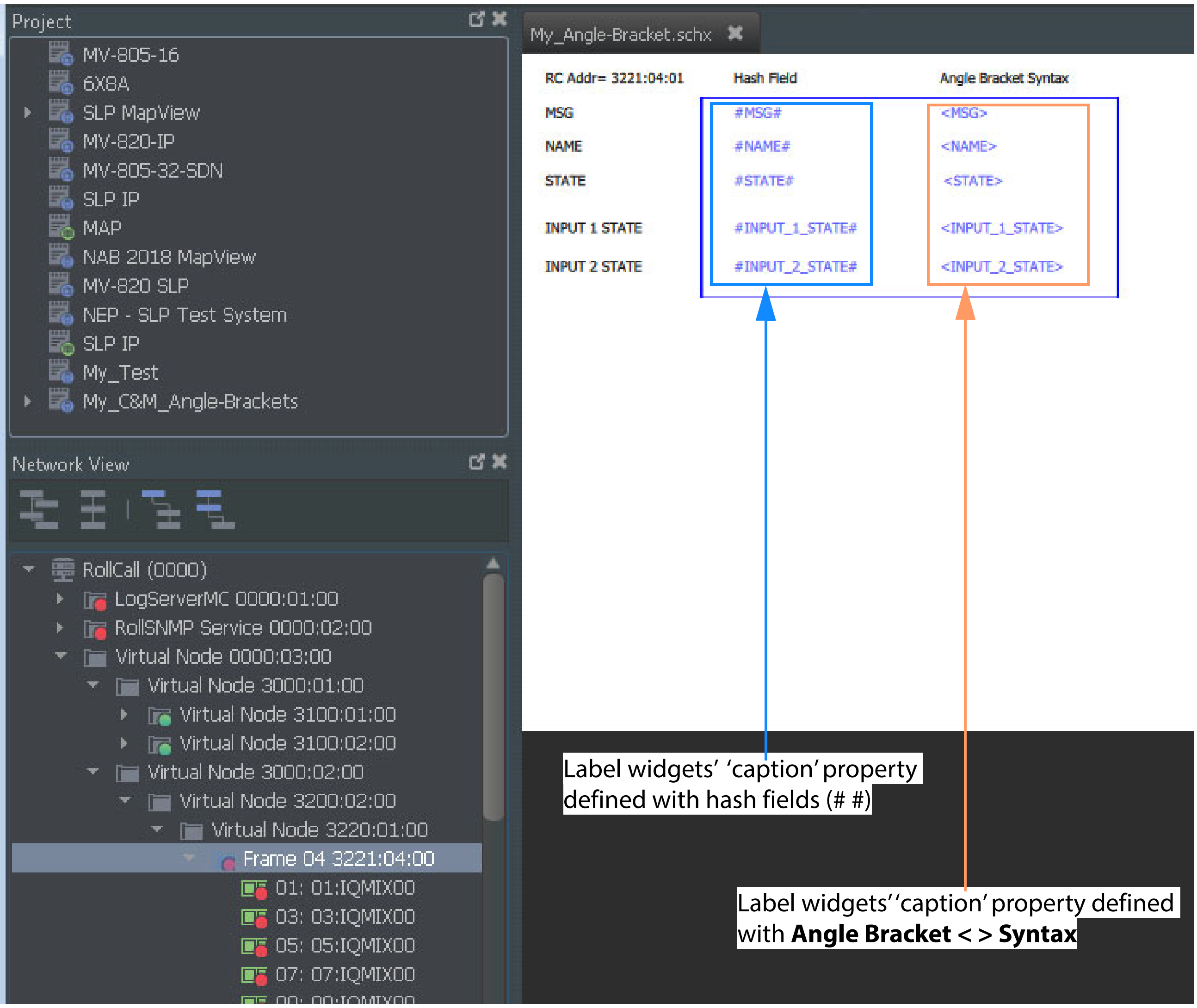
Fig. 6-15: #Hash Field# Method
a) Using the #Hash Field# Method for Accessing Log Field Header Values.
b) Exercising #Hash Field# Method.

<Angle-Bracket> Syntax

The hash field syntax (#Hash Field# Syntax, on page 164) invokes a **Log Field** Behaviour 'behind-the-scenes' in Orbit.

The 'angle-bracket' syntax (<UNIT_STATUS>) invokes both a **Log Field** Behaviour and an **Alarm** Behaviour 'behind-the-scenes' in Orbit. The State value of the item being monitored is then used to apply (color) formatting to the log field text, according to the status of the State.

This < > syntax is shown in Figure 6-16 and the resulting exercised MapView screen is shown in Figure 6-17.

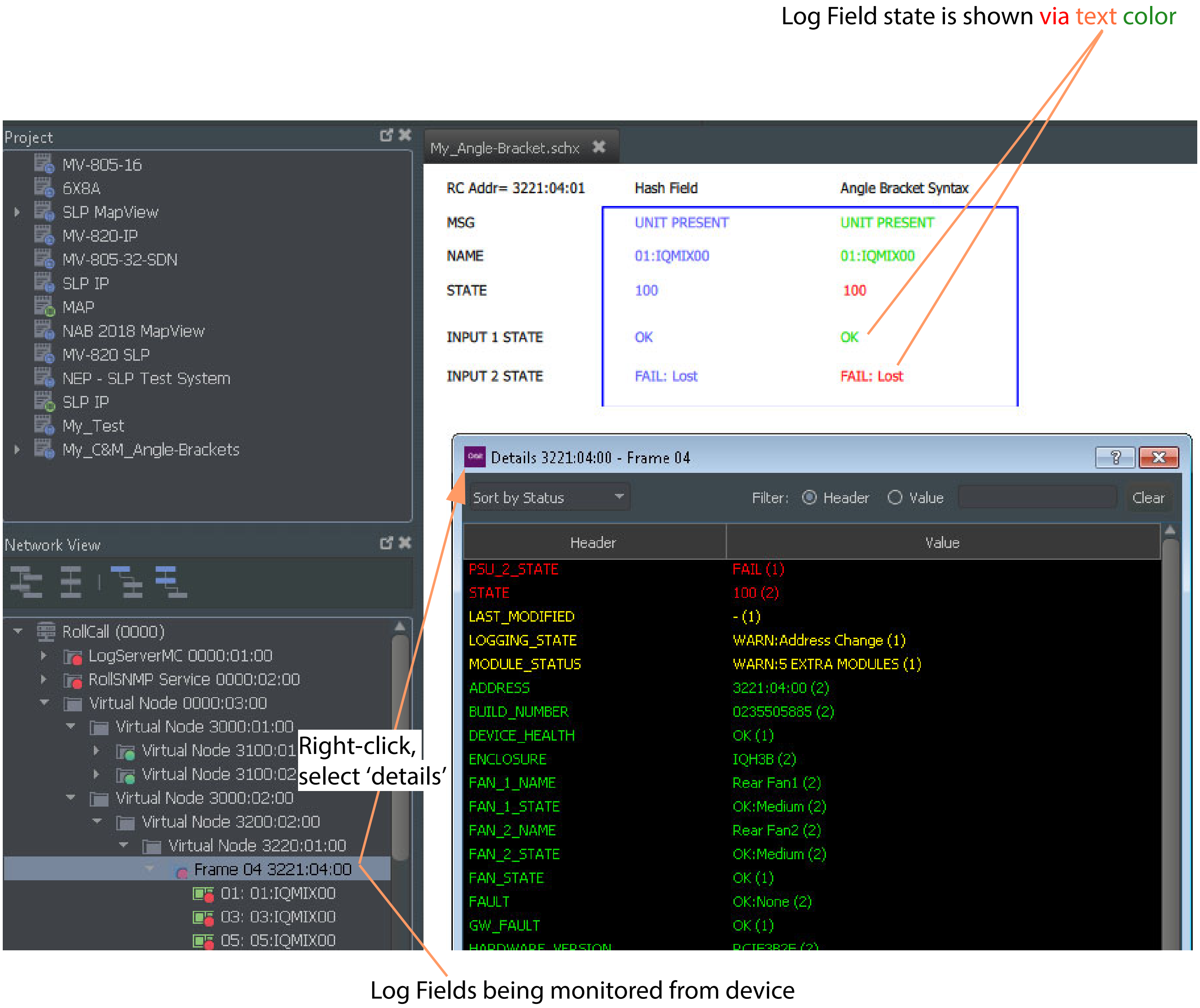


RC Addr= 3221:04:01	Hash Field	Angle Bracket Syntax
MSG	#MSG#	<MSG>
NAME	#NAME#	<NAME>
STATE	#STATE#	<STATE>
INPUT 1 STATE	#INPUT_1_STATE#	<INPUT_1_STATE>
INPUT 2 STATE	#INPUT_2_STATE#	<INPUT_2_STATE>

Label widgets' 'caption' property defined with hash fields (# #)

Label widgets' 'caption' property defined with **Angle Bracket < > Syntax**

Fig. 6-16: MapView Screen with Angle-Bracket < > Syntax



Log Field state is shown via text color

RC Addr= 3221:04:01	Hash Field	Angle Bracket Syntax
MSG	UNIT PRESENT	UNIT PRESENT
NAME	01:IQMIX00	01:IQMIX00
STATE	100	100
INPUT 1 STATE	OK	OK
INPUT 2 STATE	FAIL: Lost	FAIL: Lost

Right-click, select ‘details’

Header	Value
PSU_2_STATE	FAIL (1)
STATE	100 (2)
LAST_MODIFIED	- (1)
LOGGING_STATE	WARN:Address Change (1)
MODULE_STATUS	WARN:5 EXTRA MODULES (1)
ADDRESS	3221:04:00 (2)
BUILD_NUMBER	0235505885 (2)
DEVICE_HEALTH	OK (1)
ENCLOSURE	IQH3B (2)
FAN_1_NAME	Rear Fan1 (2)
FAN_1_STATE	OK:Medium (2)
FAN_2_NAME	Rear Fan2 (2)
FAN_2_STATE	OK:Medium (2)
FAN_STATE	OK (1)
FAULT	OK:None (2)
GW_FAULT	OK (1)
HARDWARE_VERSION	RCIE3B2E (2)

Log Fields being monitored from device

Fig. 6-17: Exercised Angle-Bracket < > Syntax

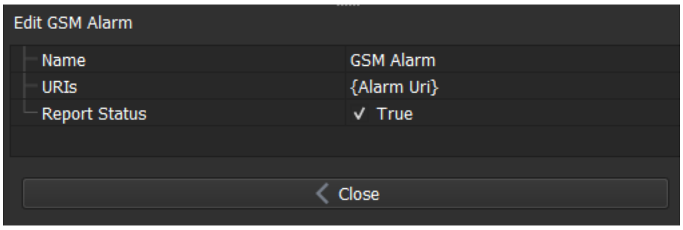
GSM Behaviours

Three **GSM** Behaviours are provided to allow iControl and Densité alarms to be accessed and used on Orbit screens.

- GSM Alarm Behaviour, on page 168 – Exposes the GSM alarm state inside an Orbit screen.
- GSM Text Behaviour, on page 169 – Exposes the textual value of a GSM alarm inside an Orbit screen.
- GSM Mask Behaviour, on page 169 – Exposes current alarm mask state and allows a user to mask/unmask GSM alarms from inside a screen.

GSM Alarm Behaviour

The **GSM Alarm** Behaviour enables Orbit MapView screens to monitor alarms from Densité devices. It can be configured for a folder or for an alarm and then its value will be kept up to date to reflect the current state of that alarm.



Edit GSM Alarm

Name	GSM Alarm
URIs	{Alarm Uri}
Report Status	✓ True

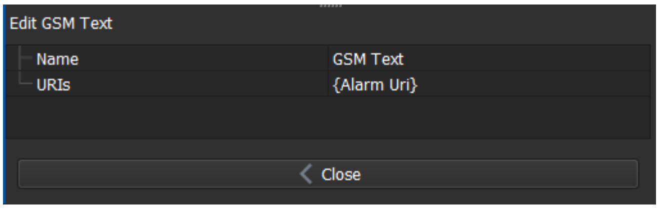
Fig. 6-18: GSM Alarm Behaviour Edit Properties.

Table 6-6: GSM Alarm Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
URIs	Text box. Enter URI reference(s) to a GSM alarm or to a **Network View** folder that is to be monitored by the Behaviour. If multiple references are entered, then the aggregate state of all is used.
Report Status	Check box. Select to enable the alarm state to be used by the Orbit MapView service to calculate overall state of the screen.

GSM Text Behaviour

The **GSM Text** Behaviour enables Orbit MapView screens to monitor alarms from Densité devices. It is used for textual alarms and can be configured for a folder, or for an alarm, and then its value will be kept up to date to reflect the current state of that alarm.



Edit GSM Text

Name: GSM Text
URIs: {Alarm Uri}

Fig. 6-19: GSM Text Behaviour Edit Properties.

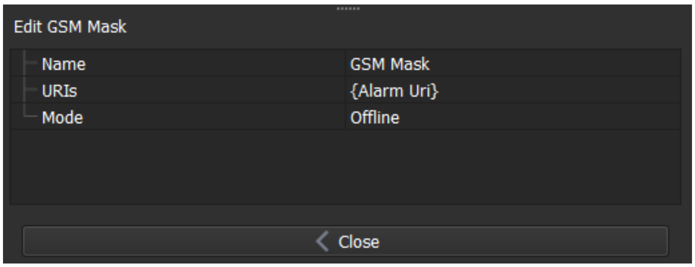
Table 6-7: GSM Text Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
URIs	Text box. Enter URI reference(s) to a GSM alarm or to a **Network View** folder that is to be monitored by the Behaviour. If multiple references are entered, then the aggregate state of all is used.

GSM Mask Behaviour

The **GSM Mask** Behaviour enables Orbit MapView screens to manage masking of alarms for Densité devices. Masking modes can be changed from Orbit MapView screens.

The **GSM Mask** Behaviour is executable; it must be configured and then actively executed (using an **Event** Binding) on an Orbit MapView screen.



Edit GSM Mask

Name: GSM Mask
URIs: {Alarm Uri}
Mode: Offline

Fig. 6-20: GSM Mask Behaviour Edit Properties.

Table 6-8: GSM Text Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
URIs	Text box. Enter URI reference(s) to a GSM alarm or to a **Network View** folder. These alarm(s) will have their masking mode changed by the Behaviour.
Mode	Drop down box. Select the masking mode (known as 'operational mode' inside the Grass Valley iControl navigator tool) to be applied when the Behaviour is executed. Options, set masking mode to: • **Offline** • **Online** • **In Maintenance** - puts the alarm(s) into maintenance mode. • **Not In Maintenance** - brings alarm(s) out of maintenance mode. • **Inverted** • **Not Inverted**

GSM Mask Behaviour Value

The **GSM Mask** Behaviour has a value. This is an array of 3 Boolean values indicating the current mask mode. The array elements are:

1. **Offline** mode - Offline enabled = 'True'; 'False' if disabled.
2. **In Maintenance** mode - In maintenance = 'True'; 'False' if not.
3. **Mask Invert** - the current mask invert setting.

> Note: If multiple alarm URIs are set on the **GSM Mask** Behaviour
> and if the mask **Mode** is inconsistent, then
> the Behaviour value element will be 'True' if *any* of the multiple alarms is in that mode.

These element values can be used to drive the state of widgets.

SNMP Behaviours

Two **SNMP** Behaviours are provided to allow Orbit to access device information from SNMP-capable devices:

- SNMP Get Behaviour, on page 171 – Reads an SNMP value from a device.
- SNMP Set Behaviour, on page 172 – Writes an SNMP value to a device.

These Behaviours enable the reading and writing of single device values via the SNMP protocol by an Orbit screen.

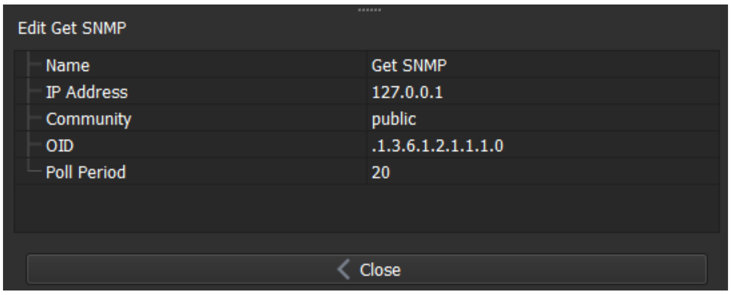
> Note: The **SNMP Get** and **Set** Behaviours are designed for simple querying of a value and setting a value. For wide-scale monitoring of multiple SNMP devices a product such as Grass Valley RollSNMP or GSM should be used.

An SNMP-capable device stores information about its status and configuration in a management information base (MIB), which has a tree-like structure. Each data element in the MIB is uniquely referenced by an object identifier (OID). Each OID identifies an element which can be read or set via SNMP.

> Note: For specific information on a device value's OID:
> - refer to vendor device documentation; or
> - use a MIB Browser tool to determine.

SNMP Get Behaviour

The **SNMP Get** Behaviour reads a single OID value from a device and returns the value of the OID. This value can be used on Orbit screens with other Bindings and Behaviours.



Edit Get SNMP

Name	Get SNMP
IP Address	127.0.0.1
Community	public
OID	.1.3.6.1.2.1.1.1.0
Poll Period	20

Fig. 6-21: SNMP Get Behaviour Edit Properties.

Table 6-9: SNMP Get Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
IP Address	Text box. Enter IP address of the device to query via SNMP.
Community	Text box. Enter SNMP community string. See vendor documentation about for the device to be accessed. **Note:** If **Community** setting is incorrect, the OID value cannot be read.

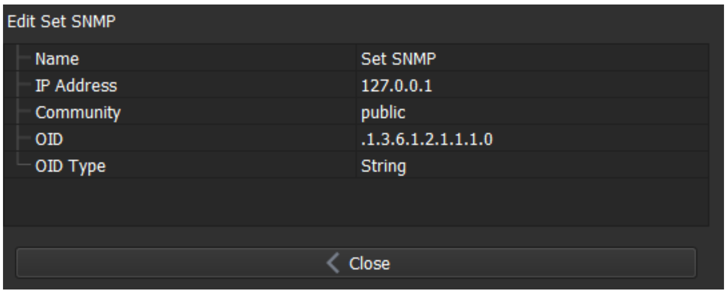
Table 6-9: SNMP Get Behaviour Properties (continued)

Property	Description
OID	Text box. Enter object identifier (OID) of the data value in the MIB.
Poll Period	Text box. Enter the time (in seconds) between SNMP Get commands.

SNMP Set Behaviour

The **SNMP Set** Behaviour writes a single OID value to a device. The Behaviour can be used with Bindings to set a particular OID value from an Orbit screen.

> Note: The **SNMP Set** Behaviour will not set an OID value again if the same value is applied to it.



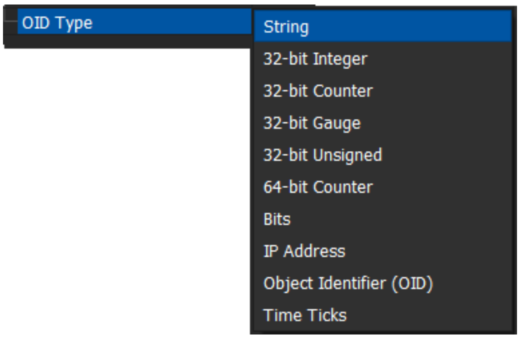
Name	Set SNMP
IP Address	127.0.0.1
Community	public
OID	.1.3.6.1.2.1.1.1.0
OID Type	String

Fig. 6-22: SNMP Set Behaviour Edit Properties.

Table 6-10: SNMP Set Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
IP Address	Text box. Enter IP address of the device to write to via SNMP.
Community	Text box. Enter SNMP community string. See vendor documentation about for the device to be accessed. Note: If **Community** is incorrect, the OID value will not be set (written to).

Table 6-10: SNMP Set Behaviour Properties (continued)

Property	Description
OID	Text box. Enter object identifier (OID) of the data value in the MIB.
OID Type	Drop down box. Select the data type of the OID value being written (set). Options: OID Type String 32-bit Integer 32-bit Counter 32-bit Gauge 32-bit Unsigned 64-bit Counter Bits IP Address Object Identifier (OID) Time Ticks See vendor documentation for the OID type.

Link Behaviour

Introduction

In an Orbit MapView project there will typically be many screens with some hierarchical structure. The **Link** Behaviour allows users to navigate around the MapView project screen hierarchy. A **Link** Behaviour may be attached to a button which links to a screen below in the hierarchy. See Figure 6-23.

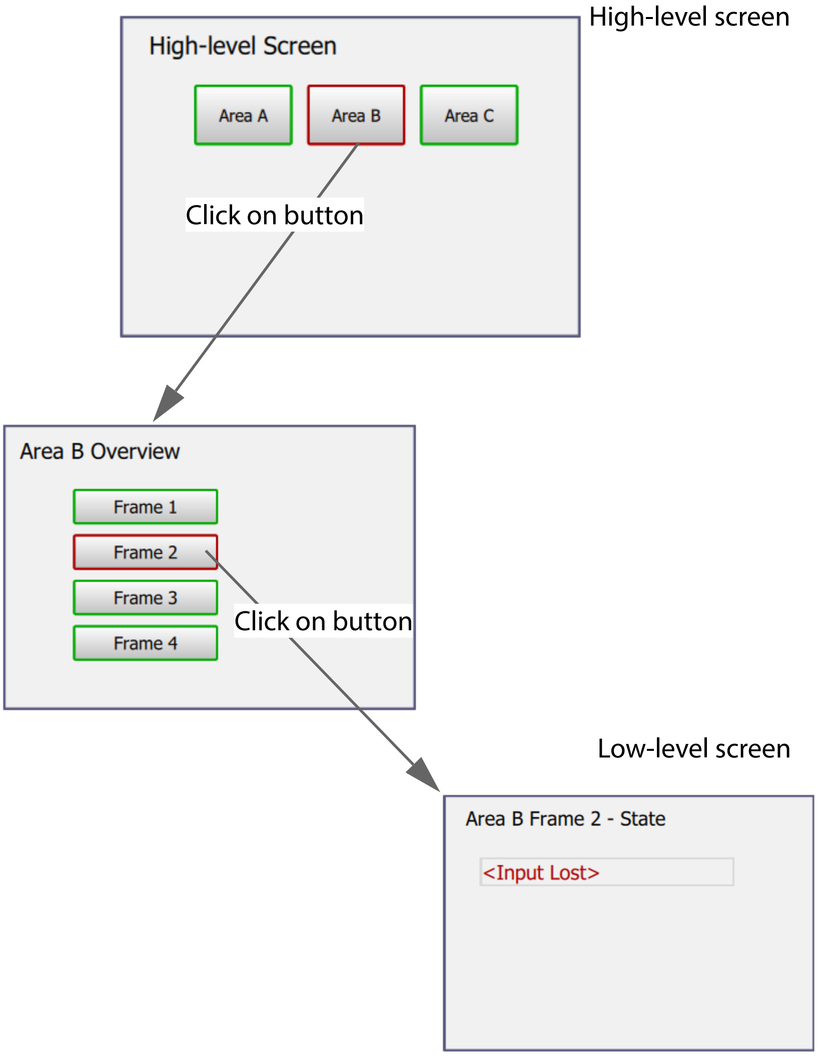


Fig. 6-23: MapView Screen Hierarchy Example

State Information

MapView screens may contain differing amounts of live information about device(s) and/or module(s); this may range from complex details about a device/module to a simple high-level status with an overall 'good/bad' indication. For monitoring applications, the user requires to be presented with up to date information, whichever screen is being shown.

A MapView monitoring project can be designed so that when any error occurs in a low-level screen, the 'error state' is reflected up the project screen hierarchy and shown at higher levels. For example, a device error shown in a low-level screen can cause a higher-level button to go red, indicating an 'error state' on the low-level screen. The states of all devices in a system being monitored will change over time and, therefore, all states shown on MapView screens are 'live' and are actively changing.

The live 'states' of each MapView screen all need to be known to an Orbit MapView project to allow a user to quickly see any errors or warnings and then drill down to the source of the problem (for example, a lost input to a device).

Schematic State

Each MapView screen file may have a 'state' based on information within it. In order to have a 'state', a screen file must have at least one **Alarm** Behaviour (defined explicitly, or defined using the shorthand <> notation).

The **Schematic State** is taken to be the aggregate of each of the **Alarm** Behaviour values within the screen. See Alarm Behaviour, on page 150.

Link State

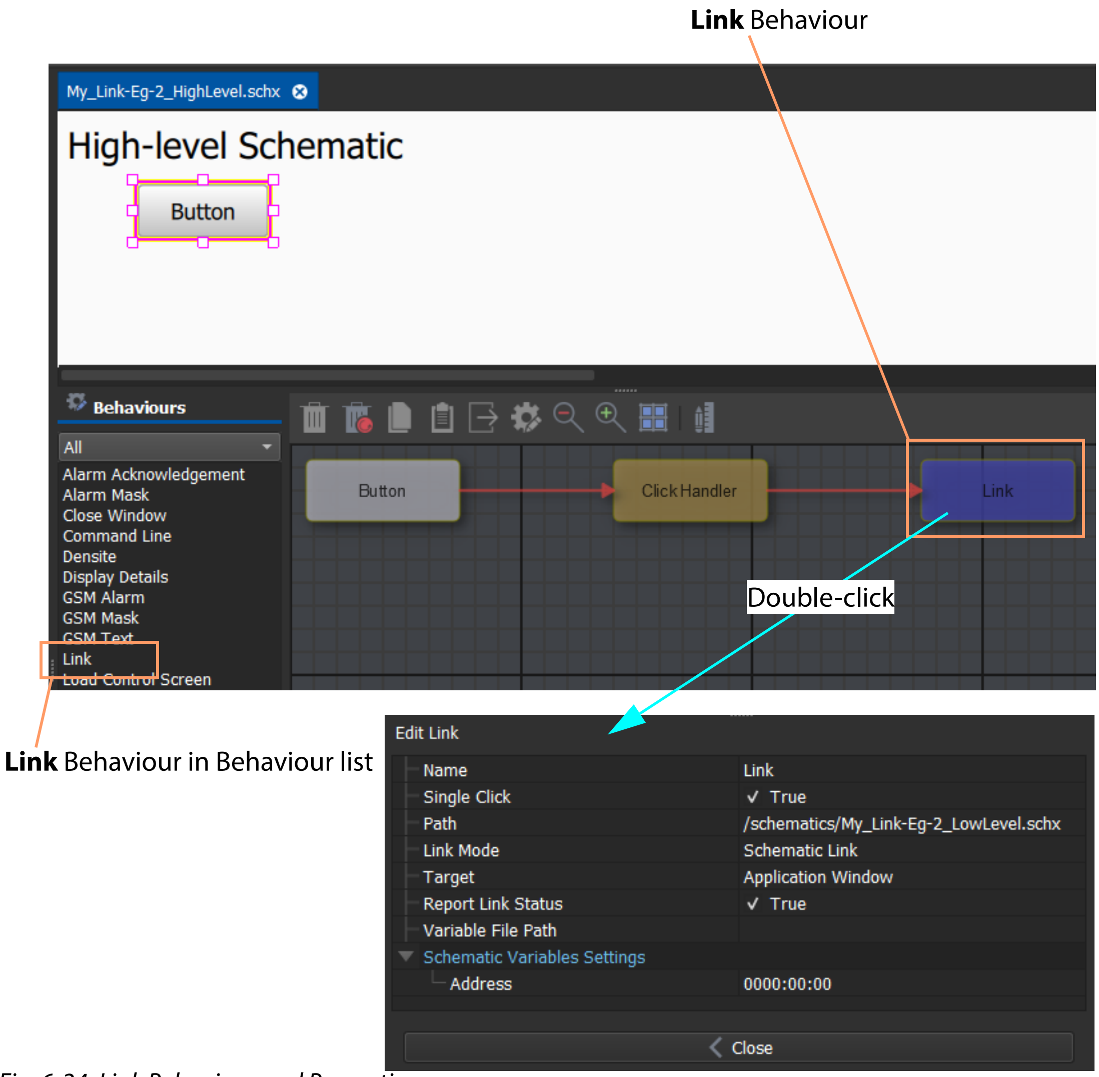
A MapView project may comprise a hierarchy of screens and each screen will have some **Schematic State** value computed from the screen's constituent 'state' values. For monitoring applications, it is essential to see the state of a lower-level screen on a higher-level button that links to it. This hierarchy of states is called **Link State**. The **Link State** of an Orbit MapView project needs to be continually calculated to ensure that any displayed screen shows the current 'state' of any linked-to, lower-level screen(s).

MapView Service and Link State

The **Link State** for an Orbit MapView project is calculated by the Orbit MapView Service: Calculation is done 'server-side', not client-side. The Orbit MapView Service is responsible for traversing the whole MapView project, calculating the **Link State** and publishing live **Link State** information for use by any subscribing Orbit client applications.

To do this, the service needs access to the same Orbit MapView project as is being run by the Orbit MapView client. This then enables button widgets etc. on all project screens to have their full Link State automatically kept up to date when a screen is opened by a user in Orbit MapView.

Link Behaviour and Properties



Link Behaviour

Link Behaviour in Behaviour list

Double-click

My_Link-Eg-2_HighLevel.schx

High-level Schematic

Behaviours

All

- Alarm Acknowledgement
- Alarm Mask
- Close Window
- Command Line
- Densite
- Display Details
- GSM Alarm
- GSM Mask
- GSM Text
- Link
- Load Control Screen

Edit Link

Name	Link
Single Click	✓ True
Path	/schematics/My_Link-Eg-2_LowLevel.schx
Link Mode	Schematic Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	
Schematic Variables Settings	
Address	0000:00:00

Fig. 6-24: Link Behaviour and Properties

Table 6-11 describes the **Link** Behaviour properties.

Table 6-11: Link Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
Single Click	Check box. To activate the link: • Select check box for single-click. • Deselect check box for double-click.
Path	Set the screen to be linked to, including file path. **Note:** The file can be any file that Orbit can open, or it can be a URL, such as http://grassvalley.com.

Table 6-11: Link Behaviour Properties (continued)

Property	Description
Link Mode	Select type of link: • **Schematic link**. • **Back button** link - Navigate the user backwards through the history of opened screens. • **Forward button** link - Navigate the user forwards through the history of opened screens.
Target	Select the target window where the linked-to file should be opened. Options: • **Application window -** Targets the main tab view. • **Monitor 1 to N -** Undocked on one of the monitors of the Orbit client computer. • **Popup -** Targets a popup window, if any are defined in Orbit. (To define popup windows, see 'Tools > Options > Pop-ups' from the main menu.) **Note:** For more information, refer to the 'Orbit - Introduction' user manual in the 'Orbit Main Menu Bar' chapter and 'Tools > Options - Pop-ups' sub-section.
Report Link Status	Check box. (Normally selected.) • Select to contribute to the overall 'state' of the screen. • Deselect to configure the **Link** Behaviour to *not* contribute to the overall schematic 'state'. Deselected example: A tool bar on a screen, or a banner, with one or more link. A screen's 'Home' button, when a 'Home' button is on a low-level screen, would be designed to link to the top-level screen and the button should show the schematic state of the linked-to screen (in this case the top-level screen). However, the 'Home' button's state should not to contribute to the low-level screen's state. Deselecting this option for the **Link** Behaviour associated with such a 'Home' button will exclude it from the calculation of the overall state of the low-level screen.
Variable File Path	File path to a text file containing a list of variables to be used when loading the linked-to screen. This is useful for re-using one screen several times using different variable values. (See Example - Screen Link States and Screen Re-Use with Variable Files, on page 243, for an example using a variable file.)
Schematic Variables Settings	A list of variables on the linked-to screen with specific settings to be used with the link. The list is obtained from the screen and any specified variable file. The list may be edited. When the link is used, values of the specified variables are used and may override those on the screen or in a variable file. This allows a single, generic linked-to screen to be flexibly reused for many different variable values.

Create a Link Between Two Screens

A link between two screens can be created by:

- adding a **Button** widget and a **Link** Behaviour and manually configuring them (see Link Behaviour with Button Widget Example, on page 237);

  or

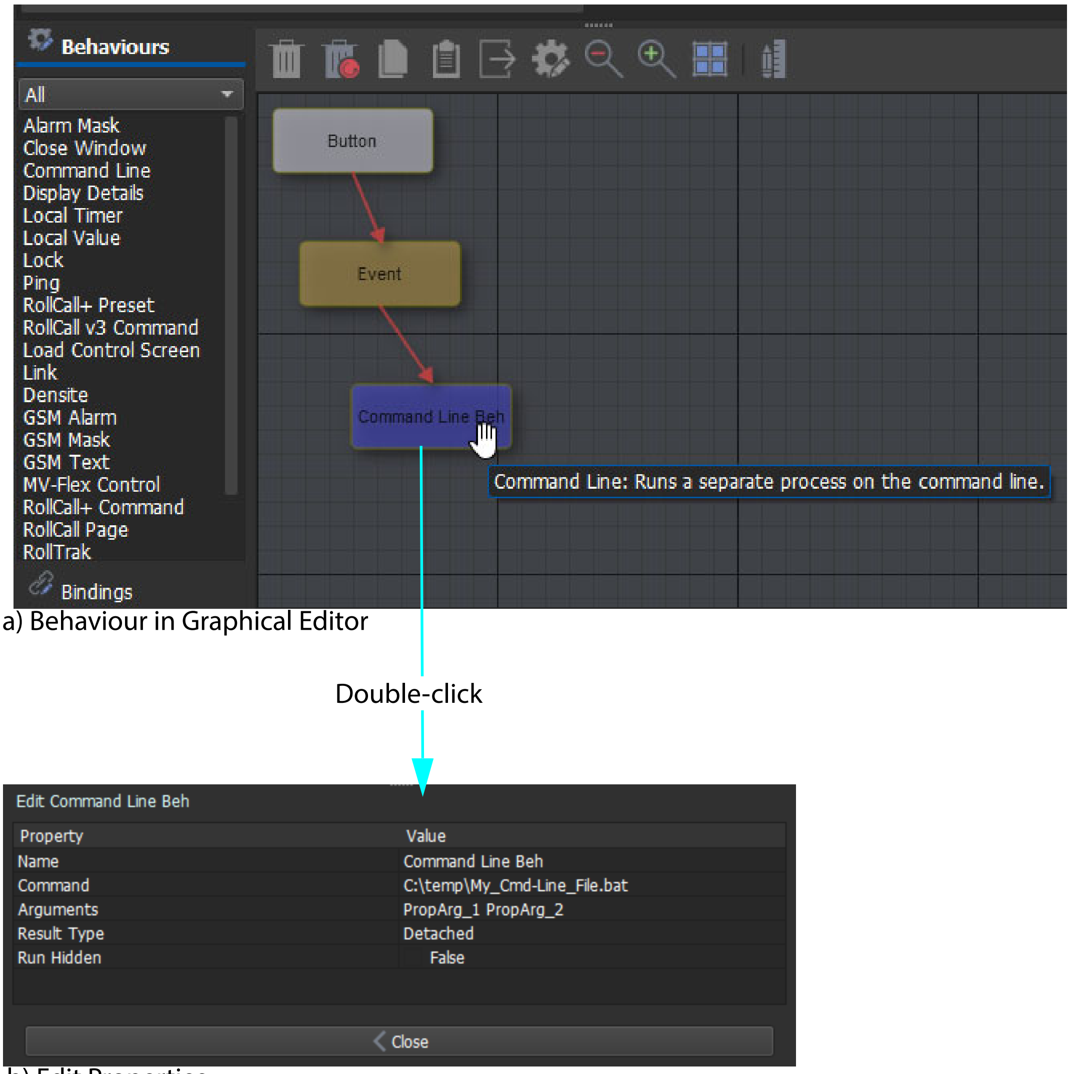
- drag-dropping a screen (from the **Project View**) onto a screen open in a schematic editor (see Drag-Drop Link Method, on page 242).

Link Behaviour Value

When a project is running, the **Link** Behaviour value is the resulting state value. (This is an integer in the range 0 to 100, inclusive - see State Values Table 2-3 on page 17.) The **Link** Behaviour value may be bound to and used to control other widget properties with a **Property** Binding.

Command Line Behaviour

A **Command Line** Behaviour runs an application or command file in a command line window on a local computer (on the Orbit client computer, or on the Orbit Services server computer if the Behaviour is in an Orbit Logic File). This enables applications or command files to be executed by a MapView screen. Initial arguments can be passed to the application from the MapView screen.



Command Line: Runs a separate process on the command line.

a) Behaviour in Graphical Editor

Double-click

Property	Value
Name	Command Line Beh
Command	C:\temp\My_Cmd-Line_File.bat
Arguments	PropArg_1 PropArg_2
Result Type	Detached
Run Hidden	False

b) Edit Properties

Fig. 6-25: Command Line Behaviour:
a) in 'Behaviour and Binding' Graphical Editor.
b) Edit Properties.

Table 6-12: Command Line Behaviour Properties

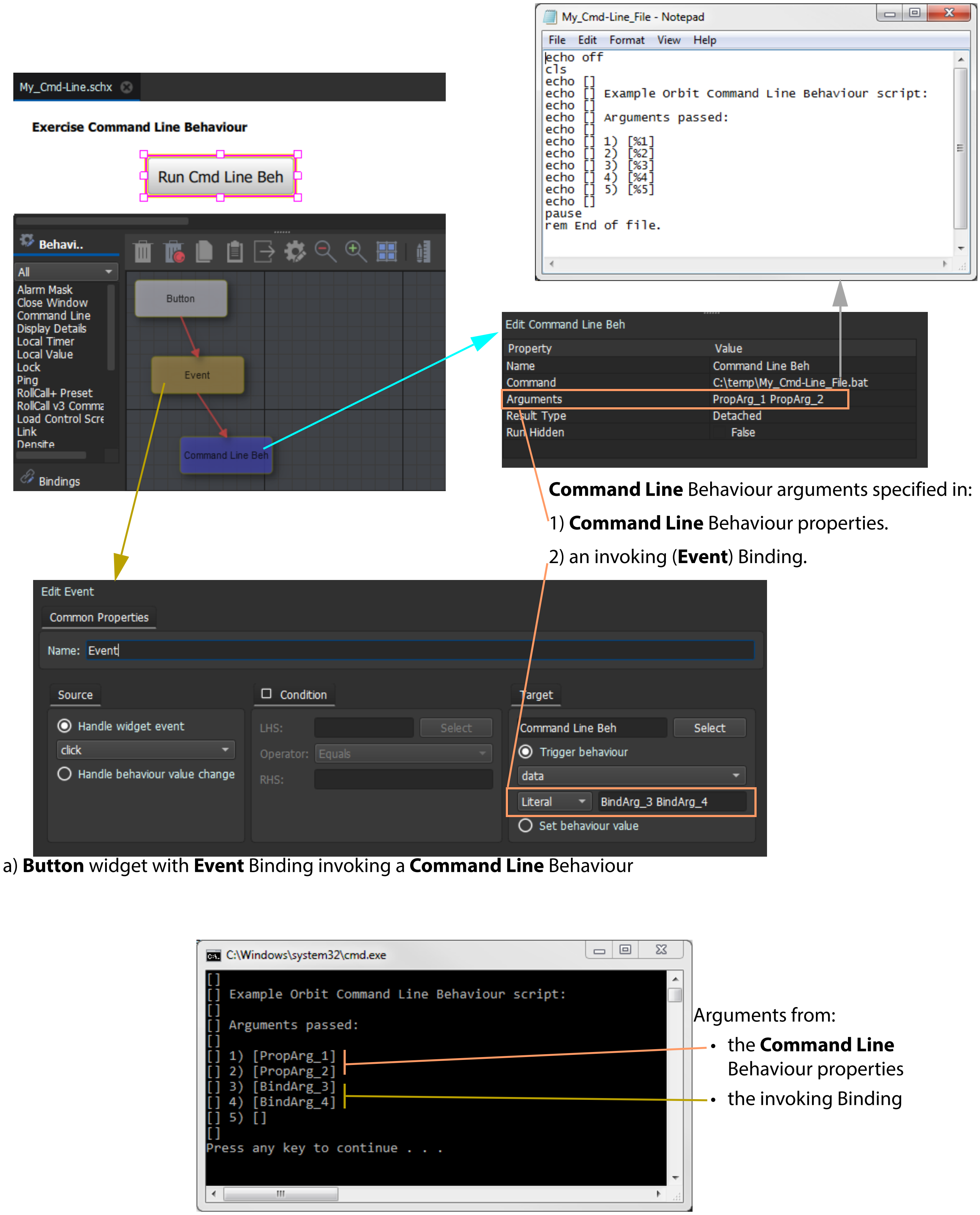
Property	Description
Name	Enter a name to identify the Behaviour.
Command	Full path to the application or command file to be executed. For example, C:\temp\My_Cmd_File.bat
Arguments	Enter fixed arguments to be passed to the executed application/command file. Use a comma-separated and/or space-separated list. For example, PropArg_1 PropArg_2
Result Type	Drop down box. Options: • **Exit Code** - Orbit MapView waits for the exit code of the application that is run. • **Standard Output** - Orbit MapView waits for and uses the 'standard output' from the application that is run. • **Detached** - Orbit MapView does not wait for any return value(s) from the application that is run.
Run Hidden	Check box. • Select (True) to hide the command line window while running. • Deselect (False) to show the command line window while running.

Arguments

Arguments specified in the properties will be passed to the application/command file when the **Command Line** Behaviour is run.

If the Behaviour is invoked by a Binding (for example an **Event** Binding) then further arguments can be defined in the Binding which are also passed to the **Command Line** Behaviour when it is executed. See Figure 6-26.

> Note: Arguments specified in an invoking Binding can be fixed, literal values, or values from other Behaviours.



```
echo off
cls
echo []
echo []  Example Orbit Command Line Behaviour script:
echo []
echo []  Arguments passed:
echo []
echo []  1)  [%1]
echo []  2)  [%2]
echo []  3)  [%3]
echo []  4)  [%4]
echo []  5)  [%5]
echo []
pause
rem End of file.
```

Edit Command Line Beh

Property	Value
Name	Command Line Beh
Command	C:\temp\My_Cmd-Line_File.bat
Arguments	PropArg_1 PropArg_2
Result Type	Detached
Run Hidden	False

Command Line Behaviour arguments specified in:

1) **Command Line** Behaviour properties.

2) an invoking (**Event**) Binding.

Edit Event

Common Properties

Name: Event

Source
● Handle widget event
click
○ Handle behaviour value change

□ Condition
LHS: Select
Operator: Equals
RHS:

Target
Command Line Beh Select
● Trigger behaviour
data
Literal BindArg_3 BindArg_4
○ Set behaviour value

a) **Button** widget with **Event** Binding invoking a **Command Line** Behaviour

```
[]
[] Example Orbit Command Line Behaviour script:
[]
[] Arguments passed:
[]
[] 1) [PropArg_1]
[] 2) [PropArg_2]
[] 3) [BindArg_3]
[] 4) [BindArg_4]
[] 5) []
[]
Press any key to continue . . .
```

Arguments from:

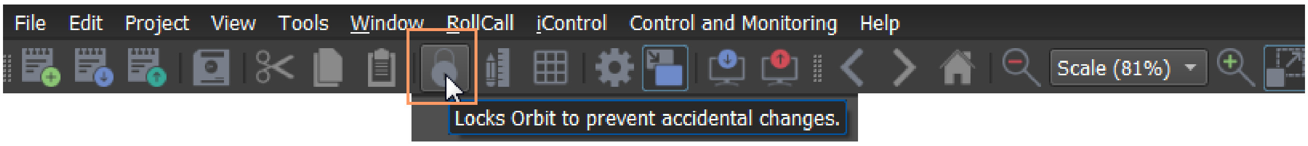
- the **Command Line** Behaviour properties
- the invoking Binding

b) Command Window Run by **Command Line** Behaviour

Fig. 6-26: Argument Passing to a Command Line Behaviour

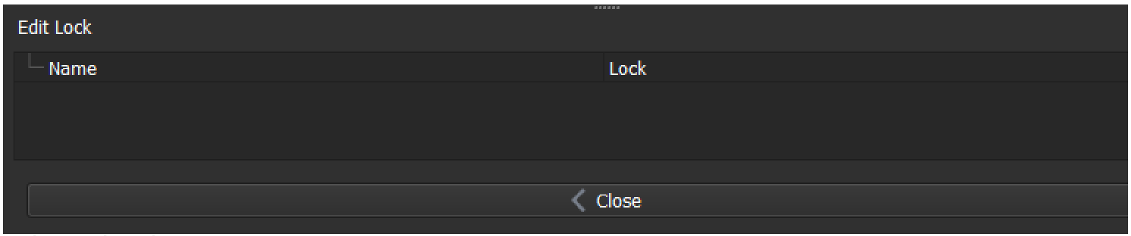
Lock Behaviour

A **Lock** Behaviour will lock or unlock the Orbit MapView screen, in a similar way to the **Lock** icon in the Orbit main tool bar.



Locks Orbit to prevent accidental changes.

a) Lock Icon in Main Tool Bar



b) Lock Behaviour Settings

Fig. 6-27: Lock Behaviour:

Table 6-13: Lock Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.

Arguments

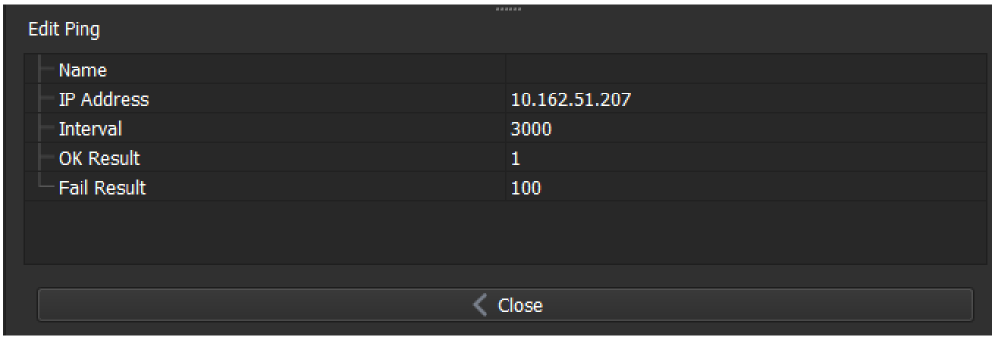
When the **Lock** Behaviour is invoked, it can have certain arguments which determine what it does (for example, 'lock' or 'unlock').

Table 6-14: Lock Behaviour Arguments

Argument	Description
lock	Lock the Orbit MapView project screen.
unlock	Unlock the Orbit MapView project screen.

Ping Behaviour

A **Ping** Behaviour will regularly ping an IP address and report an 'OK' or a 'Fail' result via the Behaviour's value.



Edit Ping

Name	
IP Address	10.162.51.207
Interval	3000
OK Result	1
Fail Result	100

Fig. 6-28: Ping Behaviour Settings

Table 6-15: Ping Behaviour Properties

Property	Description
Name	Enter a name to identify the Behaviour.
IP Address	Text box. Enter IP address.
Interval	Text box. Interval (ms) between Ping commands.
OK Result	Text box. Enter value to use if Ping is successful.
Fail Result	Text box. Enter value to use if Ping fails.

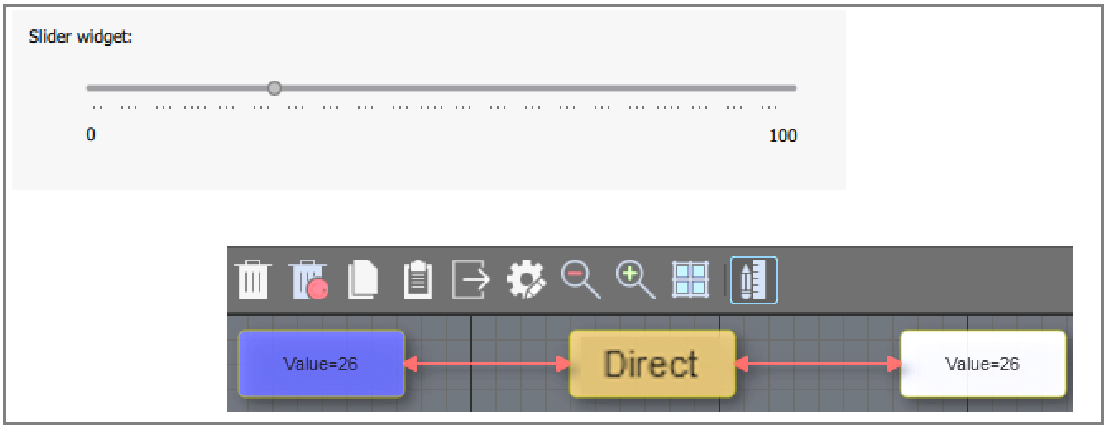
7 Examples with Bindings

Summary

Examples with Bindings

This chapter presents some MapView-specific screen examples showing the use of some Bindings.

Example - Direct Binding with a Slider Widget



This example creates a slider bound to a Behaviour using a **Direct** Binding.

This example uses:

- Widgets - **Slider**.
- Bindings - **Direct**.
- Behaviours - **Local Value**.

Create Screen

1 Open a new screen.

2 Add a **Slider** widget to the screen.

3 Select the **Slider** widget and open the 'Behaviours and Bindings' graphic editor.

(Click on the **Edit Behaviours** button, ⚙)

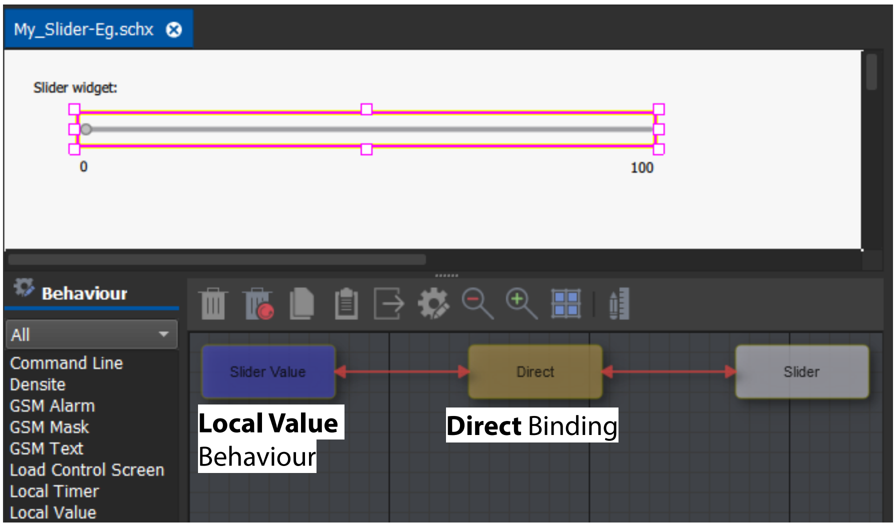
4 Click on 'Local Value' in the Behaviours list to add a **Local Value** Behaviour box to the graphic editor.

Orbit recognizes what is being done and automatically:

- names the Behaviour "Slider Value"; and
- adds a **Direct** binding with Behaviour, Binding and widget connected.

See Figure 7-1.

5 Click **Save**.

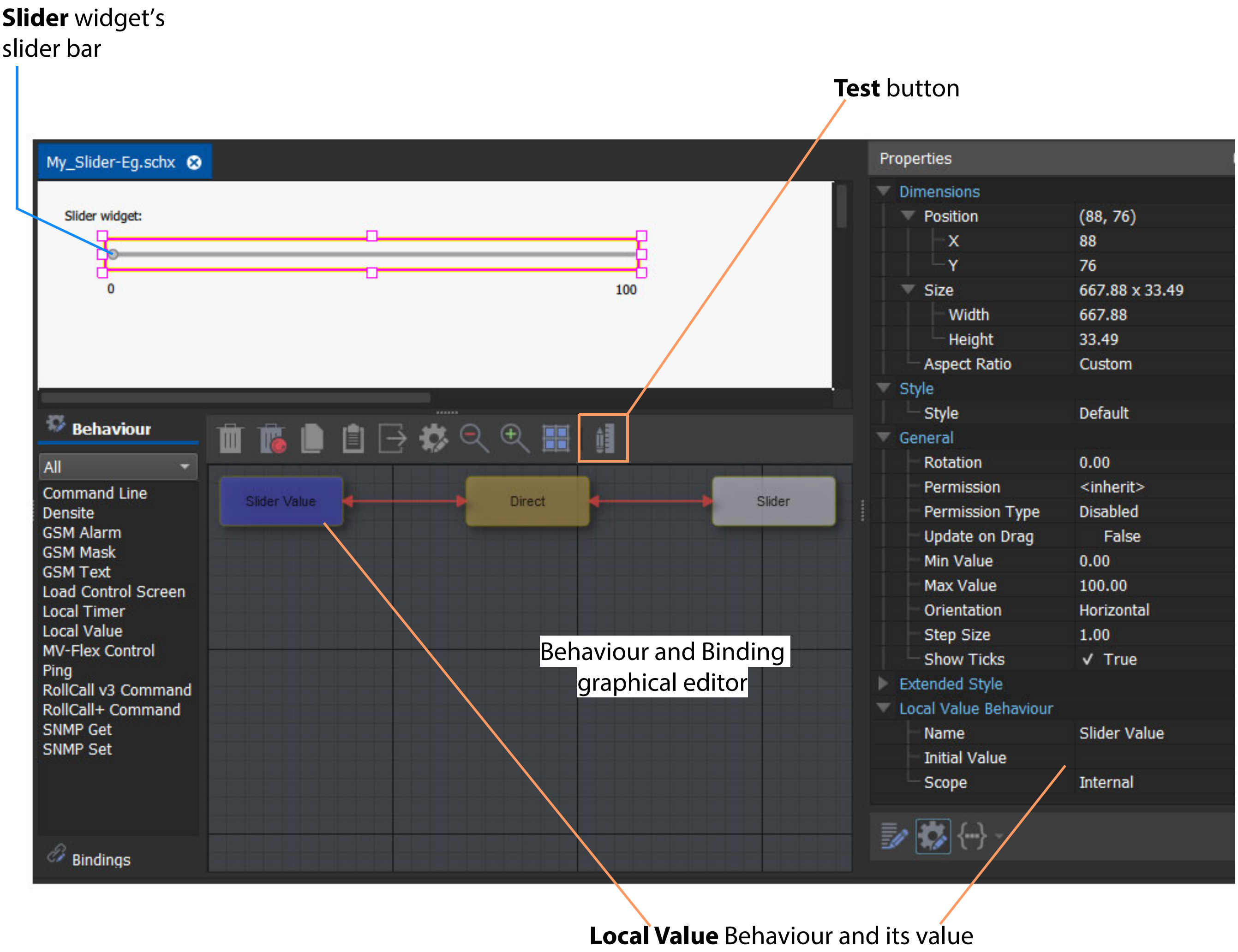


Local Value Behaviour

Direct Binding

Fig. 7-1: Slider Widget and 'Slider Value' Local Value Behaviour with a Direct Binding

Exercise the Slider Example



Properties

- Dimensions
  - Position: (88, 76)
    - X: 88
    - Y: 76
  - Size: 667.88 x 33.49
    - Width: 667.88
    - Height: 33.49
  - Aspect Ratio: Custom
- Style
  - Style: Default
- General
  - Rotation: 0.00
  - Permission: <inherit>
  - Permission Type: Disabled
  - Update on Drag: False
  - Min Value: 0.00
  - Max Value: 100.00
  - Orientation: Horizontal
  - Step Size: 1.00
  - Show Ticks: ✓ True
- Extended Style
- Local Value Behaviour
  - Name: Slider Value
  - Initial Value:
  - Scope: Internal

Fig. 7-2: Exercising Slider Widget Example

1. Enter test mode (click on the **Test** button,).
2. Exercise the slider (drag its slider bar).

   When the slider is dragged and then released, the value is shown in the 'Behaviour and Binding' graphic editor and the **Local Value** Behaviour's value is updated.

   See Figure 7-3.

Here, the **Slider** widget value is only updated when the slider is released after dragging.
However, the **Slider** widget value may be updated while the slider is dragged.
To select this mode of slider value operation:

3. End the exercising by clicking the **Test** button again.

   This enters 'Design Mode'.
4. Select the **Slider** widget and change a widget property in the **Property** box:
   - Set the 'Update on Drag' property to 'True'.

   Note: If the **Properties** box is blank, then select the screen background and then re-select the widget.
5. Click **Save**.

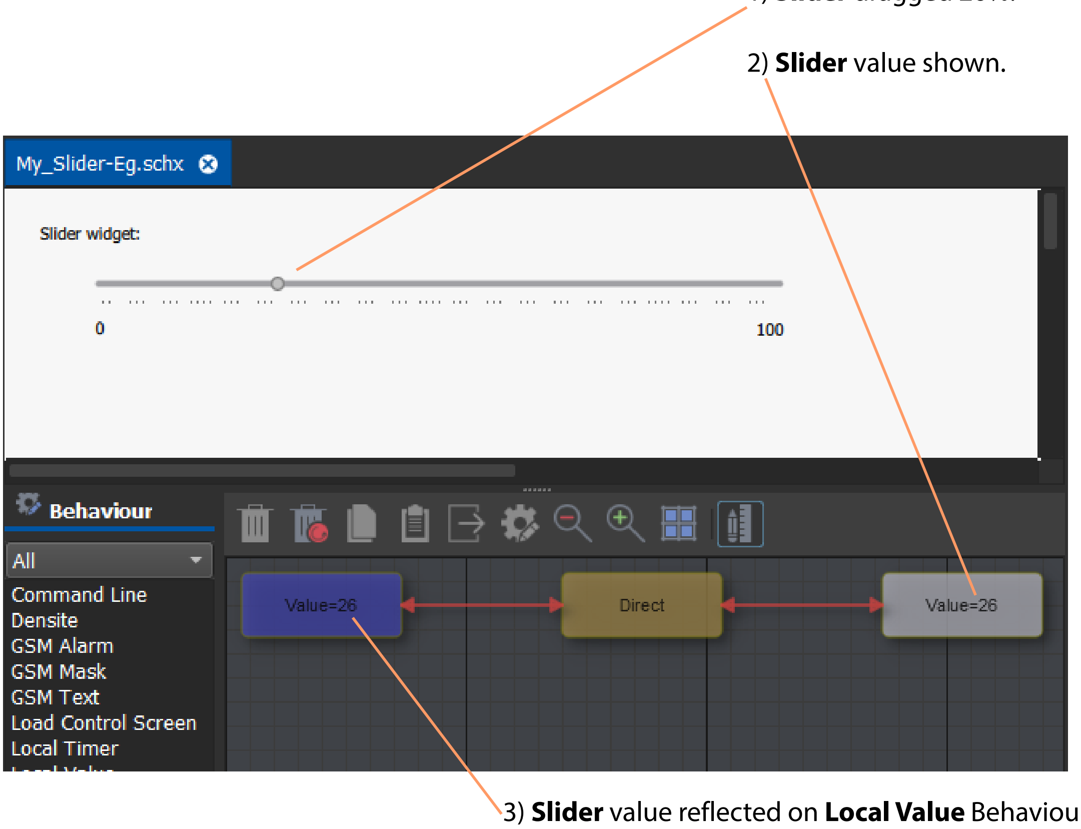
6 Re-enter test mode by clicking on the **Test** button.

7 Re-exercise the slider.

When the slider is dragged, the **Slider** widget value is shown in the 'Behaviour and Binding' graphic editor, and the **Local Value** Behaviour's value, is *continually* updated without having to release the slider.

1) **Slider** dragged 26%.

2) **Slider** value shown.



3) **Slider** value reflected on **Local Value** Behaviour

Fig. 7-3: Exercising Slider Widget and 'Slider Value' Local Value Behaviour with a Direct Binding

8 To end the exercising, click the **Test** button.

This enters 'Design Mode'.

Add a Further Slider

A second slider may be added and connected to the first:

In 'Design Mode':

1 Add a second **Slider** widget to the screen.

2 Set its 'Update on Drag' property to 'True'.

3 Select the new slider and add a **Direct** Binding.

4 Connect the new **Direct** Binding to the **Local Value** Behaviour that is connected to the first Slider. (Select this with the 'Source Behaviour 'button on the Binding's edit screen.)

5 Click **Save**.

See Figure 7-4.

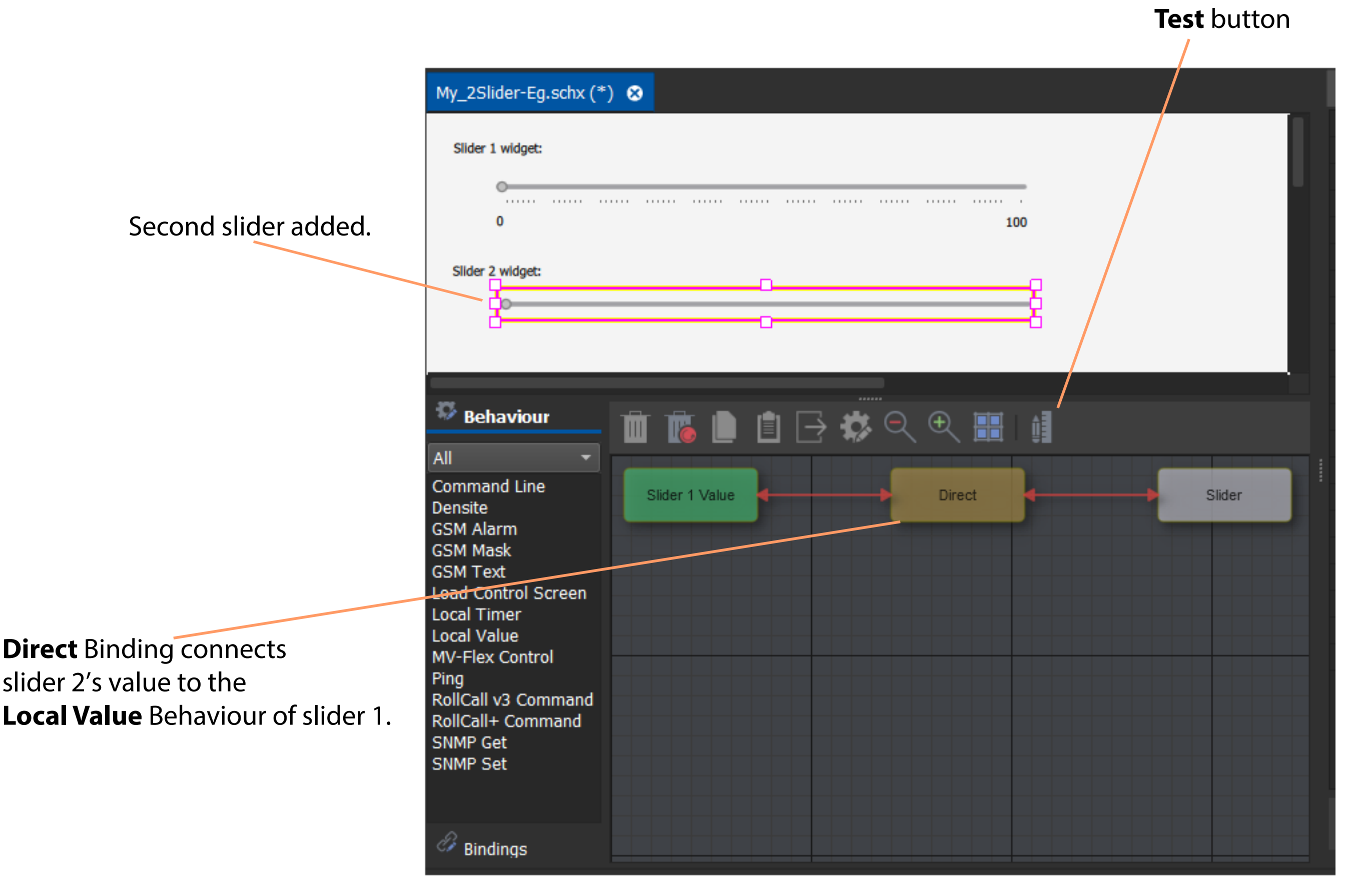


Fig. 7-4: Second Slider Widget Added

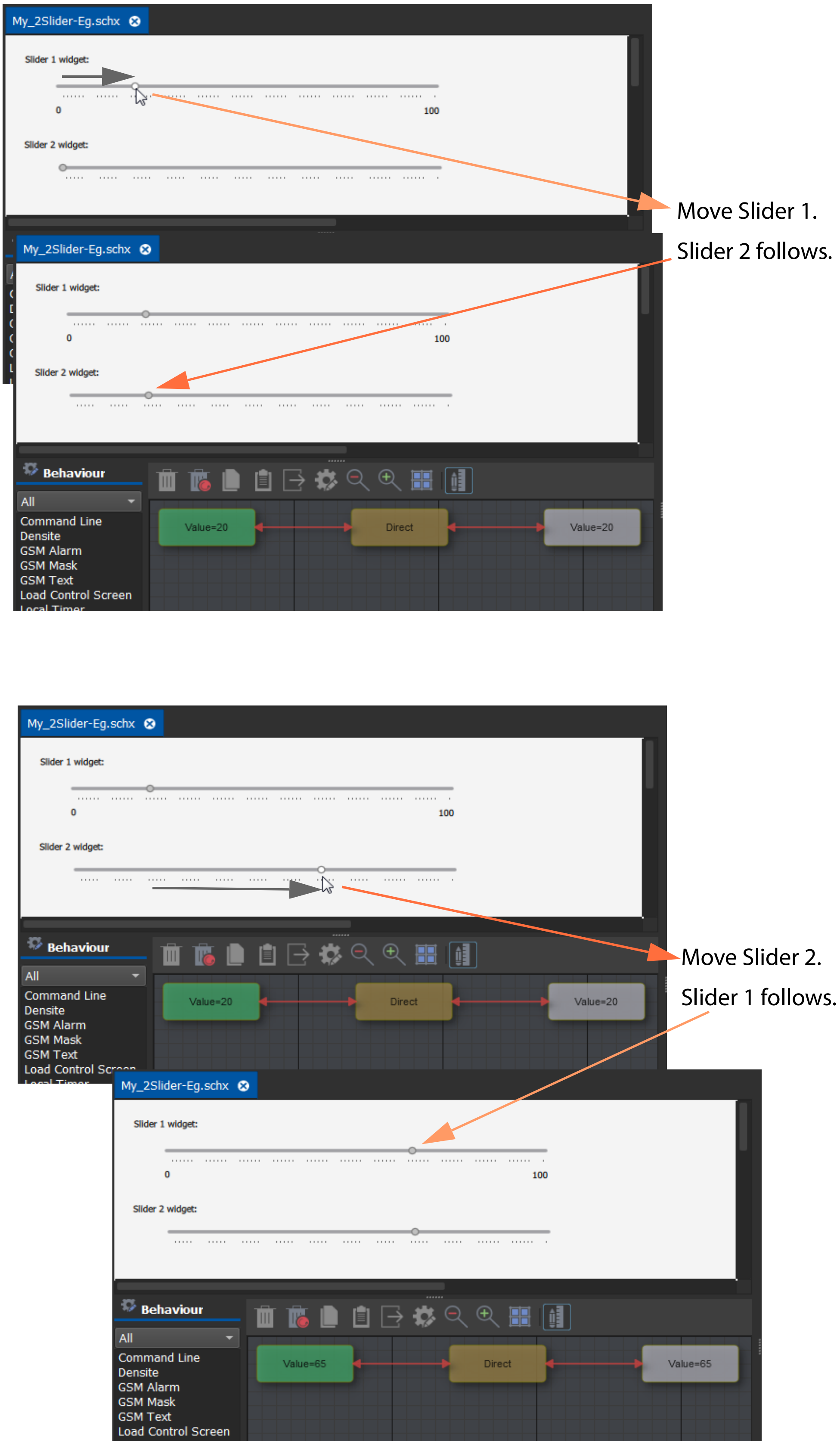
Now exercise these connected sliders:

6 Click the **Test** button to exercise the two-slider screen.

The sliders are now synchronized: Move one slider and the second also changes position, and vice versa. See Figure 7-5 on page 188.

7 To end the exercising, click the **Test** button.

This enters 'Design Mode'.



Move Slider 1.

Slider 2 follows.

Move Slider 2.

Slider 1 follows.

Fig. 7-5: Exercising Two Sliders

Example - Property Binding and Tally Lamp Widget

This example creates a slider to control a tally lamp color using a **Property** Binding.

The example uses:

- Widgets - **Tally Lamp**, **Slider**.
- Bindings - **Property**, **Direct**.
- Behaviours - **Local Value**.

Create Screen

1 Open a new screen.

2 Add a **Slider** widget to the screen.

3 Select the **Slider** widget and open the 'Behaviours and Bindings' graphic editor.

(Click on the **Edit Behaviours** button, .)

4 Click on 'Local Value' in the Behaviours list to add a **Local Value** Behaviour to the graphic editor.

Orbit recognizes what is being done and automatically names the Behaviour "Slider Value" and adds a **Direct** binding with Behaviour, Binding and widget connected. See Figure 7-6.

5 Click **Save**.

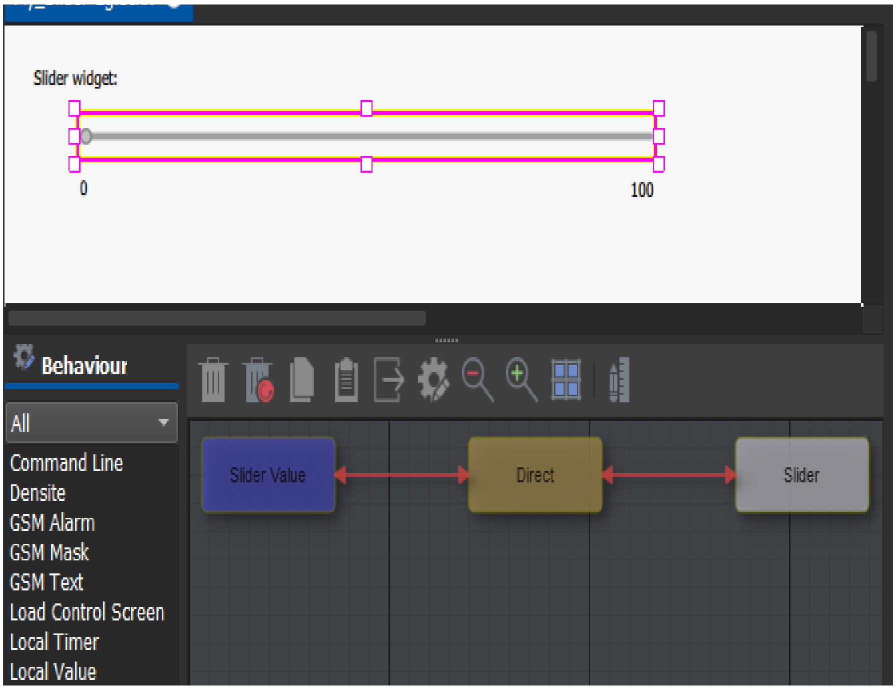


Fig. 7-6: Slider Widget and 'Slider Value' Local Value Behaviour with a Direct Binding

6 Add a **Label** widget to the screen, select it and open the 'Behaviours and Bindings' graphic editor.

7 Add a **Direct** Binding.

8 Double-click on the **Direct** Binding to edit its properties.

9 Set the source Behaviour to be the **Local Value** Behaviour which holds the **Slider** widget value.

10 Click **Close**.

11 Click **Save**.

The label will now usefully display the slider value on screen when the screen is run.

12 Add a **Tally Lamp** widget to the screen. See Figure 7-7.

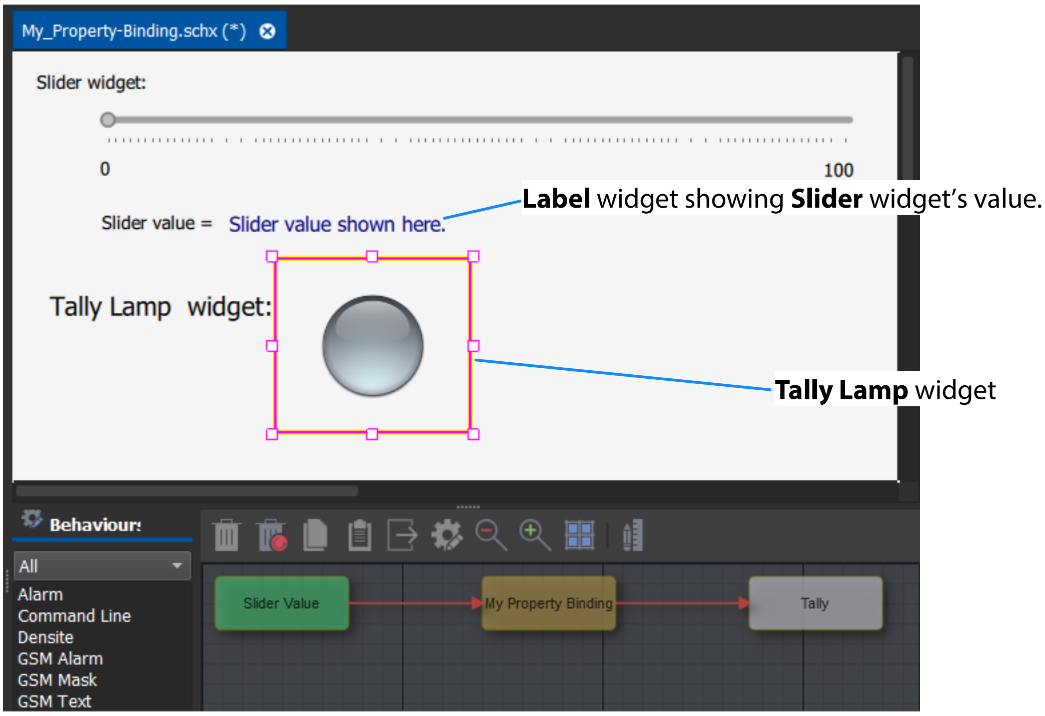


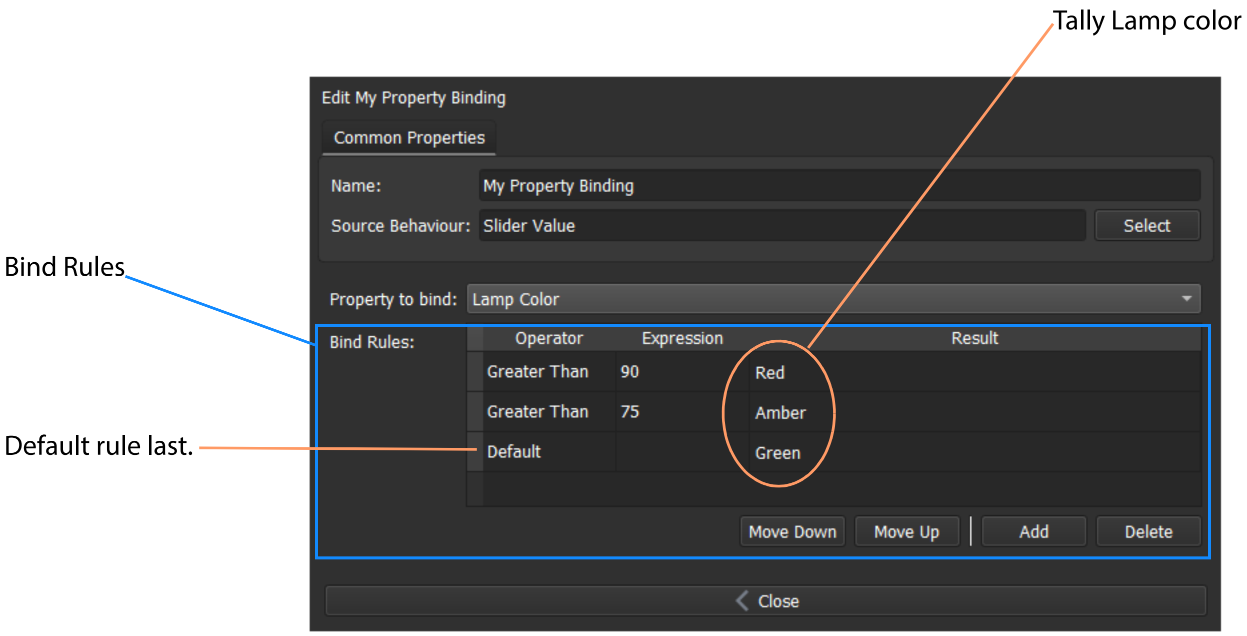
Fig. 7-7: Tally Lamp Added

13 Select the **Tally Lamp** widget and open the 'Behaviours and Bindings' graphic editor.

14 Add a **Property** Binding and double-click on it to edit its properties.

15 Set:

- 'Source Behaviour' to be the **Local Value** Behaviour holding the slider value.
- Set the 'Property to Bind' to be 'Tally Lamp Color'.
- Set up 'Bind Rules' as shown in Figure 7-8.



Tally Lamp color

Bind Rules

Default rule last.

Edit My Property Binding

Common Properties

Name: My Property Binding

Source Behaviour: Slider Value

Property to bind: Lamp Color

Bind Rules:

Operator	Expression	Result
Greater Than	90	Red
Greater Than	75	Amber
Default		Green

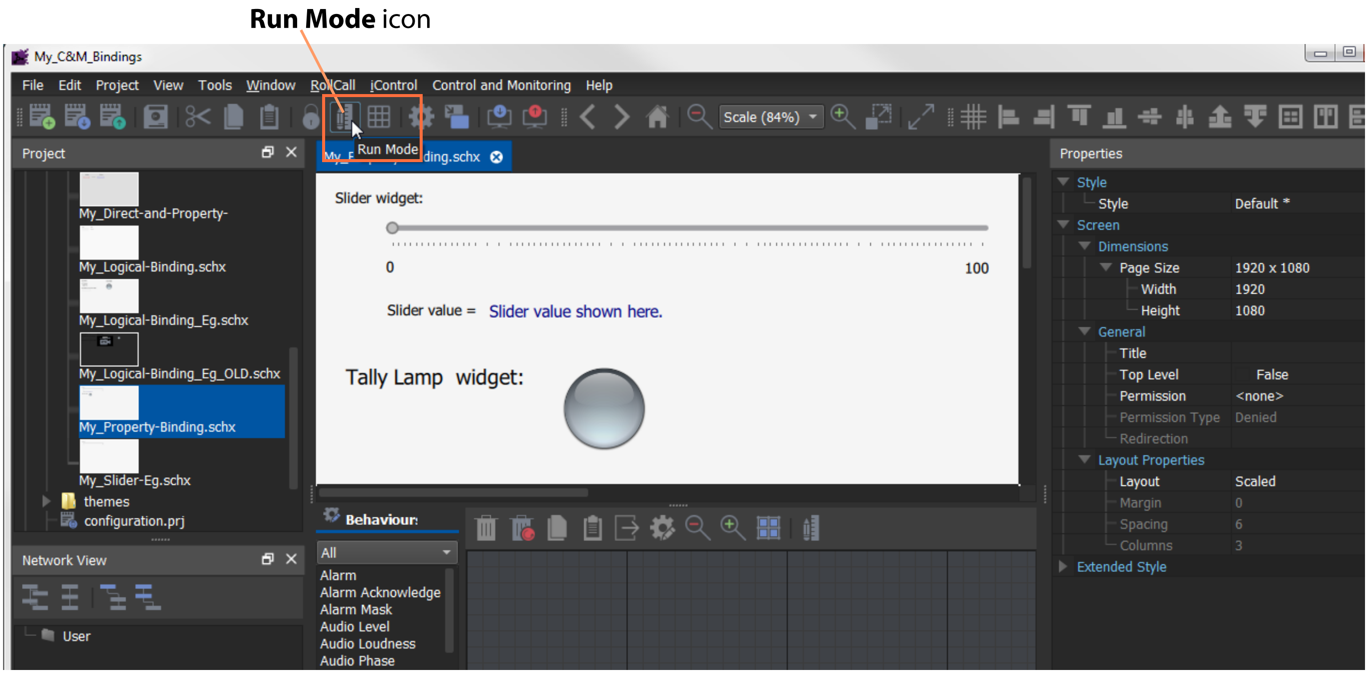
Fig. 7-8: Setting the Property Binding's Properties

16 Click **Close**. Click **Save**. Click **Save Project**.

The slider value is now bound to the tally lamp color and the slider value range (0-100) is mapped to the lamp color values for 'Red', 'Green', and 'Yellow' (1, 2, and 3 respectively).

Exercise the Example

1 To enter 'Run Mode', click the **Run Mode** icon in the main tool bar, see Figure 7-9.



Run Mode icon

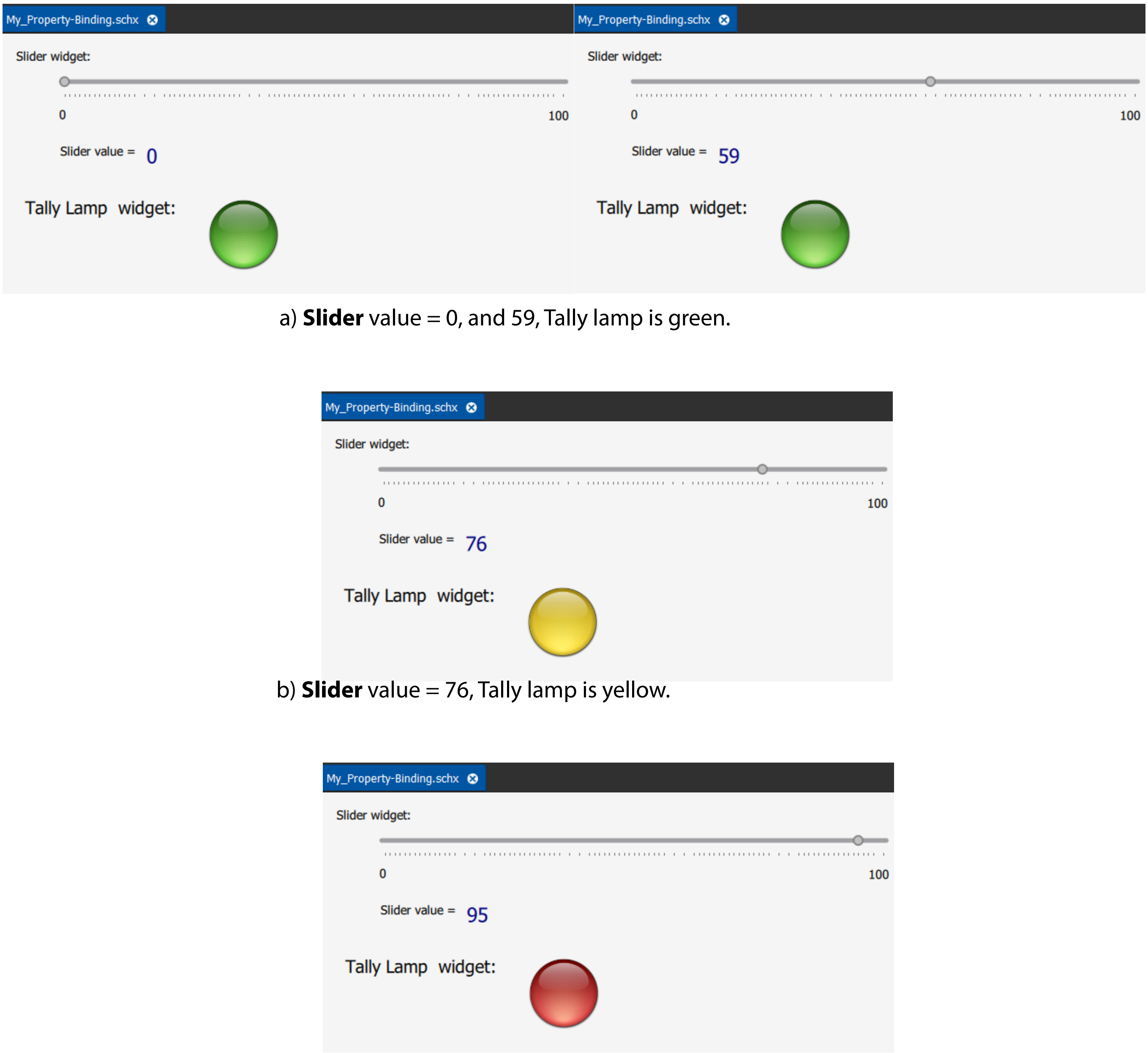
Property	Value
Style	
Style	Default *
Screen	
Dimensions	
Page Size	1920 x 1080
Width	1920
Height	1080
General	
Title	
Top Level	False
Permission	<none>
Permission Type	Denied
Redirection	
Layout Properties	
Layout	Scaled
Margin	0
Spacing	6
Columns	3
Extended Style	

Fig. 7-9: Enter Run Mode

2 Exercise the slider.

When the slider is dragged and released, the lamp color changes according to the Bind rules set up.

See Figure 7-10.



My_Property-Binding.schx

Slider widget:

Slider value = 0

Tally Lamp widget:

My_Property-Binding.schx

Slider widget:

Slider value = 59

Tally Lamp widget:

a) **Slider** value = 0, and 59, Tally lamp is green.

My_Property-Binding.schx

Slider widget:

Slider value = 76

Tally Lamp widget:

b) **Slider** value = 76, Tally lamp is yellow.

My_Property-Binding.schx

Slider widget:

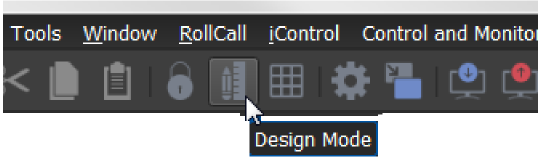
Slider value = 95

Tally Lamp widget:

c) **Slider** value = 95, Tally lamp is red.

Fig. 7-10: Exercising Slider and Tally Lamp

3 To end the exercising, click the **Design Mode** icon in the main tool bar.



This enters ‘Design Mode’.

Example - String Op Binding and Math Binding

Gbits/sec ⇨ GBits/sec

Mbits/sec ⇨ GBits/sec

kbits/sec ⇨ GBits/sec

This example processes log field messages from some device issuing log messages in a RollCall system. The example shows how to use a log field value and convert it into a 'Gbits/sec' figure, which may then be used and further summed with other log field figures, as required.

The example uses:

- Widgets - none.
- Bindings - **String Op**, **Math**, **Mapped**.
- Behaviours - **Log Field**, **Local Value**.

The monitored log messages indicate a data rate value (for example, data rate through a network switch). The log field value will change from being Gbits, Mbits, etc. as the data rate changes.

For example, a log field may contain:
'10.4 Gbits/sec'; '1000 Mbits/sec'; '1250 Kbits/sec'; or '900 bits/sec'.

To convert to Gbits/sec, we need to extract the numeric part of the log field message and the textual part. And then derive a divisor value from the textual part (see Table 7-1) and use this on the numeric value to yield a normalized Gbits/sec result.

Table 7-1: Divisor

Textual Part	Divisor Value
Gbit/sec	1.0
Mbit/sec	1000.0
Kbit/sec	1000000.0
bit/sec	1000000000.0

We can do all this purely in Orbit, with no special coding. The steps involved are described below:

1. Step 1: Get Data from Device, on page 194
2. Step 2: Extract the Textual Information (String Op Binding), on page 195.
3. Step 3: Determine a Divisor (Mapped Binding), on page 196
4. Step 4: Convert the Data Rate Value to Gbits/sec (Math Binding), on page 197

Step 1: Get Data from Device

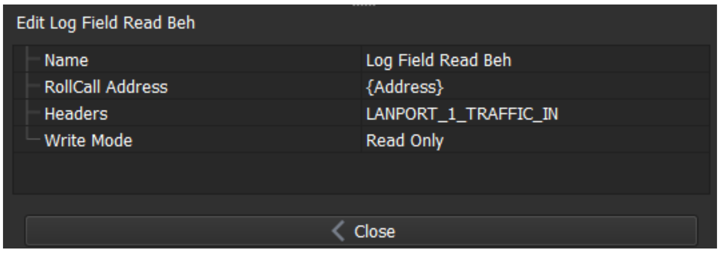
1 Open a new Orbit MapView screen.

2 Open the 'Behaviours and Bindings' graphical editor. (Click on the **Edit Behaviours** button.) And ensure no item is selected in the screen.

3 Add a **Log Field** Behaviour to the 'Behaviours and Bindings' graphical editor.

4 Double-click on the **Log Field** Behaviour and configure it:

- Enter a new, meaningful **Name**. (E.g. 'Log Field Beh')
- Enter a RollCall address. Use an Orbit screen variable for flexibility, e.g. {Address}. (Ultimately, this must be set to the RollCall address of the system device being monitored.)
- Enter a RollCall log field header name for the Behaviour to use, i.e. the field containing the data information. For example, LANPORT_1_TRAFFIC_IN for data rate information.



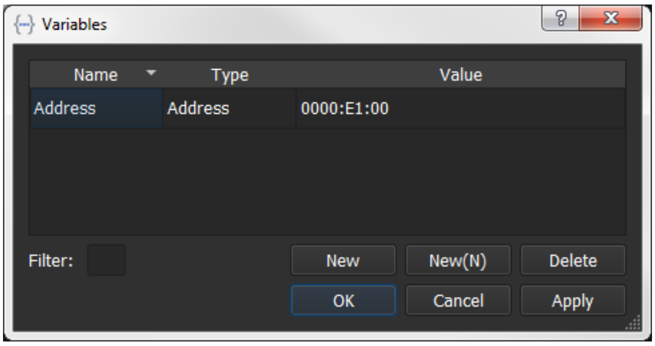
Edit Log Field Read Beh

Name	Log Field Read Beh
RollCall Address	{Address}
Headers	LANPORT_1_TRAFFIC_IN
Write Mode	Read Only

5 Click **Close**.

6 Right-click in the screen background and select 'Variables...'.

7 Set the screen variable 'Address' to the monitored device's RollCall address.
For example, '0000:E1:00'.
Click **OK**.



Variables

Name	Type	Value
Address	Address	0000:E1:00

8 Click **Save**.

The **Log Field** Behaviour value will contain the required log message text from the device at the RollCall address.

Note: The log field message text will be of the form: 10.4 Gbits/sec

Step 2: Extract the Textual Information (String Op Binding)

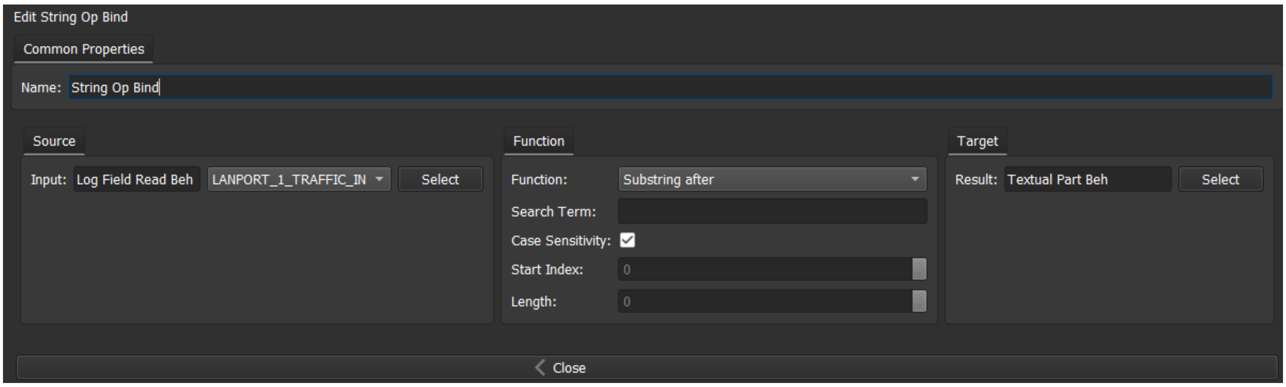
1 Add a **String Op** Binding to the screen in the ‘Behaviours and Bindings’ graphical editor.

2 Double-click on the Binding to configure it:

- Enter a new, meaningful **Name**.
- Select the (Log Field) Behaviour to use as input string source.
  And select the header to use.
- Select the ‘Substring after’ operation.
  And set ‘Search Term’ to be a space character.

  (This get the textual part of the message, after the numeric value.)

3 Place the result in a new, target **Local Value** Behaviour. (E.g. ‘Textual Part Beh’)



Edit String Op Bind

Common Properties

Name: String Op Bind

Source

Input: Log Field Read Beh | LANPORT_1_TRAFFIC_IN

Function

Function: Substring after

Search Term:

Case Sensitivity: ☑

Start Index: 0

Length: 0

Target

Result: Textual Part Beh

Fig. 7-11: String Op Binding Settings

4 Click **Close**. Click **Save**.

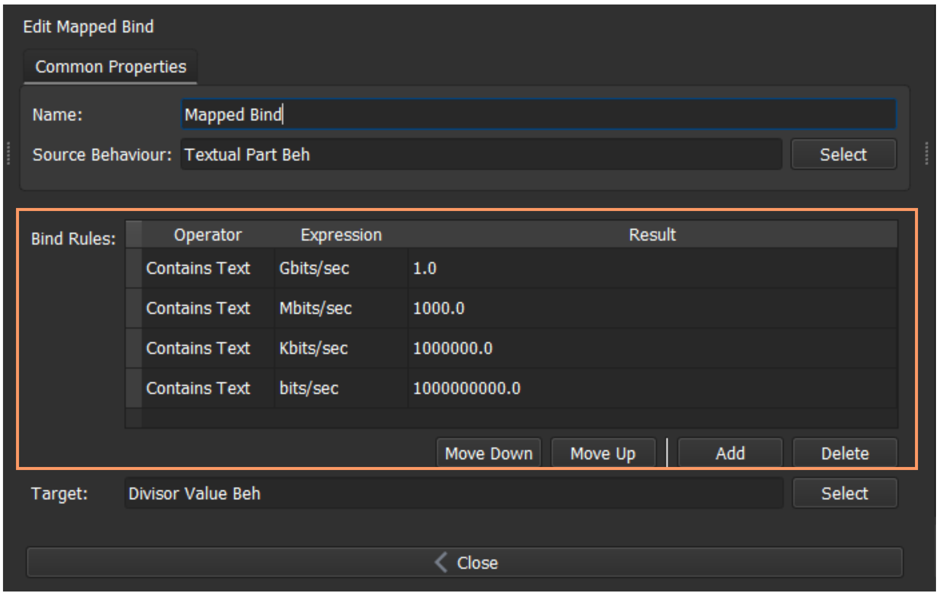
The **String Op** Binding extracts the textual part of the log message and places it into the **Local Value** Behaviour.

Step 3: Determine a Divisor (Mapped Binding)

1 Add a **Mapped** Binding to the screen in the 'Behaviours and Bindings' graphical editor.

2 Configure the Binding:

- Enter a meaningful **Name**.
- Source is the 'Textual Part Beh'.
- Set Bind Rules as shown in Figure 7-12.
- Set the target for the result of the Bind Rules to be a new, target **Local Value** Behaviour. (E.g. 'Divisor Value Beh')



Name: Mapped Bind

Source Behaviour: Textual Part Beh

Bind Rules:

Operator	Expression	Result
Contains Text	Gbits/sec	1.0
Contains Text	Mbits/sec	1000.0
Contains Text	Kbits/sec	1000000.0
Contains Text	bits/sec	1000000000.0

Target: Divisor Value Beh

Fig. 7-12: Mapped Binding Settings, Bind Rules Shown

3 Click **Close**. Click **Save**.

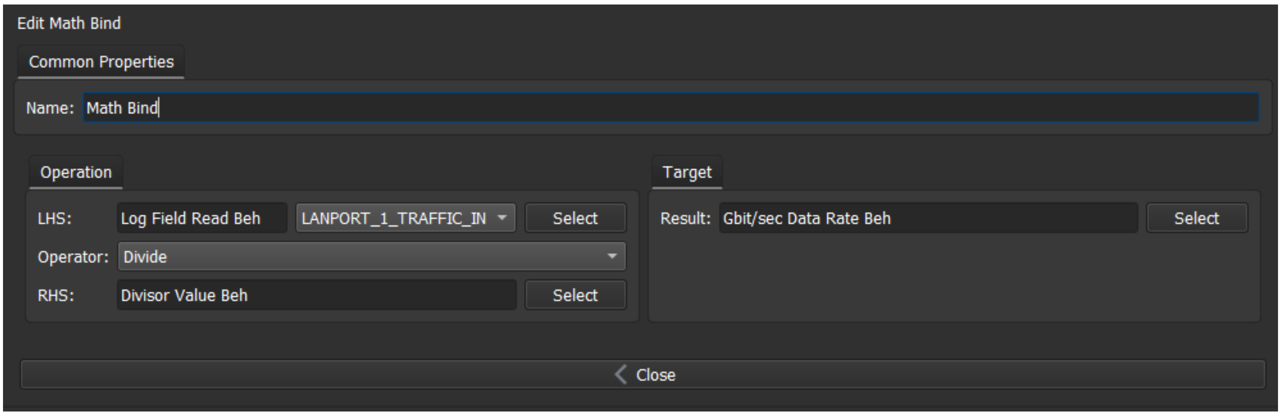
The **Mapped** Binding determines a divisor value and stores it in the **Local Value** Behaviour.

Step 4: Convert the Data Rate Value to Gbits/sec (Math Binding)

1 Add a **Math** Binding to the screen in the 'Behaviours and Bindings' graphical editor.

2 Configure:

- Select the 'Log Field Read Beh' Behaviour with its corresponding header as the LHS input.
- Select the 'Divide' operator.
- Select the 'Divisor Value Beh' Behaviour as the RHS input.
- Select a new, target **Local Value** Behaviour for the final result. ('Gbit/sec Data Rate Beh')



Edit Math Bind

Common Properties

Name: Math Bind

Operation

LHS: Log Field Read Beh LANPORT_1_TRAFFIC_IN Select

Operator: Divide

RHS: Divisor Value Beh Select

Target

Result: Gbit/sec Data Rate Beh Select

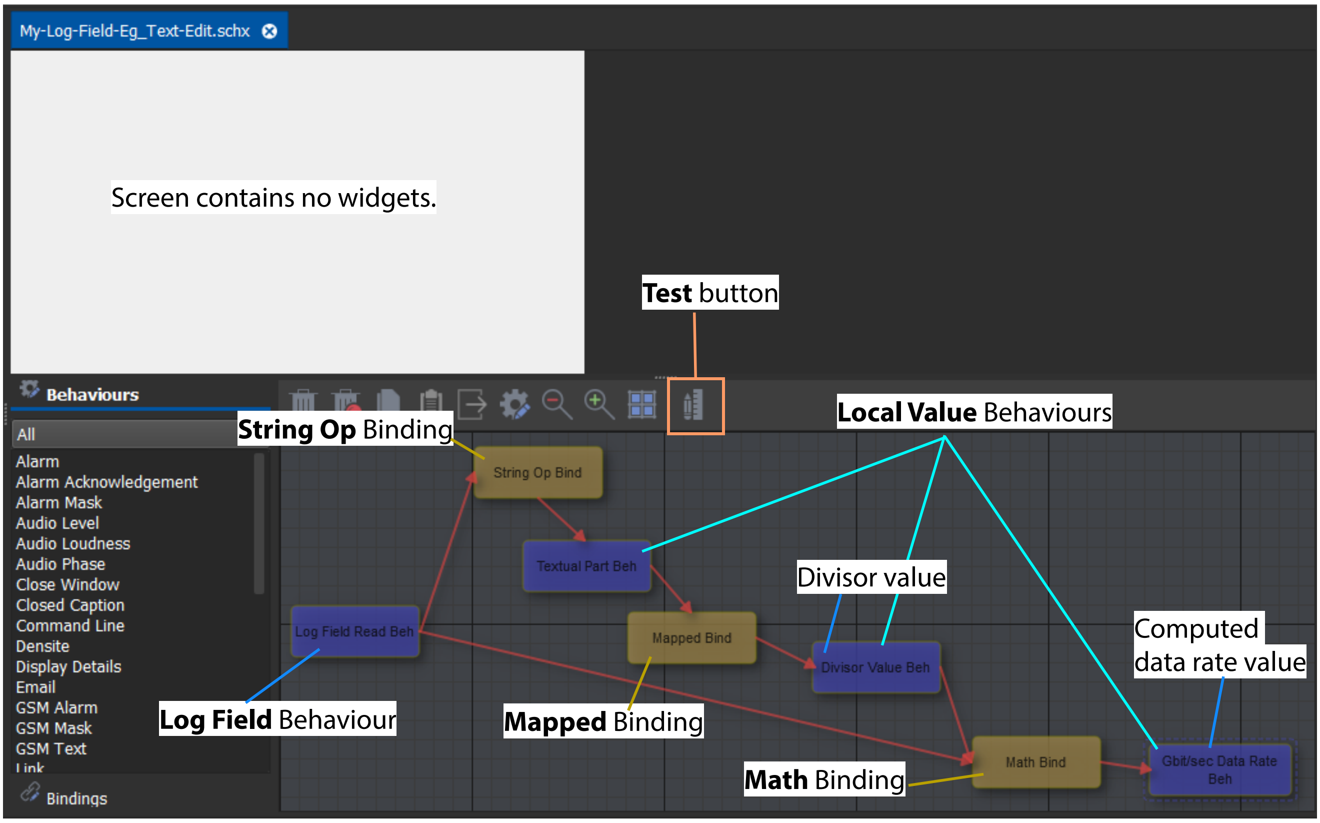
Close

Fig. 7-13: Math Binding Settings

3 Click **Close**. Click **Save**.

The **Local Value** Behaviour will contain the computed data rate result in units of 'Gbits/sec'.

The resulting screen contains no widgets, only Behaviours and Bindings. See Figure 7-14.



Screen contains no widgets.

Test button

String Op Binding

Local Value Behaviours

Divisor value

Computed data rate value

Log Field Behaviour

Mapped Binding

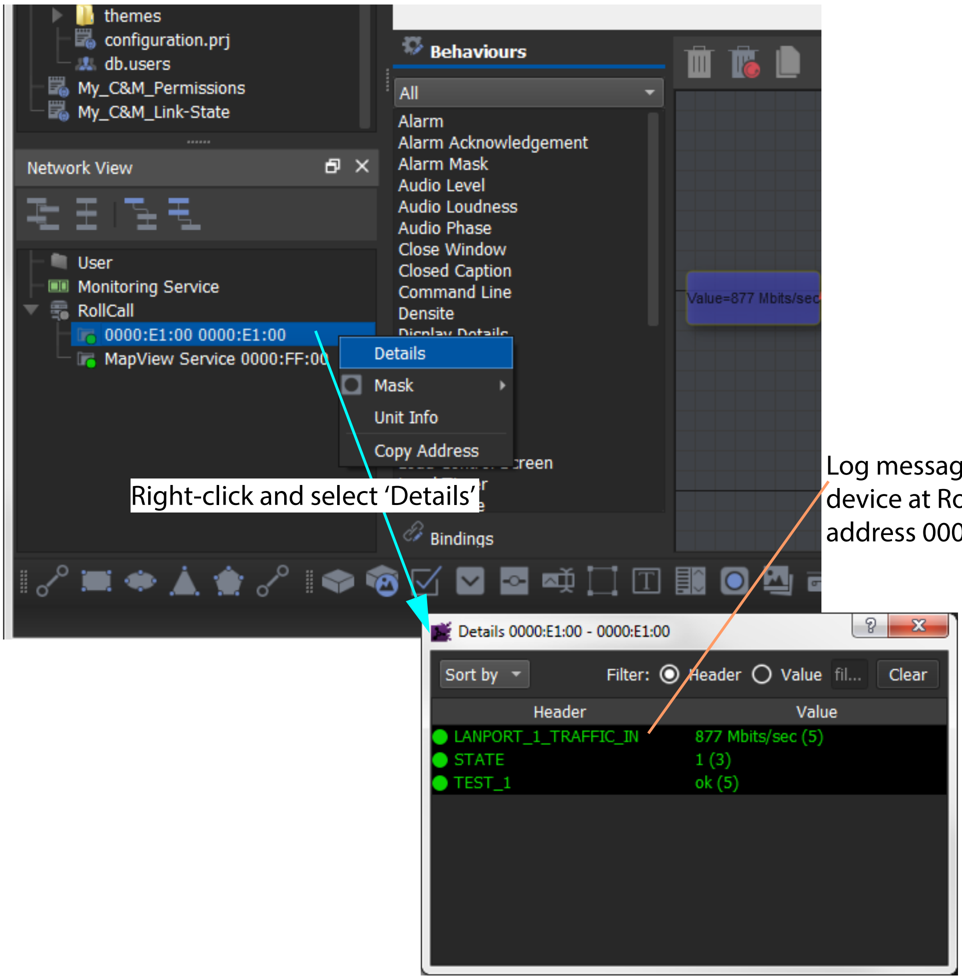
Math Binding

Fig. 7-14: Screen and Behaviours and Bindings

Exercise the Example

Test Stimulus

For the purposes of this example, it is assumed that some external RollCall device is issuing log messages that can be monitored. This forms the stimulus for exercising the example. Such a device and its log messages may be seen in the **Network View**.



Header	Value
LANPORT_1_TRAFFIC_IN	877 Mbits/sec (5)
STATE	1 (3)
TEST_1	ok (5)

Fig. 7-15: External Device in Network View

Note: If there is no suitable external device at the time of exercising the example, then a dummy test source of data may be set up:

- Replace the **Log Field** Behaviour, which contains the monitored log message, by a **Text Edit** widget.
- Connect the widget to a **Local Value** Behaviour.

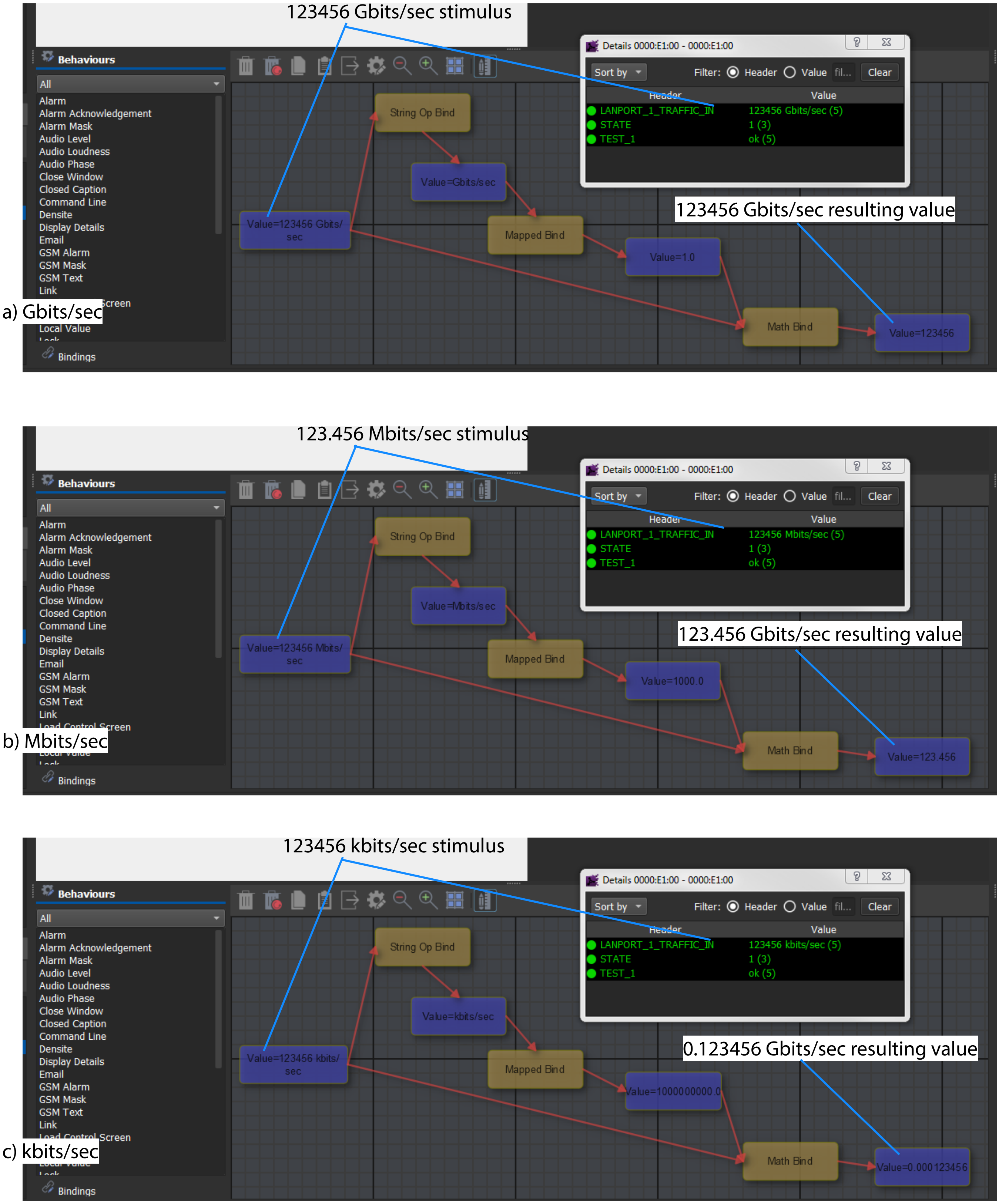
When exercising a widget-under-test, enter message text into the **Text Edit** widget. The resulting **Local Value** Behaviour value can be used as a stimulus for exercising the widget-under-test.

Exercising

To exercise the example:

1 Click the **Test** button.

The resulting exercising is shown in Figure 7-16a, b and c.



123456 Gbits/sec stimulus

123456 Gbits/sec resulting value

a) Gbits/sec

Details 0000:E1:00 - 0000:E1:00

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● LANPORT_1_TRAFFIC_IN	123456 Gbits/sec (5)
● STATE	1 (3)
● TEST_1	ok (5)

123.456 Mbits/sec stimulus

123.456 Gbits/sec resulting value

b) Mbits/sec

Details 0000:E1:00 - 0000:E1:00

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● LANPORT_1_TRAFFIC_IN	123456 Mbits/sec (5)
● STATE	1 (3)
● TEST_1	ok (5)

123456 kbits/sec stimulus

0.123456 Gbits/sec resulting value

c) kbits/sec

Details 0000:E1:00 - 0000:E1:00

Sort by ▾ Filter: ● Header ○ Value fil... Clear

Header	Value
● LANPORT_1_TRAFFIC_IN	123456 kbits/sec (5)
● STATE	1 (3)
● TEST_1	ok (5)

Fig. 7-16: Exercising: a) Gbits/sec; b) Mbits/sec; and c) kbits/sec.

As the log field string changes value to ‘Gbits/sec’, ‘Mbits/sec’ and ‘kbits/sec’ the final value will always be in ‘GBits/sec’ units. (This normalized value can then be used elsewhere in Orbit, as required.)

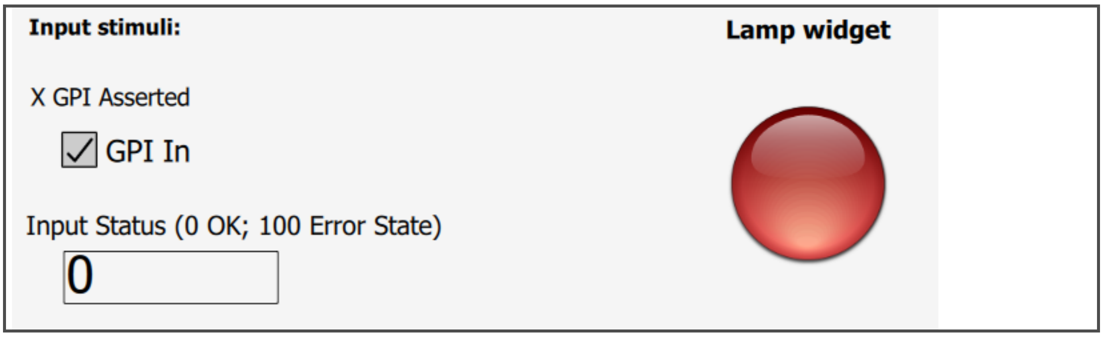
2 Click the **Test** button again to end Test mode and enter ‘Design Mode’.

Note: **Component with no widgets:**
You can put this functionality inside a component with no widgets and then re-use it on many screens.

With such a component on a screen:

- create a **Local Value** Behaviour at the component instance on the screen; and
- use a **Mapped** Binding with a single default rule to copy the component value into the screen.

Example - Logical Binding and *Simulated* GPI or Alarm State



Input stimuli:

X GPI Asserted

☑ GPI In

Input Status (0 OK; 100 Error State)

0

The example below *simulates* an Alarm state value (from some source) in order to provide a stimulus for exercising the **Logical Binding**.

This example uses:

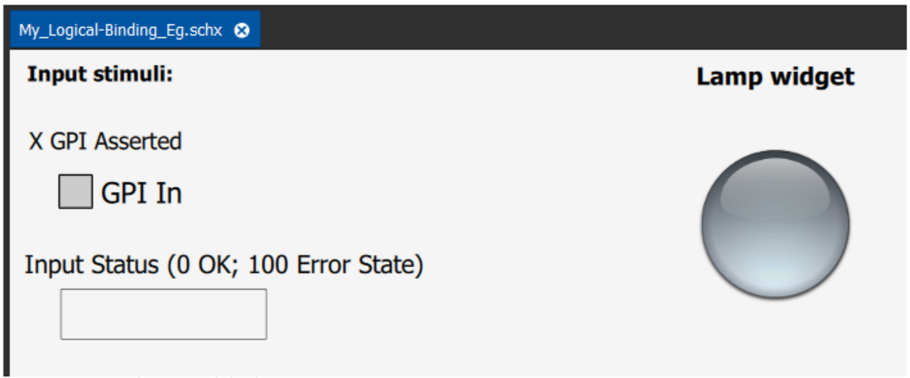
- Widgets - **Lamp**, **Check Box, Text Edit**.
- Bindings - **Logical**.
- Behaviours - **Local Value**.

Note: Orbit MapView can show an on-screen alarm triggered by one or more external device GPIs or by some device video input error. For instance, an MV-8 series multiviewer's GPI port values or video input status are exposed as RollCall commands to Orbit and Orbit Behaviours are used to get information.
Typically, these Behaviours return values of '0' or '1', or some state value (in the range 0 to 100, where 100 signifies an error).
(This example simulates an alarm source for the purposes of exercising the **Logical** Binding.)

Build the Example

This example describes how to illuminate a lamp from Behaviour values that can be 0 or 1, or 0 to 100.

1 Open a screen schematic (C&M project).
2 Add a **Lamp** widget.
3 Add a **Check Box** widget.
4 Add a **Text Edit** widget.
See Figure 7-17.



My_Logical-Binding_Eg.schx

Input stimuli:

X GPI Asserted

□ GPI In

Input Status (0 OK; 100 Error State)

Lamp widget

Fig. 7-17: Widgets Added

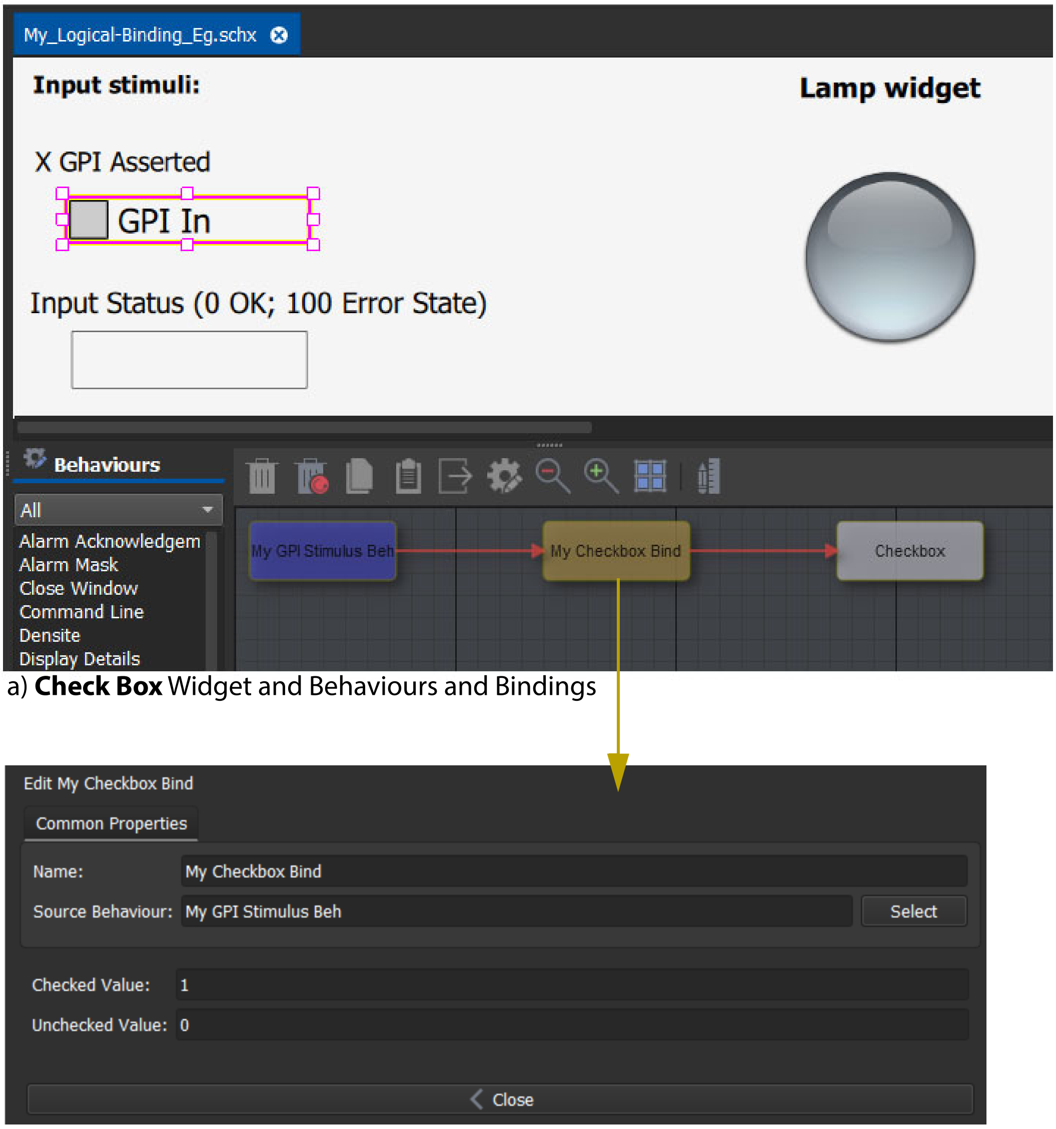
The following are carried out with the ‘Behaviours and Bindings’ graphical editor:

5 Click on the **Edit Behavours** button, ⚙, to open the ‘Behaviours and Bindings’ graphical editor.

6 Select the **Check Box** widget and add a **Local Value** Behaviour.

A **Check Box** Binding is automatically added, connecting the **Check Box** widget to the **Local Value** Behaviour.

7 Edit Binding settings as shown in Figure 7-18.



a) **Check Box** Widget and Behaviours and Bindings

Edit My Checkbox Bind

Common Properties

Name: My Checkbox Bind

Source Behaviour: My GPI Stimulus Beh

Checked Value: 1

Unchecked Value: 0

b) **Check Box** Binding Settings

Fig. 7-18: Check Box Widget Behaviours and Bindings

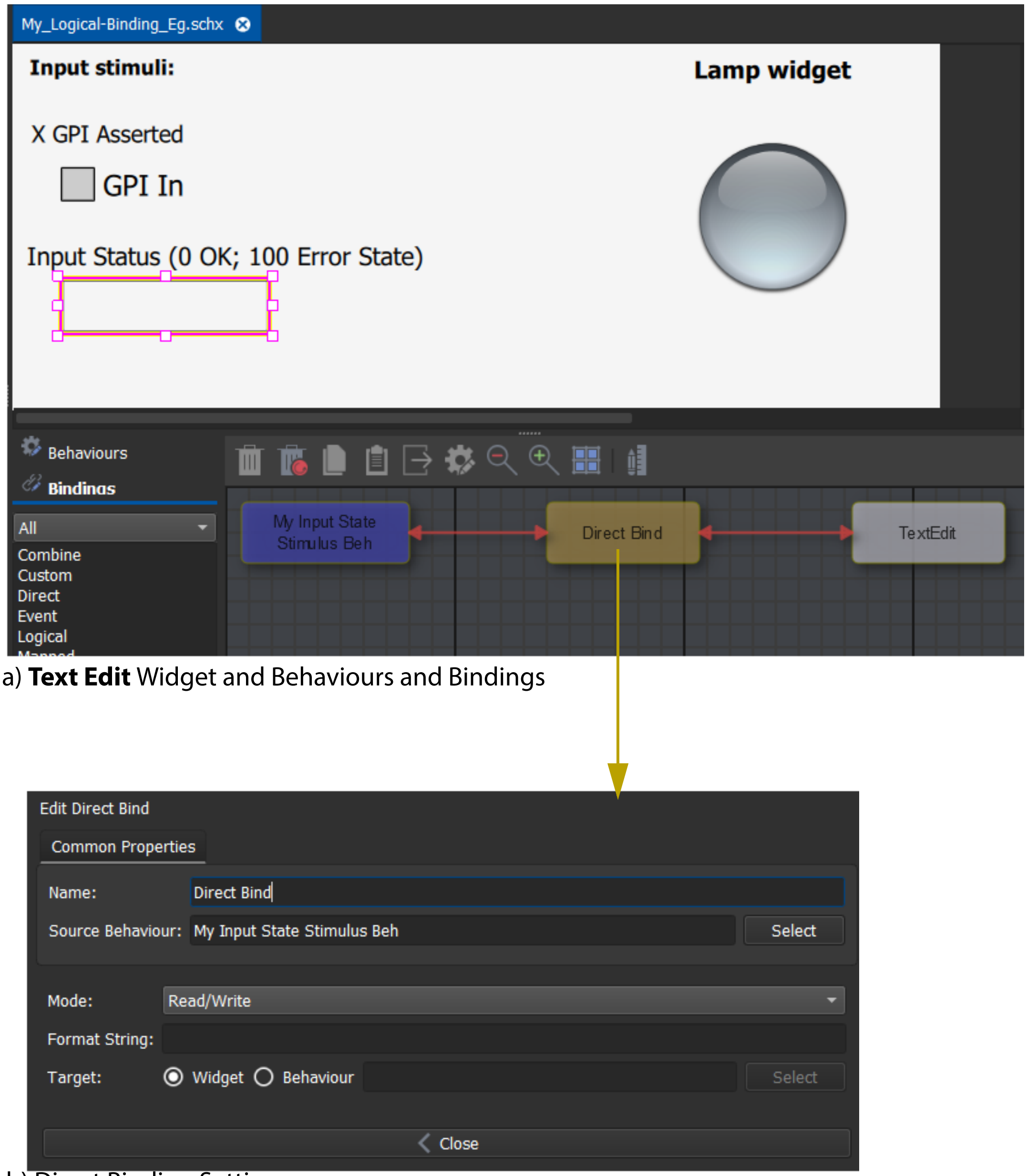
8 Click **Close**.

9 Click **Save**.

10 Select the **Text Edit** widget and add a **Local Value** Behaviour.

A **Direct** Binding is automatically added.

11 Edit Binding settings as shown in Figure 7-19.



a) **Text Edit** Widget and Behaviours and Bindings

Edit Direct Bind

Common Properties

Name: Direct Bind

Source Behaviour: My Input State Stimulus Beh

Mode: Read/Write

Format String:

Target: ● Widget ○ Behaviour

b) Direct Binding Settings

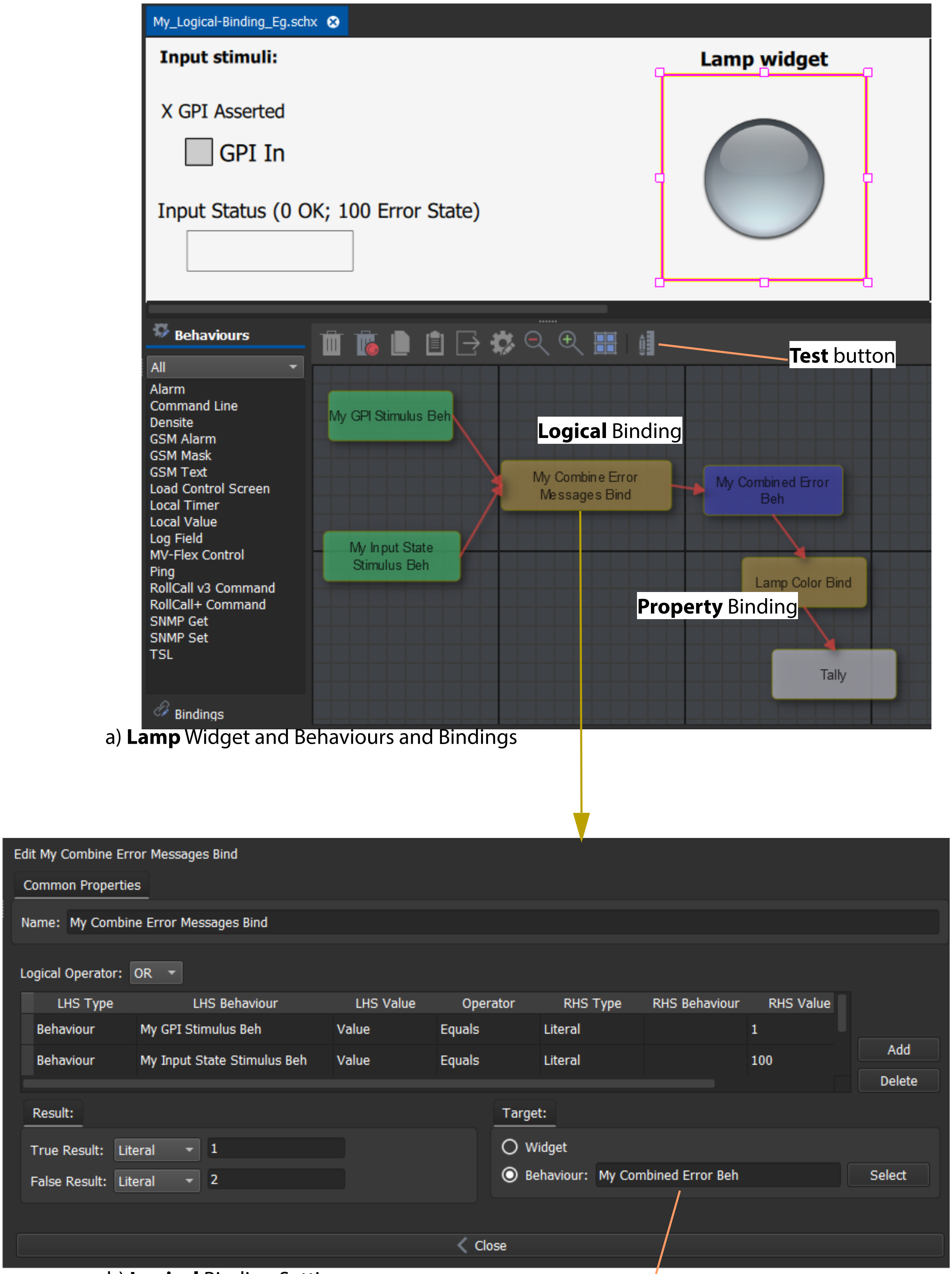
Fig. 7-19: Text Edit Widget Behaviours and Bindings

12 Click **Close**.

13 Click **Save**.

14 Select the **Lamp** widget and add a **Logical** Binding and a **Property** Binding.

15 Edit Binding settings as shown in Figure 7-20 and Figure 7-21.



a) **Lamp** Widget and Behaviours and Bindings

Edit My Combine Error Messages Bind

Common Properties

Name: My Combine Error Messages Bind

Logical Operator: OR

LHS Type	LHS Behaviour	LHS Value	Operator	RHS Type	RHS Behaviour	RHS Value
Behaviour	My GPI Stimulus Beh	Value	Equals	Literal		1
Behaviour	My Input State Stimulus Beh	Value	Equals	Literal		100

Result:

True Result: Literal 1

False Result: Literal 2

Target:

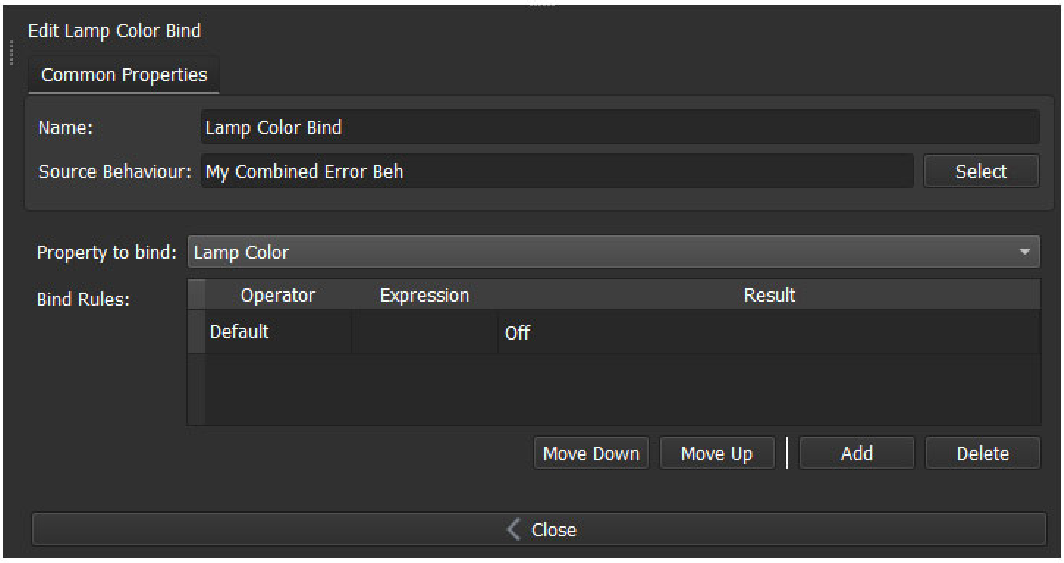
○ Widget

● Behaviour: My Combined Error Beh

b) **Logical** Binding Settings

Target a new **Local Value** Behaviour

Fig. 7-20: Text Edit Widget Behaviours and Bindings



Edit Lamp Color Bind

Common Properties

Name: Lamp Color Bind

Source Behaviour: My Combined Error Beh

Property to bind: Lamp Color

Bind Rules:

Operator	Expression	Result
Default		Off

Fig. 7-21: Property Binding Settings

16 Click **Close**.

17 Click **Save**.

Exercise the Logical Binding

Exercising the screen exercises the Logical Binding used.

1 Select the **Lamp** widget and click the **Test** button.

The widget is exercised and the **Lamp** is illuminated green. See Figure 7-22a.

Exercise the **Lamp** widget by providing test stimuli:

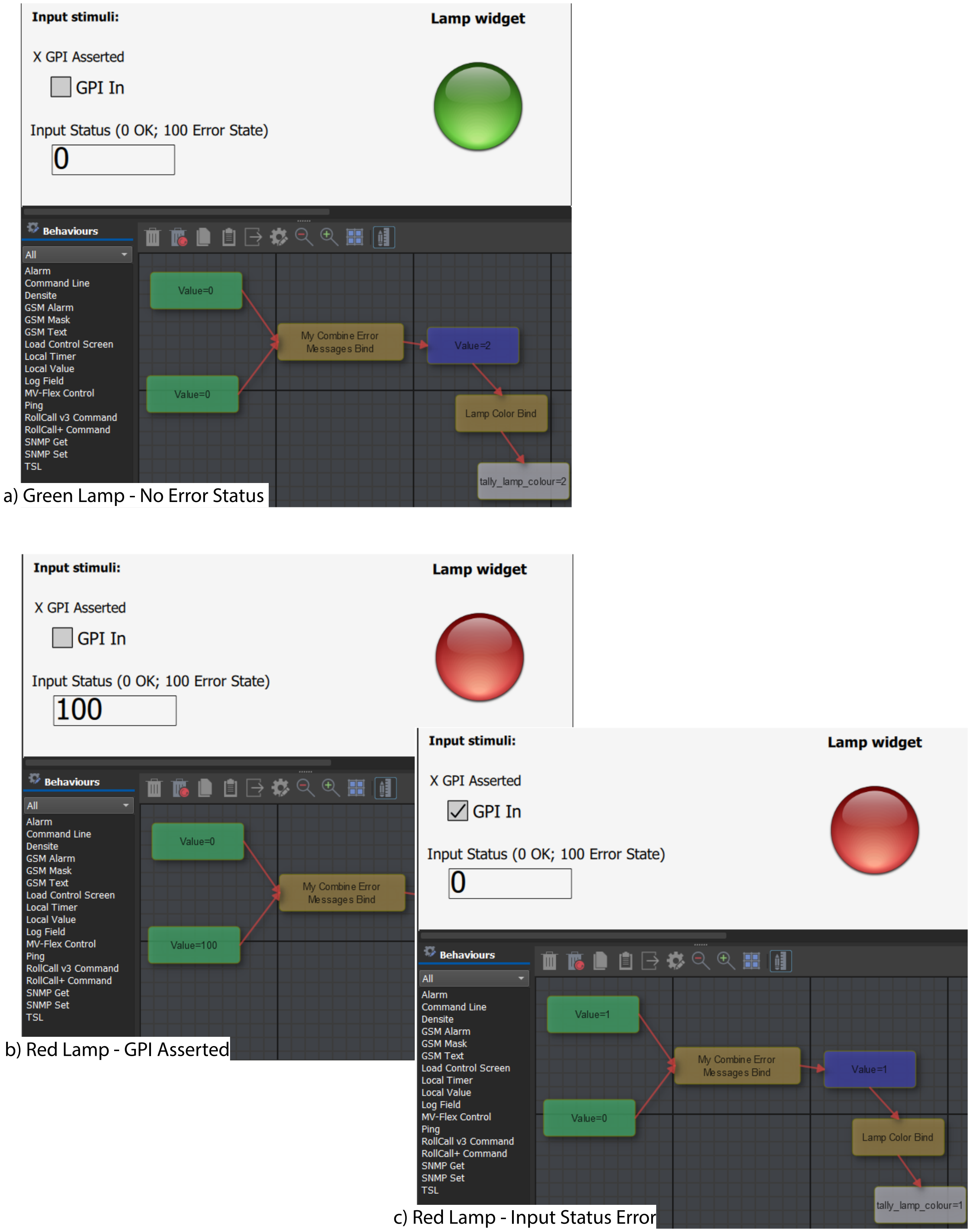
2 Select the **Check Box**.
The **Lamp** illuminates red. See Figure 7-22b.

3 Deselect the **Check Box**.
The **Lamp** illuminates green.

4 Enter ‘100’ at the Text Edit widget.
The **Lamp** illuminates red. See Figure 7-22c.

Halt the exercising:

5 Click the **Test** button.



Input stimuli:

X GPI Asserted

□ GPI In

Input Status (0 OK; 100 Error State)

0

Lamp widget

a) Green Lamp - No Error Status

Input stimuli:

X GPI Asserted

□ GPI In

Input Status (0 OK; 100 Error State)

100

Lamp widget

b) Red Lamp - GPI Asserted

Input stimuli:

X GPI Asserted

☑ GPI In

Input Status (0 OK; 100 Error State)

0

Lamp widget

c) Red Lamp - Input Status Error

Fig. 7-22: Exercising Logical Binding with Lamp Widget:
a) No Error.
b) GPI Asserted.
c) Input Status Error Asserted.

Controlling Border Color

The example of Example - Logical Binding and Simulated GPI or Alarm State, on page 201, controls a on-screen Tally Lamp color. Alternatively, a rectangular border color may be controlled, briefly described here using a rectangle shape widget.

From the example of Example - Logical Binding and Simulated GPI or Alarm State, on page 201:

1. Add a **Rectangle** shape widget.
2. Select the widget and display the ‘Behaviours and Bindings’ graphical editor.
3. Add three **Property** Bindings. (See Figure 7-23.)

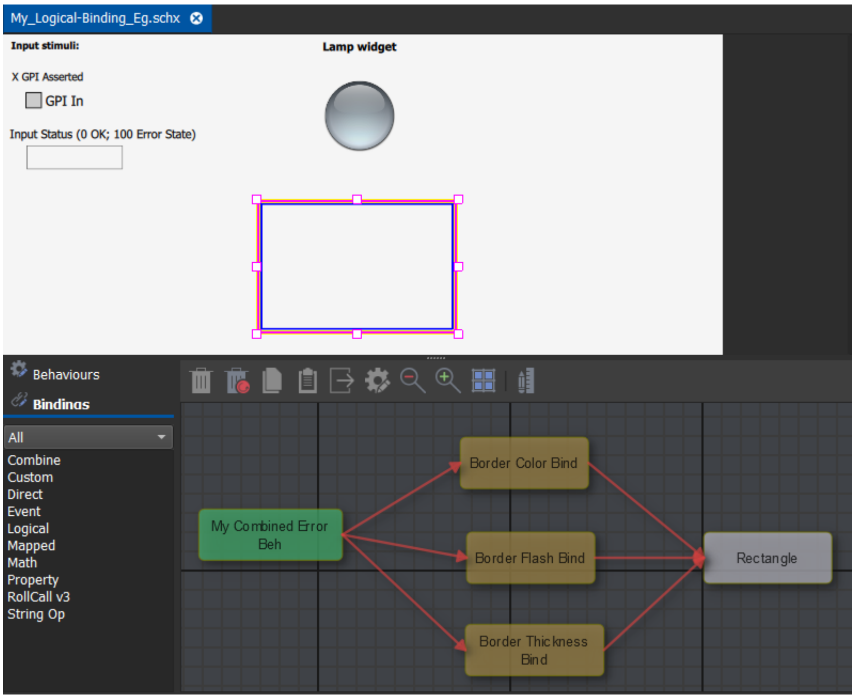
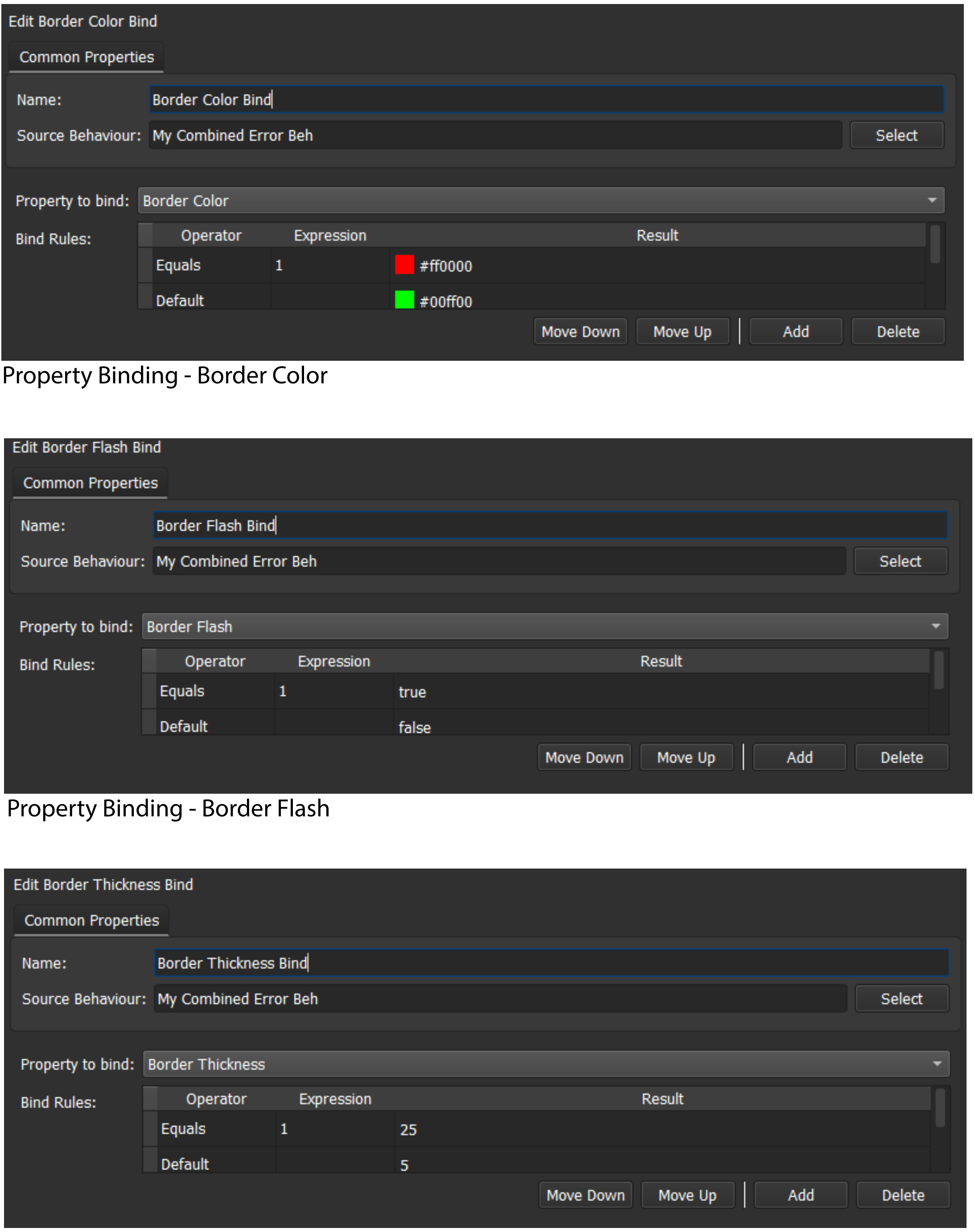


Fig. 7-23: Adding a Rectangle

4. Configure each Binding as shown in Figure 7-24.
   Note: Each Binding is configured to use the same source Behaviour (‘My Combined Error Beh’ from Example - Logical Binding and Simulated GPI or Alarm State, on page 201.)



Name: Border Color Bind

Source Behaviour: My Combined Error Beh

Property to bind: Border Color

Bind Rules:

Operator	Expression	Result
Equals	1	🟥 #ff0000
Default		🟩 #00ff00

a) Property Binding - Border Color

Name: Border Flash Bind

Source Behaviour: My Combined Error Beh

Property to bind: Border Flash

Bind Rules:

Operator	Expression	Result
Equals	1	true
Default		false

b) Property Binding - Border Flash

Name: Border Thickness Bind

Source Behaviour: My Combined Error Beh

Property to bind: Border Thickness

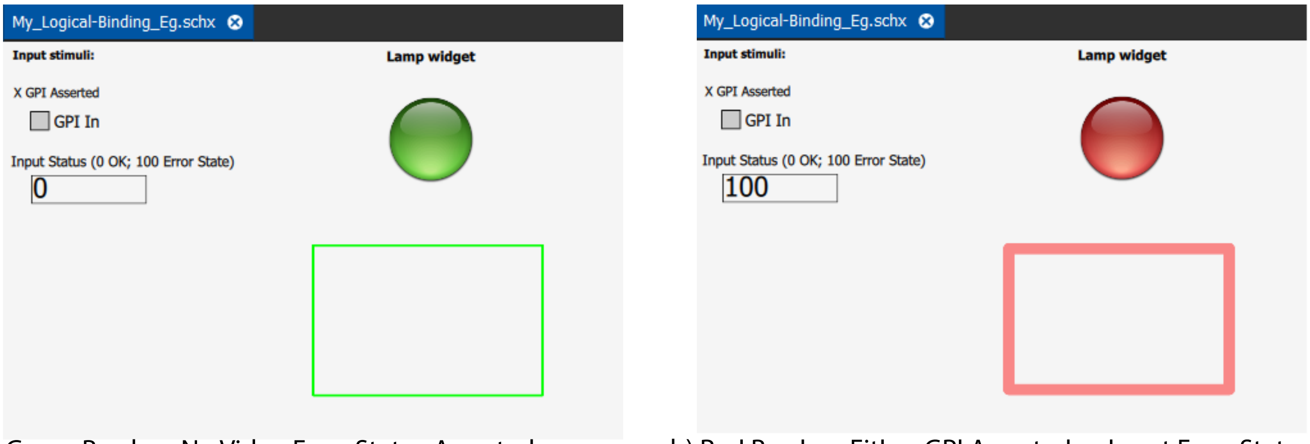
Bind Rules:

Operator	Expression	Result
Equals	1	25
Default		5

c) Property Binding - Border Thickness

Fig. 7-24: Property Binding Configurations

Exercising the design is shown in Figure 7-25.



My_Logical-Binding_Eg.schx

Input stimuli:

Lamp widget

X GPI Asserted

□ GPI In

Input Status (0 OK; 100 Error State)

0

a) Green Border - No Video Error Status Asserted

My_Logical-Binding_Eg.schx

Input stimuli:

Lamp widget

X GPI Asserted

□ GPI In

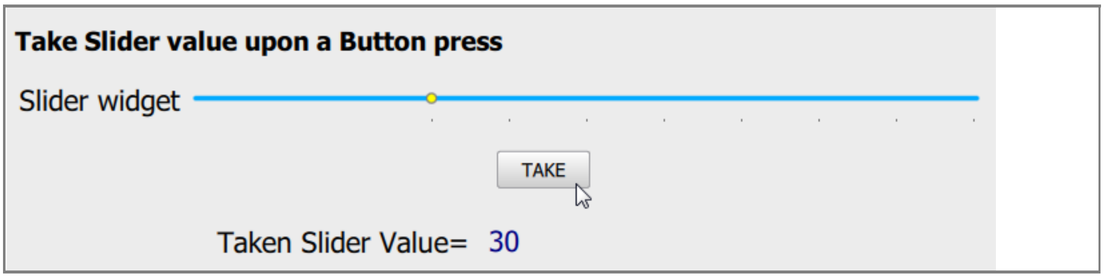
Input Status (0 OK; 100 Error State)

100

b) Red Border - Either GPI Asserted or Input Error Status

Fig. 7-25: Exercising:
a) Thin Green Border.
b) Flashing, Thick Red Border.

Example - Event Binding and 'Taking' a Slider Value



This example uses:

- Widgets - **Slider**, **Button**, **Label**.
- Bindings - **Event**.
- Behaviours - **Local Value**.

This example uses an **Event** Binding to take a snapshot of a value from a **Slider** widget and pass it on for use in a MapView screen.

Build the Example

1. Open a screen schematic.
2. Add a **Slider** widget. This will be our source of data values.
3. Add a **Button** widget. This will have an **Event** Binding.
4. Add a **Label** widget. This will show the resulting slider value taken with a button click.

   See Figure 7-26.

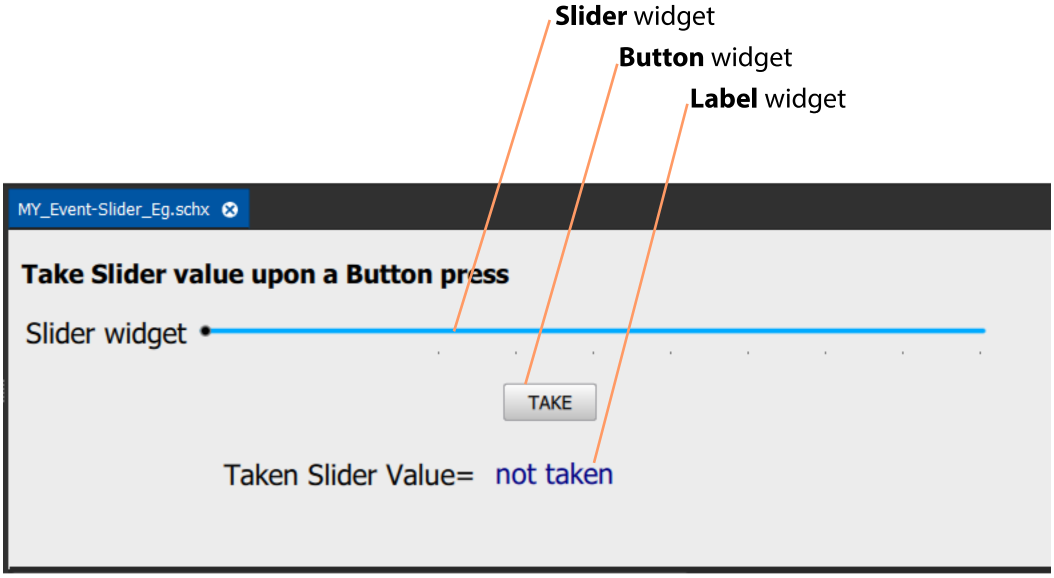


Fig. 7-26: Widgets Added

The following are carried out with the 'Behaviours and Bindings' graphical editor:

5. Click on the **Edit Behavours** button, ⚙, to open the 'Behaviours and Bindings' graphical editor.
6. Select the **Slider** widget and add a **Local Value** Behaviour. See Figure 7-27.

   A **Local Value** Behaviour and **Direct** Binding are added.

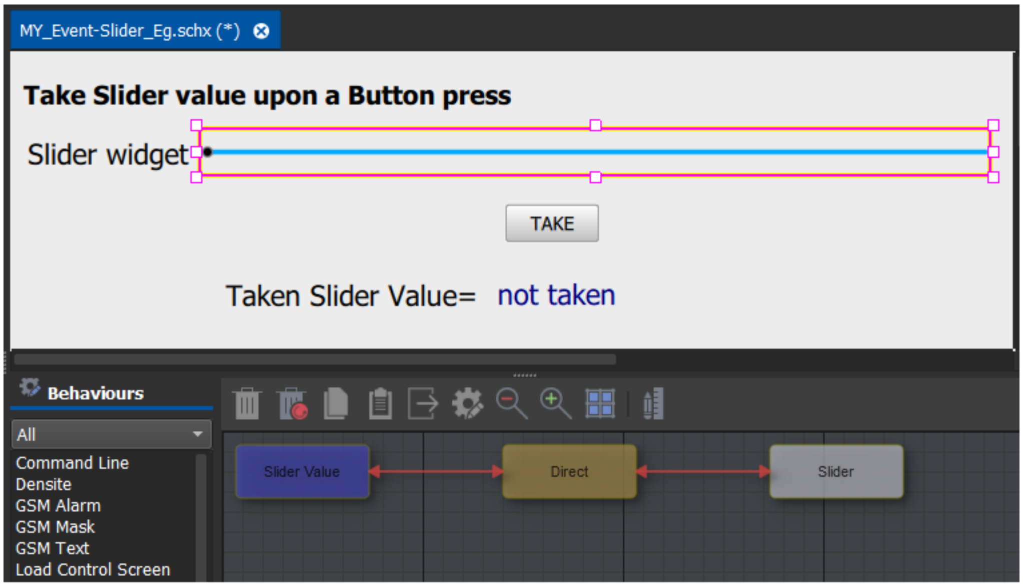


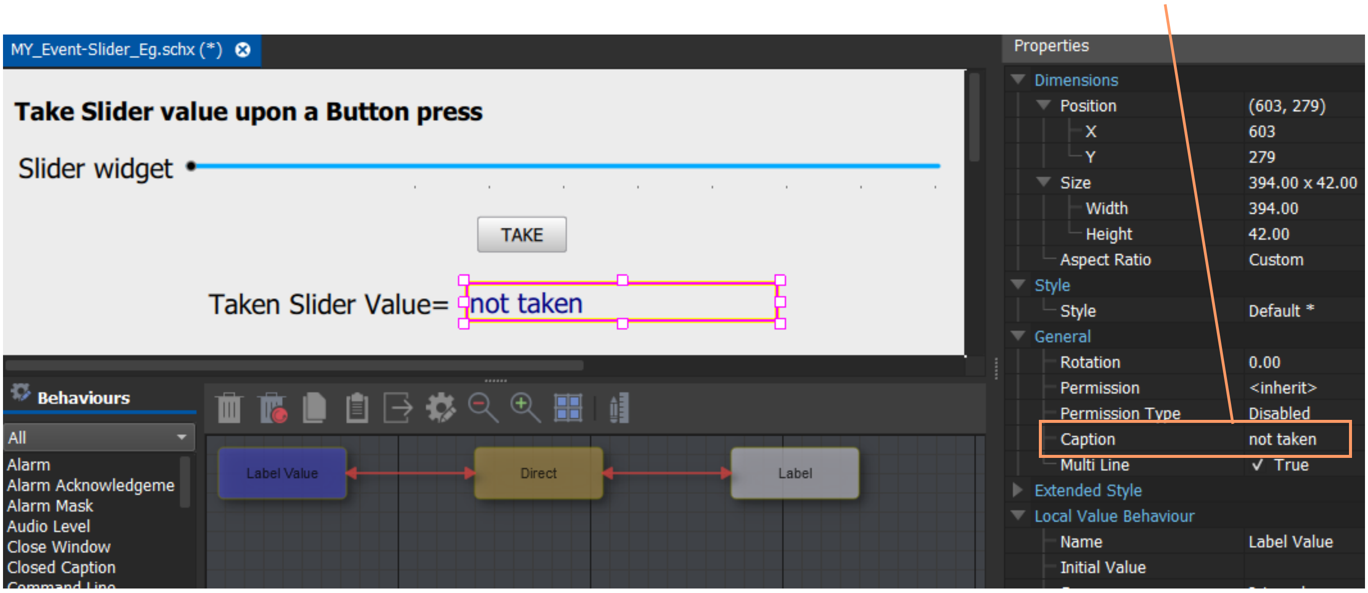
Fig. 7-27: Slider Widget Behaviours and Bindings

7 Select the **Label** widget and set the Caption property to ‘not taken’. See Figure 7-28.

8 Add a **Local Value** Behaviour.

A **Local Value** Behaviour and **Direct** Binding are added.

Set **Caption** property to ‘not taken’



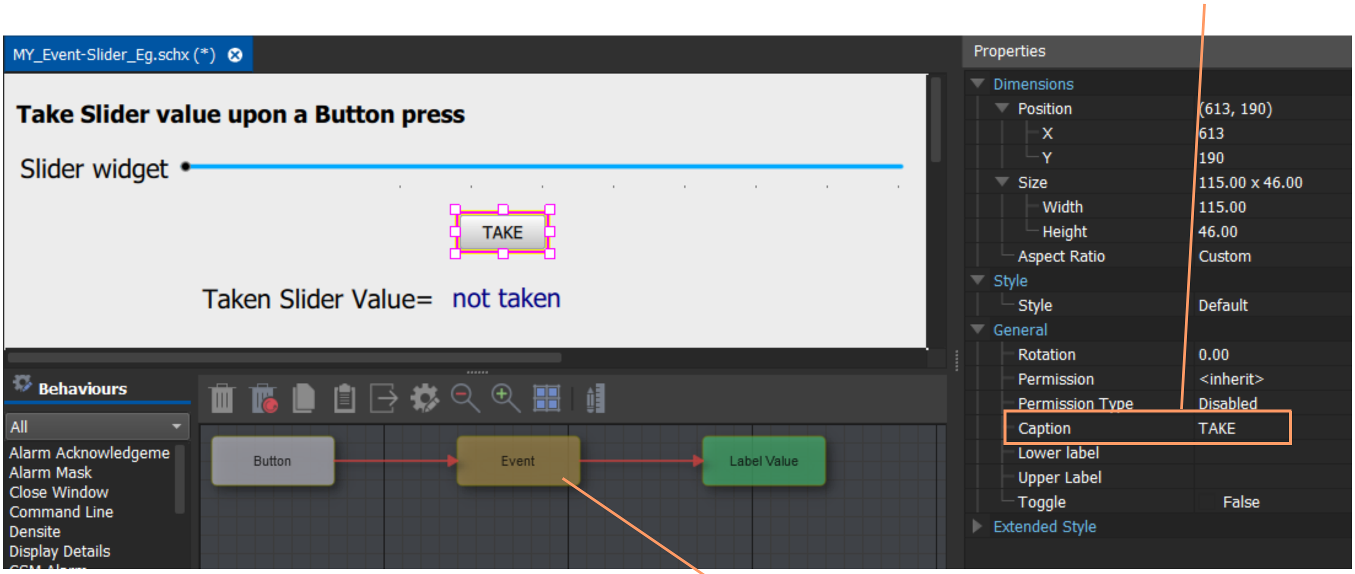
Dimensions	
Position	(603, 279)
X	603
Y	279
Size	394.00 x 42.00
Width	394.00
Height	42.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	not taken
Multi Line	✓ True
Extended Style	
Local Value Behaviour	
Name	Label Value
Initial Value	

Fig. 7-28: Label Widget Behaviours and Bindings

9 Select the **Button** widget and set the Caption property to 'TAKE'. See Figure 7-29.

10 Add an **Event** Binding. See Figure 7-29.

Set **Caption** property to 'TAKE'



Dimensions	
Position	(613, 190)
X	613
Y	190
Size	115.00 x 46.00
Width	115.00
Height	46.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	TAKE
Lower label	
Upper Label	
Toggle	False
Extended Style	

Event Binding

Fig. 7-29: Button Widget Behaviours and Bindings

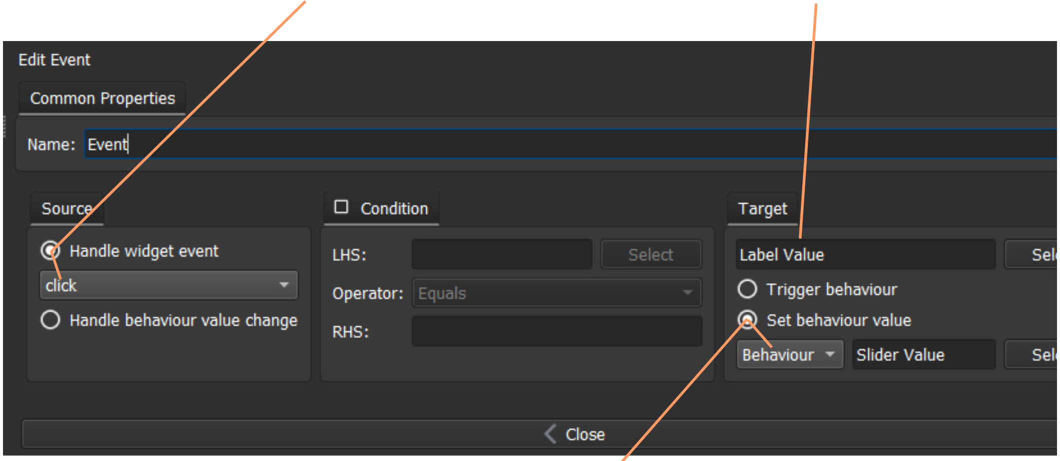
11 Edit the **Event** Binding settings (double-click on the Binding in the graphical editor) to:

- handle a click widget event;
- target the Label value; and
- set the Label value to the Slider value.

See Figure 7-30.

Handle widget click event

Target Behaviour



Name: Event

● Handle widget event
click
○ Handle behaviour value change

□ Condition
LHS:
Operator: Equals
RHS:

Label Value
○ Trigger behaviour
● Set behaviour value
Behaviour Slider Value

Set to the Slider value

Fig. 7-30: Edit Event Binding Settings

12 Click **Close**.

13 Click **Save**.

Exercise the Event Binding

Exercising the screen exercises the **Event** Binding used.

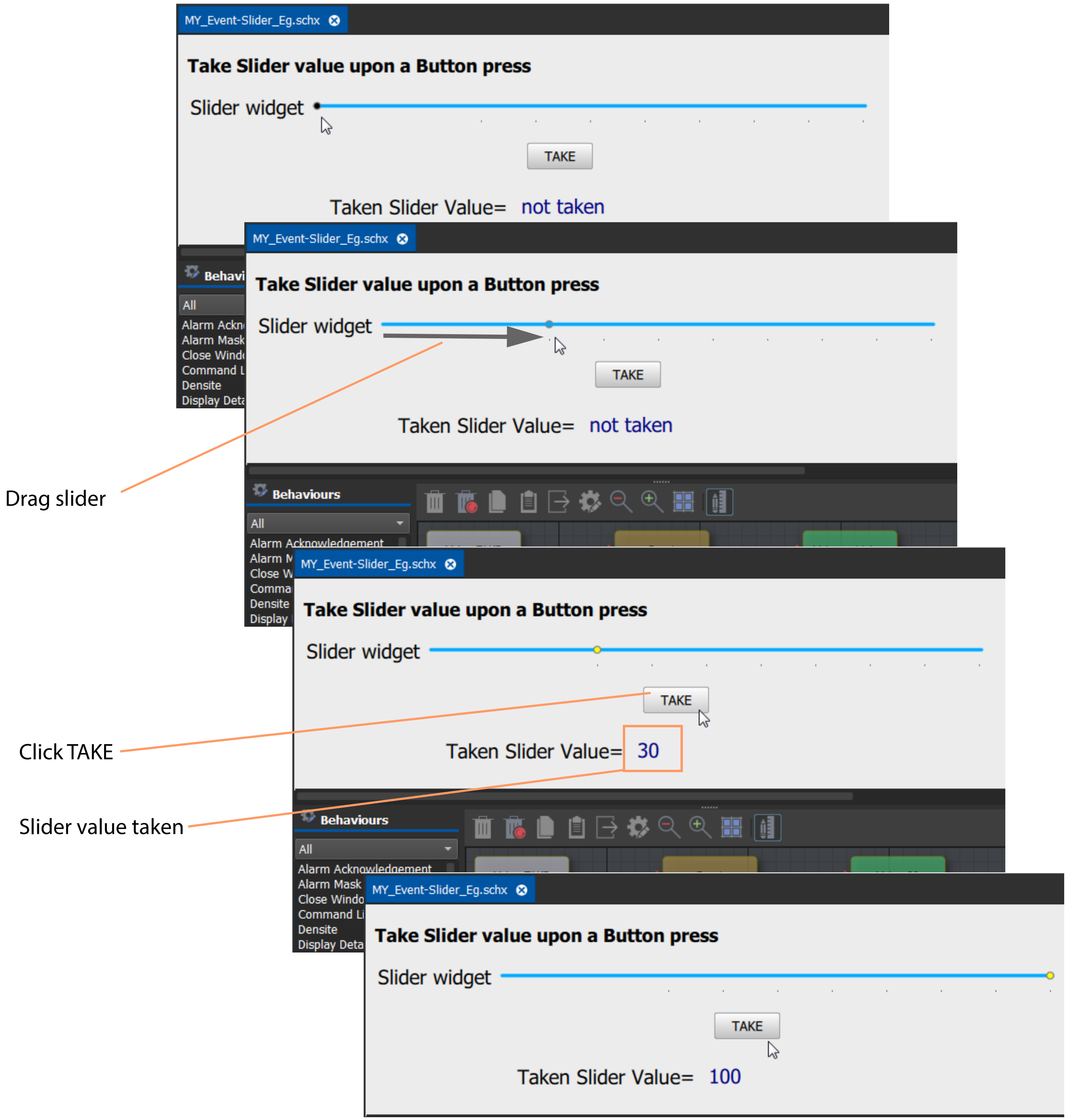
1 Select the **Button** widget and click the **Test** button.

2 Move the slider and then click on the TAKE button.
The label value will show the slider position value when the TAKE button is clicked. (The slider values in this example are from 0 to 100.)

See Figure 7-31.

To halt the exercising:

3 Click the **Test** button.



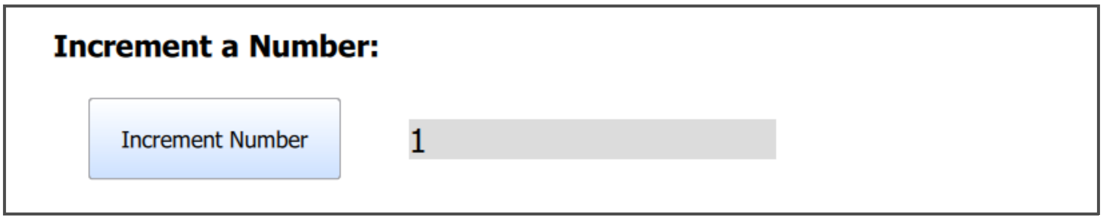
Drag slider

Click TAKE

Slider value taken

Fig. 7-31: Exercising Event Binding with Slider and Button

Example - Button Click Increments a Value by One



This example increments a Behaviour value by one upon a button click and shows how to avoid a non-stabilized loop.

This example uses:

- Widgets - **Button**.
- Bindings - **Event**, **Math**.
- Behaviours - **Local Value**.

Build the Example

1 Create a new screen and open it in the schematic editor.

2 Add a **Button** widget and add a **Label** widget.

3 Set the button's 'Caption' property to 'Increment Number'
and open the 'Behaviours and Bindings' graphical editor.

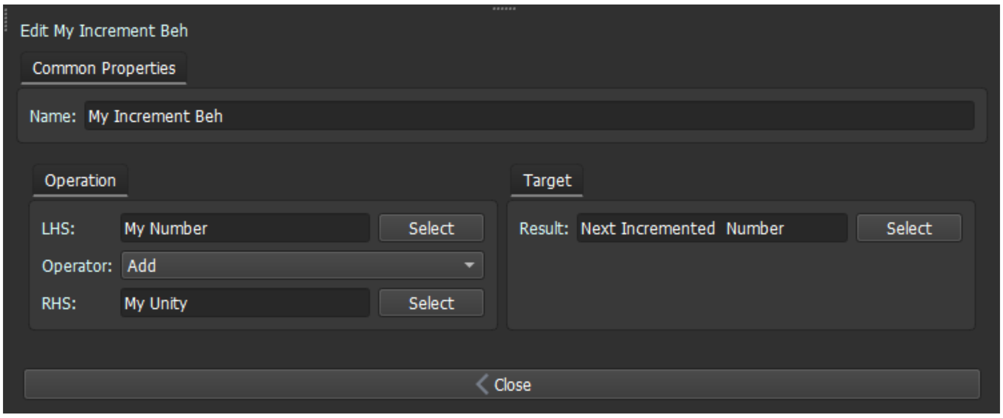
4 Select the **Button** widget on the screen.

5 Add three **Local Value** Behaviours, one **Math** Binding and one **Event** Binding onto the 'Behaviours and Bindings' graphical editor.

6 Configure the **Local Value** Behaviours:

- Name = 'My Number'; Initial Value = 0.
- Name = 'Next Incremented Number'.
- Name = 'My Unity'; Initial Value = 1.

7 Configure the **Math** Binding to add one to 'My Number' and place the result in **Local Value** 'Next Incremented Number'. See Figure 7-32.



Edit My Increment Beh

Common Properties

Name: My Increment Beh

Operation

LHS: My Number

Operator: Add

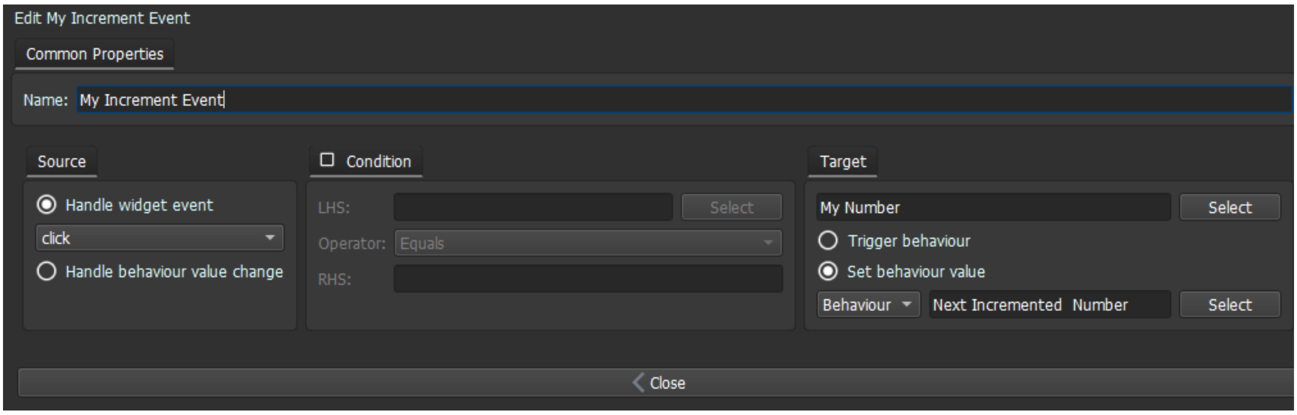
RHS: My Unity

Target

Result: Next Incremented Number

Fig. 7-32: Math Binding Properties

8 Configure the Event Binding to set the value of 'My Number' to the value of 'Next Incremented Number' when the button is clicked. See Figure 7-33.



Edit My Increment Event

Common Properties

Name: My Increment Event

Source
- ● Handle widget event: click
- ○ Handle behaviour value change

☐ Condition
- LHS:
- Operator: Equals
- RHS:

Target
- My Number
- ○ Trigger behaviour
- ● Set behaviour value
- Behaviour: Next Incremented Number

Fig. 7-33: Event Binding Properties

9 Configure the **Label** to show **Local Value** 'My Number', using a **Direct** Binding.

10 Click **Save**.

11 Click **Save Project**.

The resulting 'Behaviours and Bindings' graphical editor for the Button widget is shown in Figure 7-34.

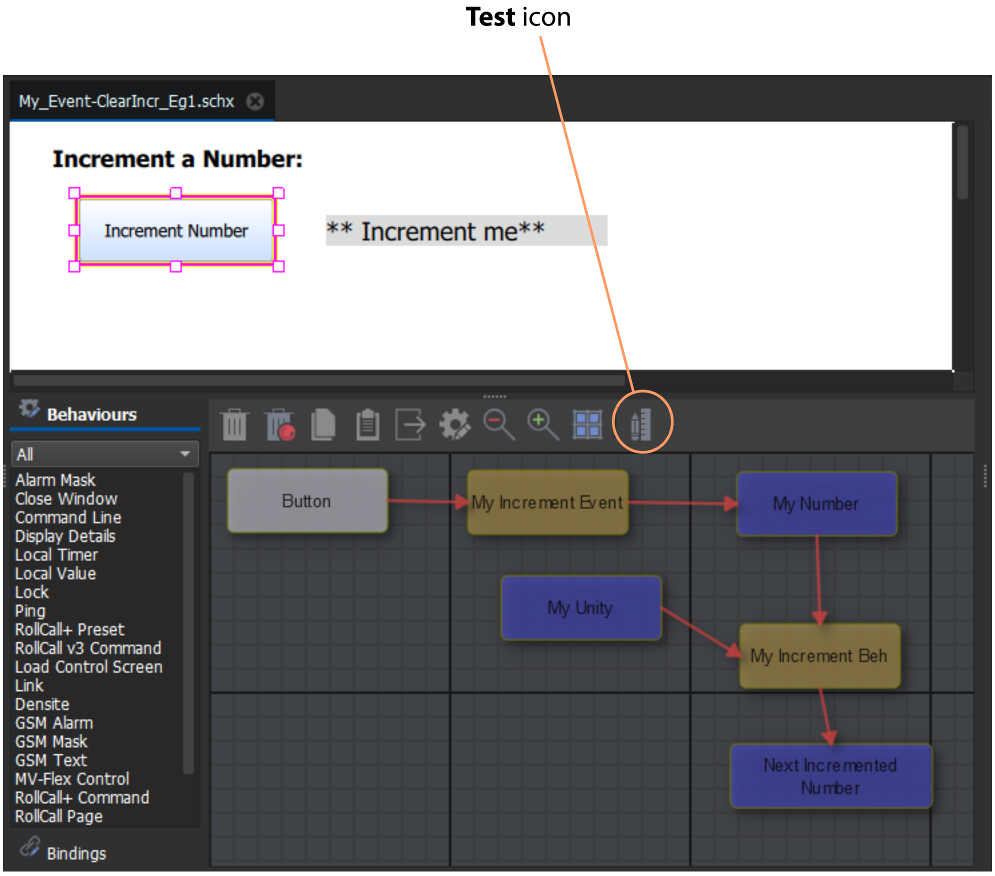


Fig. 7-34: Increment a Number Example

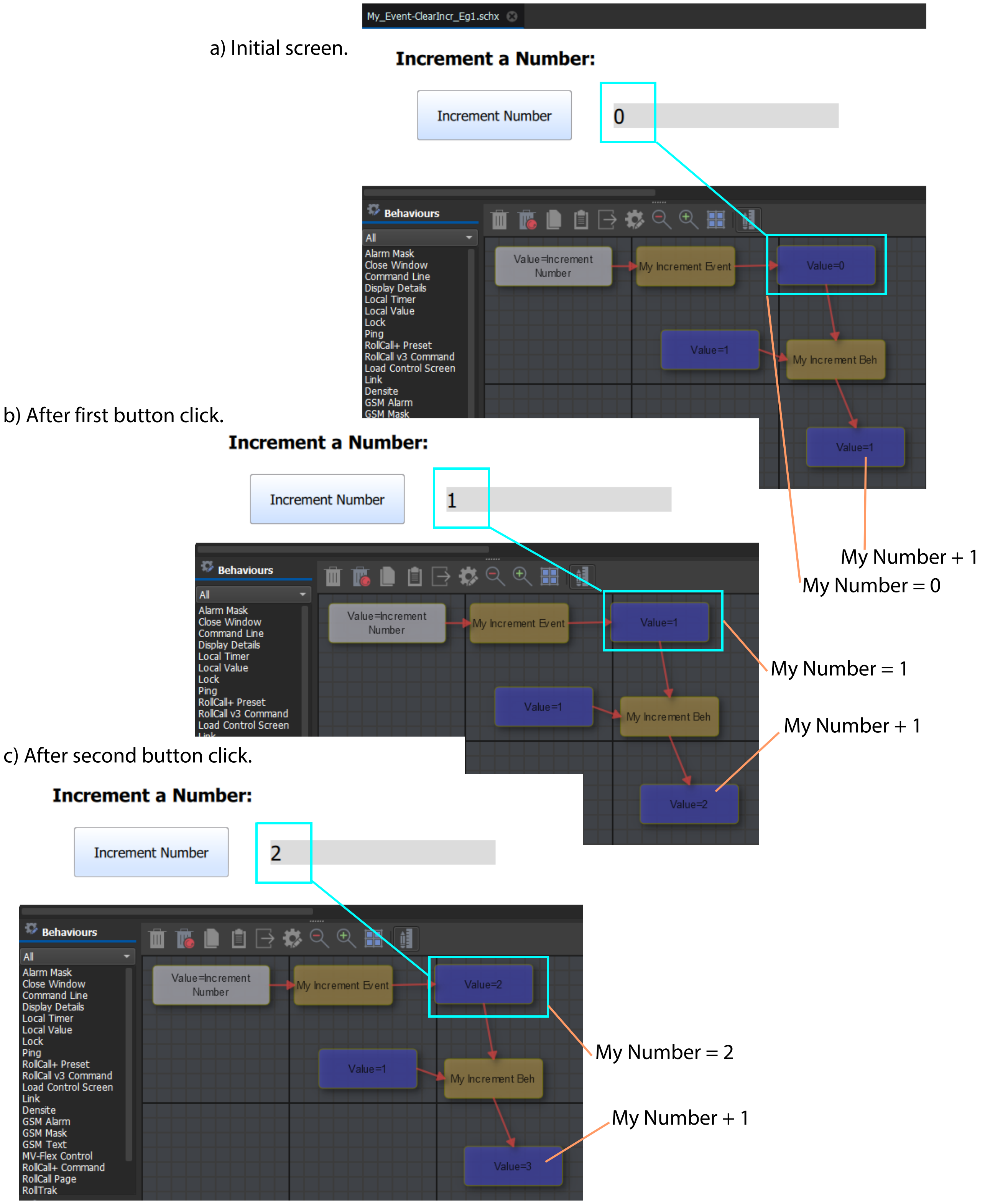
Exercise the Example

1 Select the **Button** widget and click the **Test** icon.

In 'Test' mode, the Behaviours and Bindings associated with the **Button** widget are shown.

2 Exercise the design by clicking the button and seeing the number increment each time. Behaviours and Bindings are annotated with their live values. See Figure 7-35.

3 To stop test mode, click the **Test** icon again.



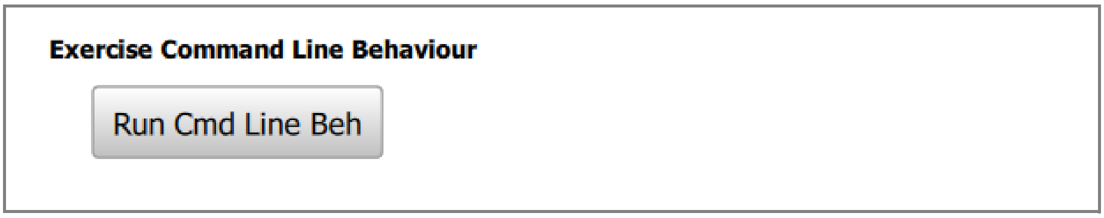
a) Initial screen.

b) After first button click.

c) After second button click.

Fig. 7-35: Exercising the 'Increment a Number' Example:
a) Initial Screen. b) After First Button Click. c) After Second Button Click.

Example - Forming a Text String for a Command Line Behaviour



This example uses a **Combine** Binding to prepare a text string argument for a **Command Line** Behaviour. The example assumes an existing source of 'MY_LOG_HEADER' log messages from RollCall address 1000:E0:01.

This example uses:

- Widgets - **Button**.
- Bindings - **Event**, **Combine**.
- Behaviours - **Command Line, Log Field, Local Value**.

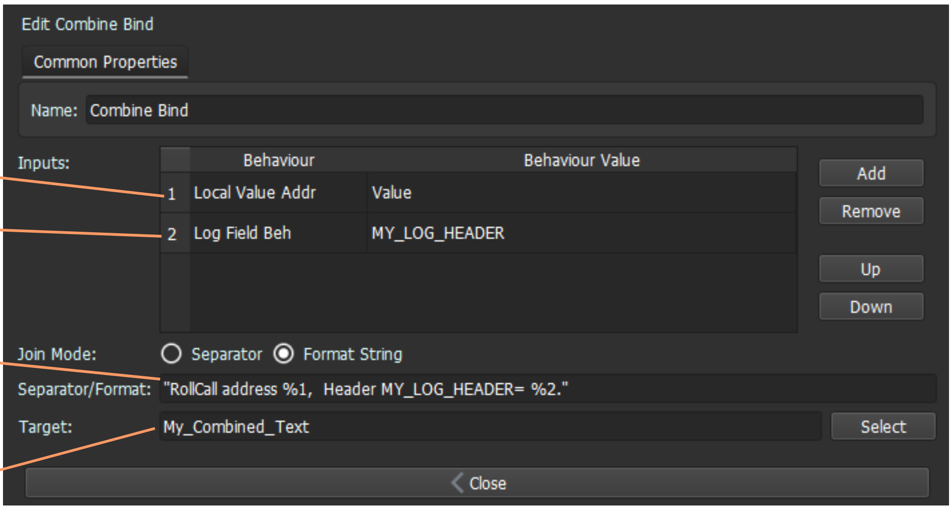
A **Log Field** Behaviour monitors a Log Field from a RollCall address. The Log Field value and the RollCall address are combined to form a text string which is passed to a Windows batch file which is executed via a **Command Line** Behaviour.

Build the Example

Screen Background

1. Create a new MapView screen.
2. Right-click on the screen background, select 'Variables...'
   and set variable 'Address' to '1000:E0:01'.
3. Add a **Local Value** Behaviour
   and set its 'Initial Value' property to '{Address}'.
4. Add a **Log Field** Behaviour
   and set its 'RollCall address' property to '{Address}'
   and its 'Headers' property to 'MY_LOG_HEADER'.
5. Add a **Combine** Binding.
6. Double-click on the Binding to edit its properties.

   In this example, the RollCall address and the Log Header value are combined and placed into another **Local Value**. See Figure 7-36.



RollCall address added

Log Field value added

Format string

Local Value Behaviour target to hold combined text string

Edit Combine Bind

Common Properties

Name: Combine Bind

Inputs:

	Behaviour	Behaviour Value
1	Local Value Addr	Value
2	Log Field Beh	MY_LOG_HEADER

Join Mode: ○ Separator ● Format String

Separator/Format: "RollCall address %1, Header MY_LOG_HEADER= %2."

Target: My_Combined_Text

Fig. 7-36: Combine Binding Property Settings

Note the use of double-quotes ("   ") in the format string.

7 Click **Close**.

The screen background's Behaviours and Binding are shown in Figure 7-37.

RollCall address

Log Field value

Local Value Behaviour holds combined text string

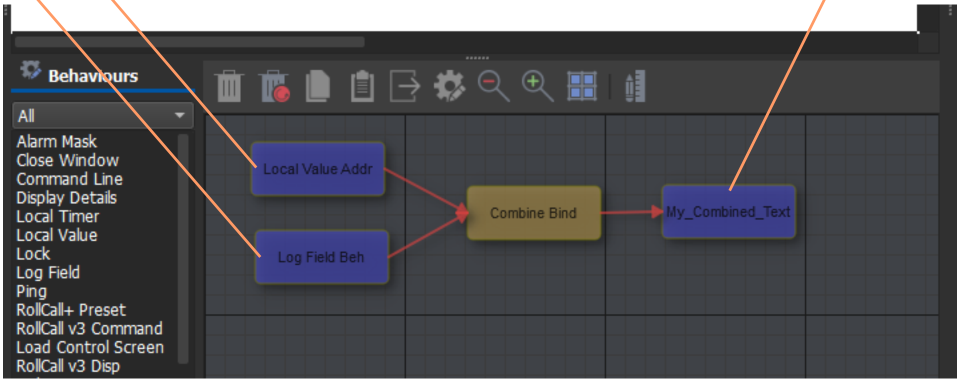


Fig. 7-37: Behaviours and Binding (Screen Background)

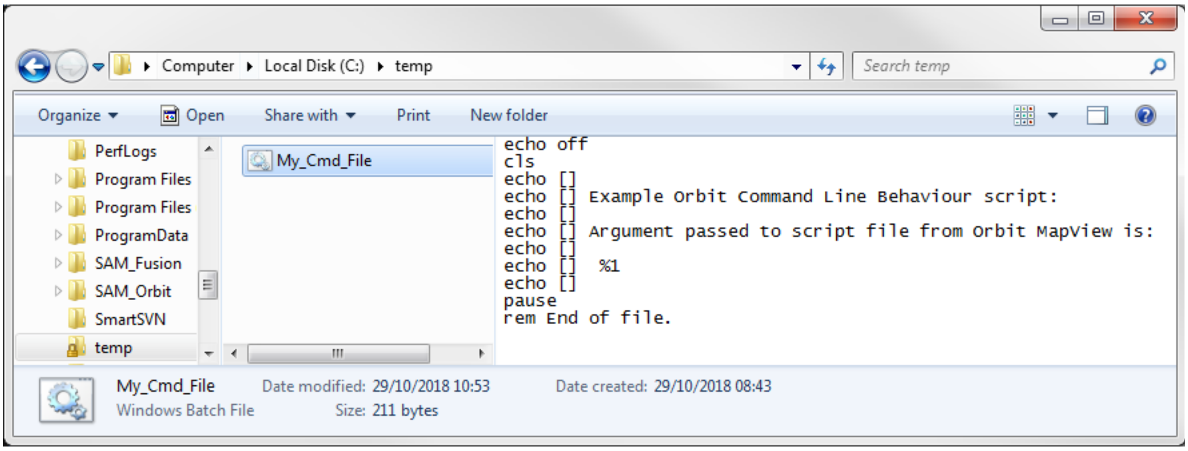
Command File for a Command Line Behaviour to Run

1 Create a batch file (.bat) on the client computer.

For example, create batch file My_Cmd_File.bat (see Figure 7-38) at location C:\temp\ and enter the following via a text editor:

```
echo off
cls
echo ▯
echo ▯ Example Orbit Command Line Behaviour script:
echo ▯
```

echo ▯ Argument passed to script file from Orbit MapView is:
echo ▯
echo ▯ %1
echo ▯
pause
rem End of file.



```
echo off
cls
echo []
echo [] Example Orbit Command Line Behaviour script:
echo []
echo [] Argument passed to script file from Orbit MapView is:
echo []
echo []  %1
echo []
pause
rem End of file.
```

My_Cmd_File Date modified: 29/10/2018 10:53 Date created: 29/10/2018 08:43
Windows Batch File Size: 211 bytes

Fig. 7-38: Batch File

Button Widget

On the MapView screen in Orbit:

1 Add a **Button** widget, an **Event** Binding and a **Command Line** Behaviour. See Figure 7-39.

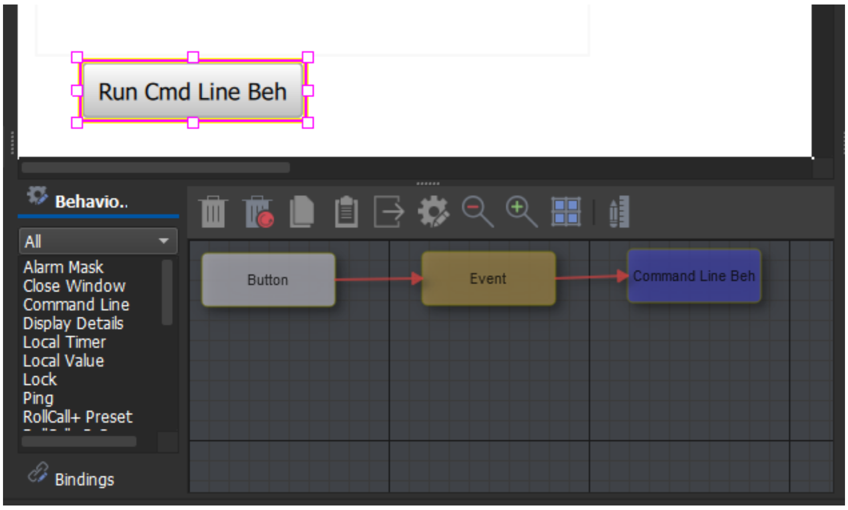


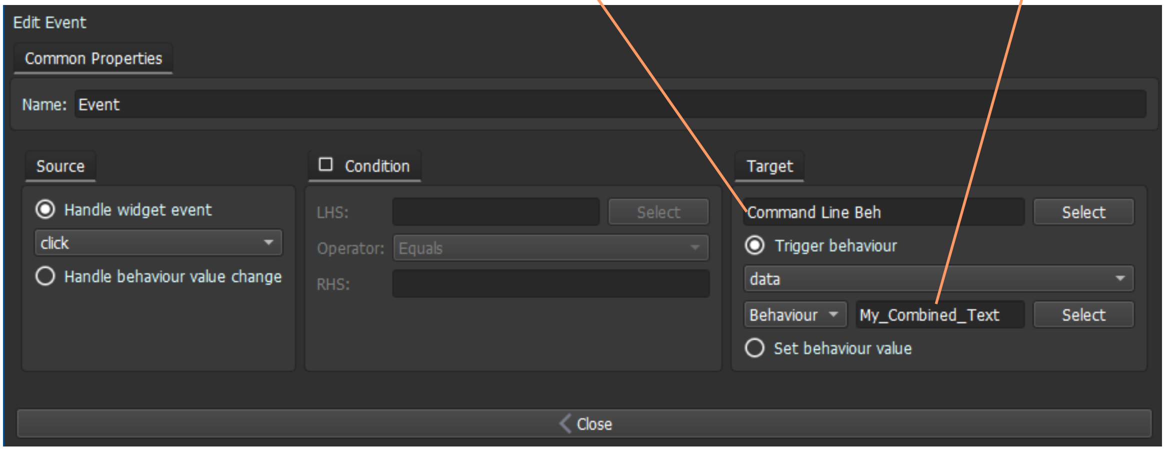
Fig. 7-39: Button Widget, Event Binding and Command Line Behaviour

2 Set the **Button** widget's 'Caption' property to describe the button's function, for example, 'Run Cmd Line Beh'.

3 Set the **Event** Binding to run the **Command Line** Behaviour when the button is clicked and to pass the combined text string to it as an argument. See Figure 7-40.

Target set to the **Command Line** Behaviour

Argument to pass is the combined text string



Edit Event

Common Properties

Name: Event

Source

● Handle widget event

click

○ Handle behaviour value change

□ Condition

LHS:

Operator: Equals

RHS:

Target

Command Line Beh

● Trigger behaviour

data

Behaviour My_Combined_Text

○ Set behaviour value

Fig. 7-40: Event Binding Settings

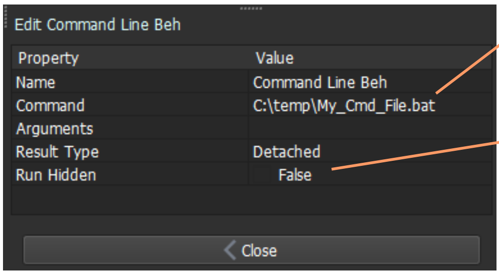
4 Set the ‘Command’ property of the **Command Line** Behaviour to be the path to the batch file. (C:\temp\My_Cmd_File.bat) See Figure 7-41.

5 Click **Close**.

6 Click **Save**.

7 Click **Save Project**.

Path to the batch file



Edit Command Line Beh

Property	Value
Name	Command Line Beh
Command	C:\temp\My_Cmd_File.bat
Arguments	
Result Type	Detached
Run Hidden	False

Set ‘Run Hidden’ property to ‘False’, to be able to see the command prompt window when it is run.

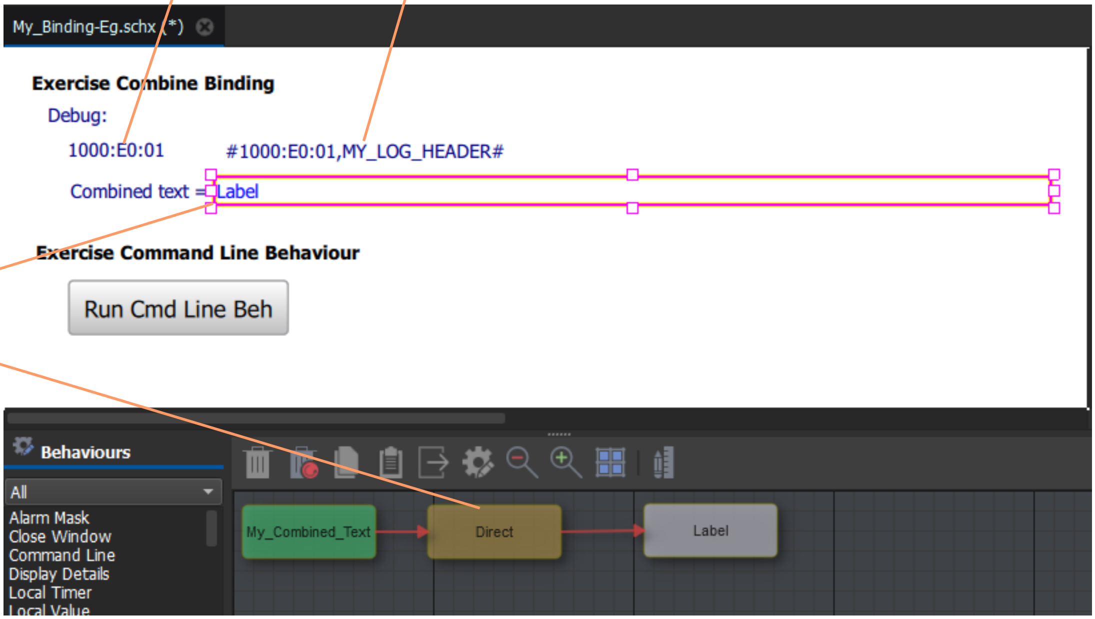
Fig. 7-41: Command Line Behaviour Settings

When the **Command Line** Behaviour is executed, the batch file will be run with the combined text string as an argument. The batch file will open a command prompt window, and print out text which will include the combined text string.

Note: Fixed arguments can be passed to the **Command Line** Behaviour with the ‘Arguments’ property.
However, dynamic arguments, such as in this example, are passed to the Behaviour where it is invoked.
(In this example, an **Event** Binding is used to invoke the **Command Line** Behaviour and pass a dynamic argument.)

Adding some Debug

It is useful to be able to see some of the interim values while getting any example to run correctly. Figure 7-42 shows some debug labels added to the screen.



Label caption = {Address}

Label caption = #{Address},MY_LOG_HEADER#

(See for #Hash Field# Syntax, on page 164, for the hash fields # #.)

Label caption set with a Direct Binding

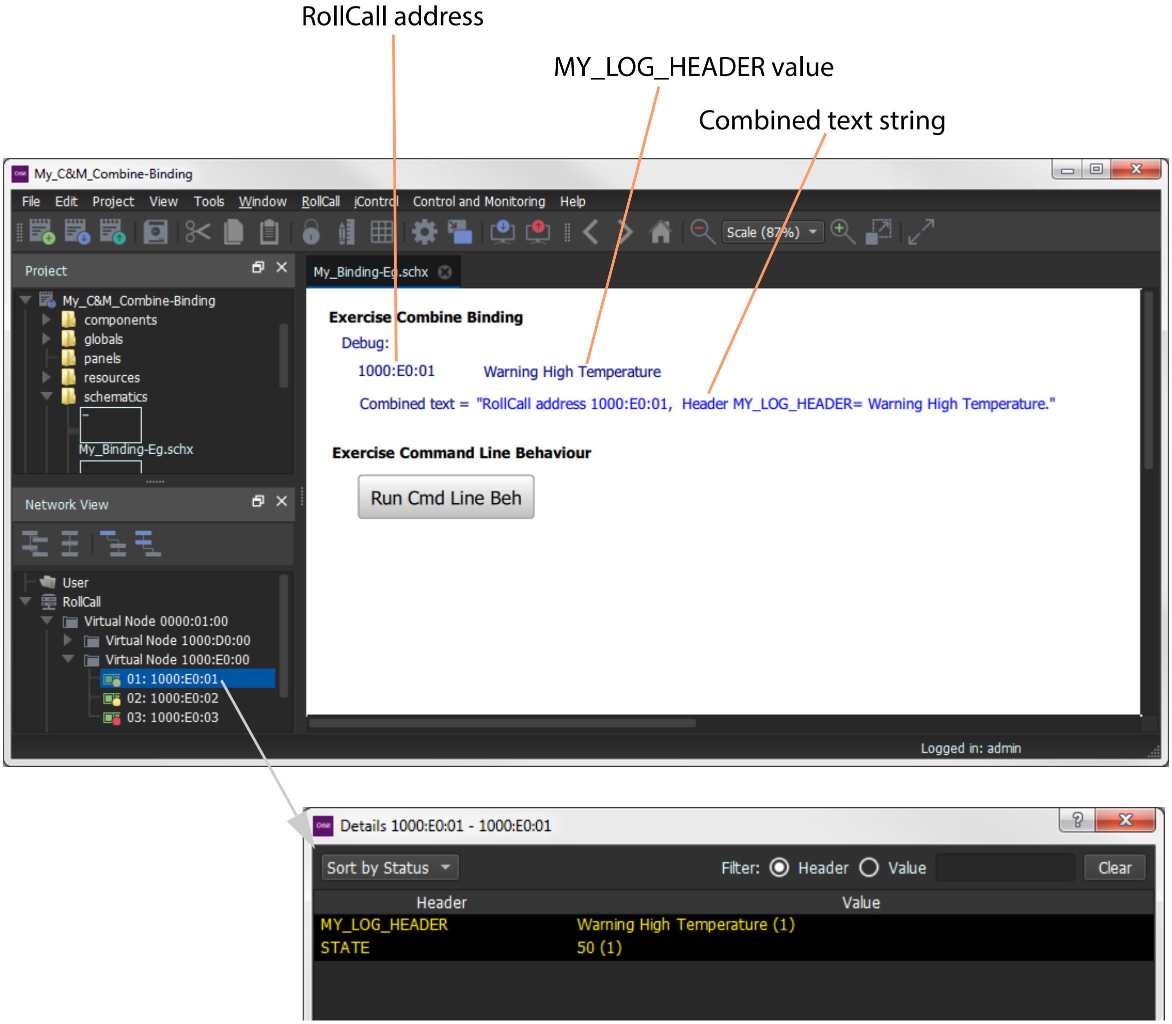
Fig. 7-42: Labels used for Debug

When all debug is added, then save the screen and save the project.

Exercise the Example

1 Click the **Run Mode** button in the main menu.

The initial screen is shown, see Figure 7-43.



Exercise Combine Binding

Debug:

1000:E0:01 Warning High Temperature

Combined text = "RollCall address 1000:E0:01, Header MY_LOG_HEADER= Warning High Temperature."

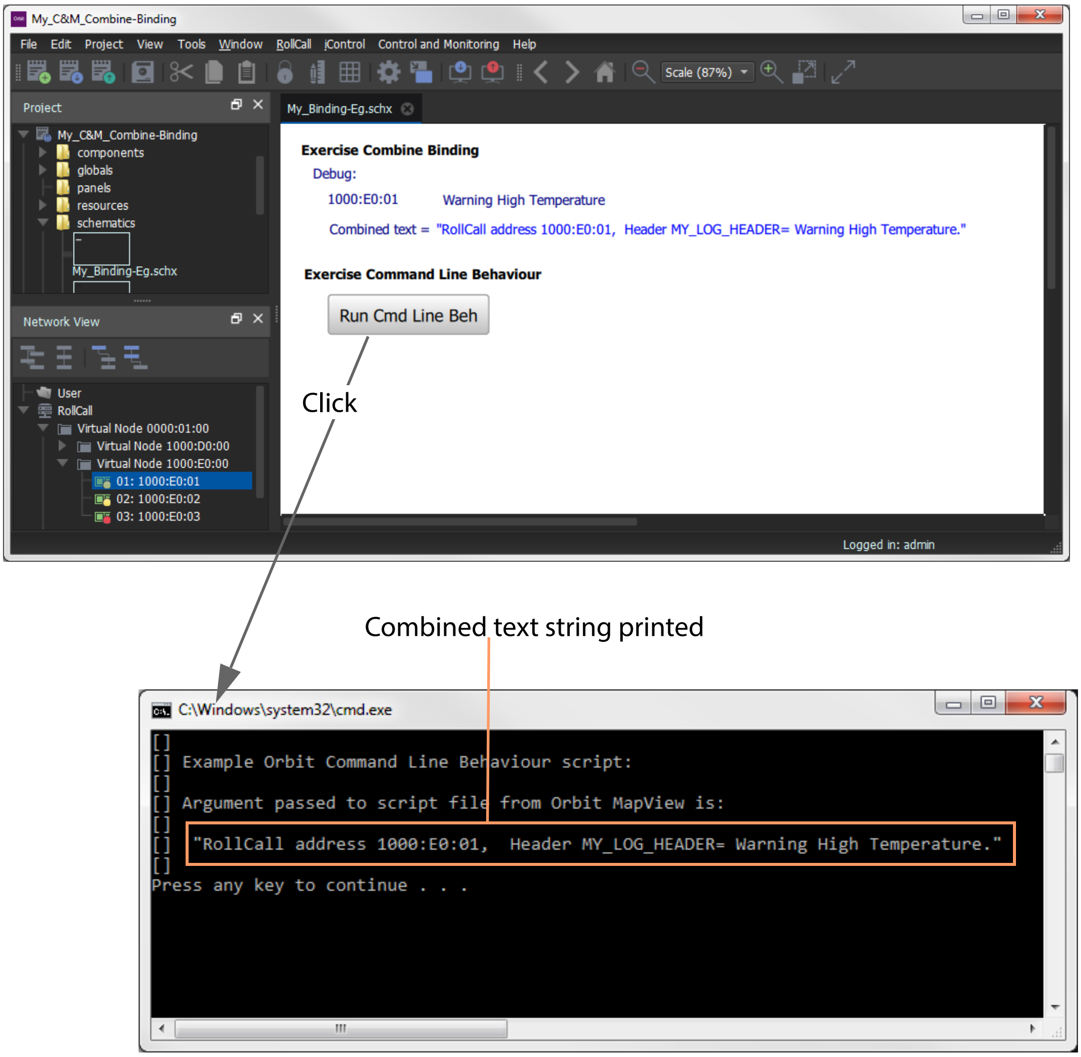
Exercise Command Line Behaviour

Header	Value
MY_LOG_HEADER	Warning High Temperature (1)
STATE	50 (1)

Fig. 7-43: Initial Screen when Run

2 Click the **Button** widget.

The command file is executed in a command line window. See Figure 7-44.



Exercise Combine Binding

Debug:

1000:E0:01 Warning High Temperature

Combined text = "RollCall address 1000:E0:01, Header MY_LOG_HEADER= Warning High Temperature."

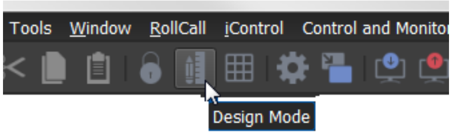
Exercise Command Line Behaviour

```
[]
[] Example Orbit Command Line Behaviour script:
[]
[] Argument passed to script file from Orbit MapView is:
[]
[] "RollCall address 1000:E0:01,  Header MY_LOG_HEADER= Warning High Temperature."
[]
Press any key to continue . . .
```

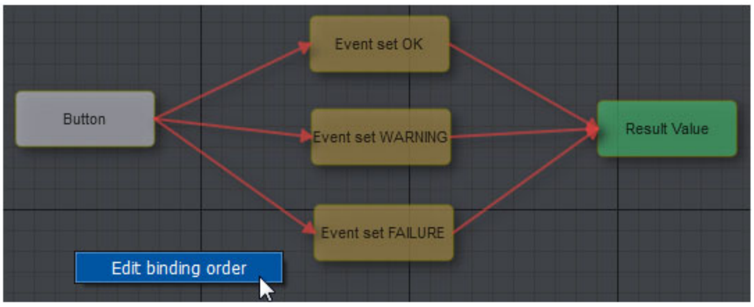
Fig. 7-44: Command Line Behaviour Executes Command File

3 In the command window, press any key.
The window closes.

4 To halt the exercising,
click the **Design Mode** button in the main menu.



Binding Execution Order



Bindings Controlling a Behaviour Value

A Behaviour value may be connected to more than one Binding. In this case, the Behaviour value will be from the last Binding to have executed.

For example, an external device which is being controlled from a MapView screen and also from some other party.

The value of a **Local Value** Behaviour on the MapView screen may then depend on:

- input from a user via a **Text Edit** widget on a MapView screen; and
- a value reported by an external device.

The value of the **Local Value** Behaviour will come from the last Binding to have been executed - either a change made by the user, or a value reported by the external device. This is the required functionality.

Determining Binding Execution Order

In some cases, Bindings controlling a Behaviour value may be triggered by the same event and a resulting value will depend on the order in which the Bindings are executed by Orbit. The Binding execution order can be set in Orbit.

For example, some state value is tested to determine if it is ‘OK’, or a ‘Warning’, or a ‘Failure’ condition with three Bindings. See Figure 7-45.

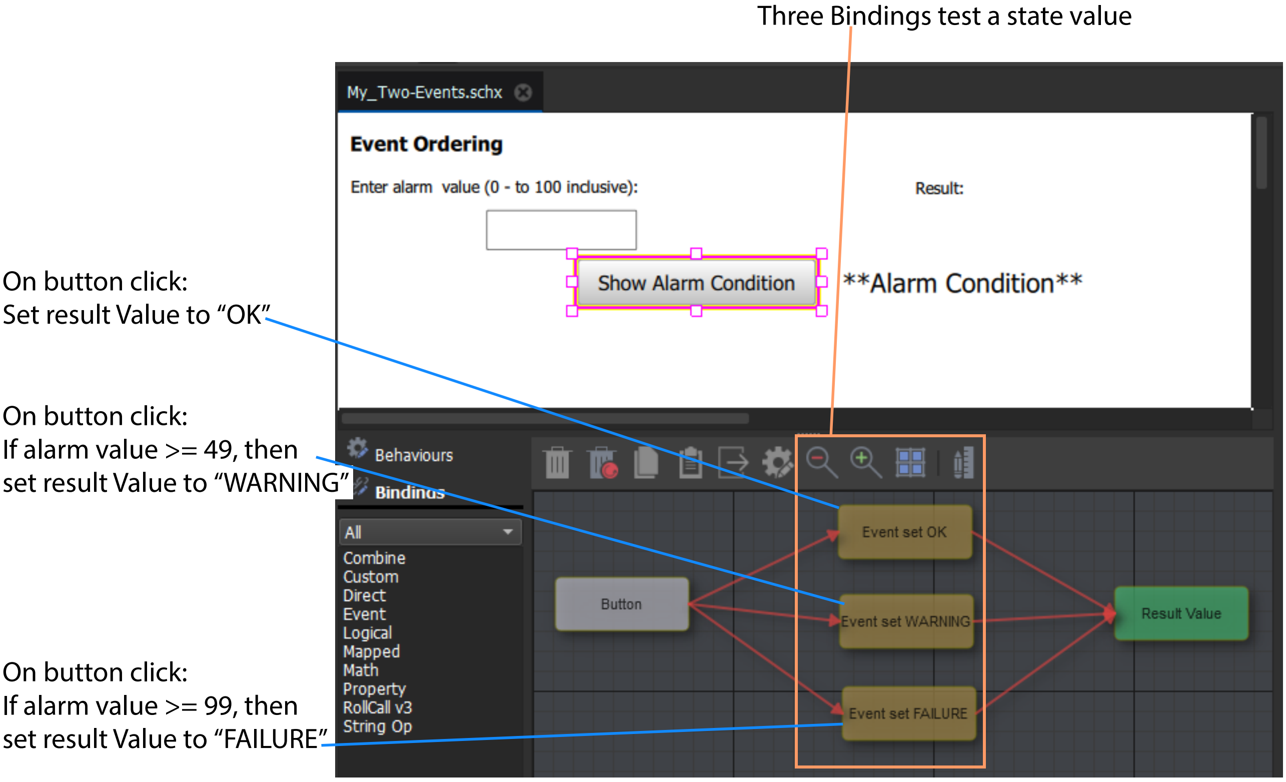


Fig. 7-45: Binding Order Example

The three Event Bindings used set the 'Result Value' to different values (OK, Warning or Failure). The resulting value will be determined by the last Binding to be executed.

In our example here, we want a 'Failure' to take precedence, followed by a 'Warning'. The Event Binding order we want is:

- 1: "Event set OK"; then
- 2: "Event set WARNING"; and lastly
- 3: "Event set FAILURE".

To set the **Event** Binding order of execution:

1 Ensure you are in 'Design Mode' in Orbit.

2 Select the **Button** widget on the screen
and show the 'Behaviour and Binding' graphical editor.

3 Right-click in the 'Behaviour and Binding' graphical editor
and select 'Edit binding order'.
The **Binding Order** dialog is shown. See Figure 7-46.

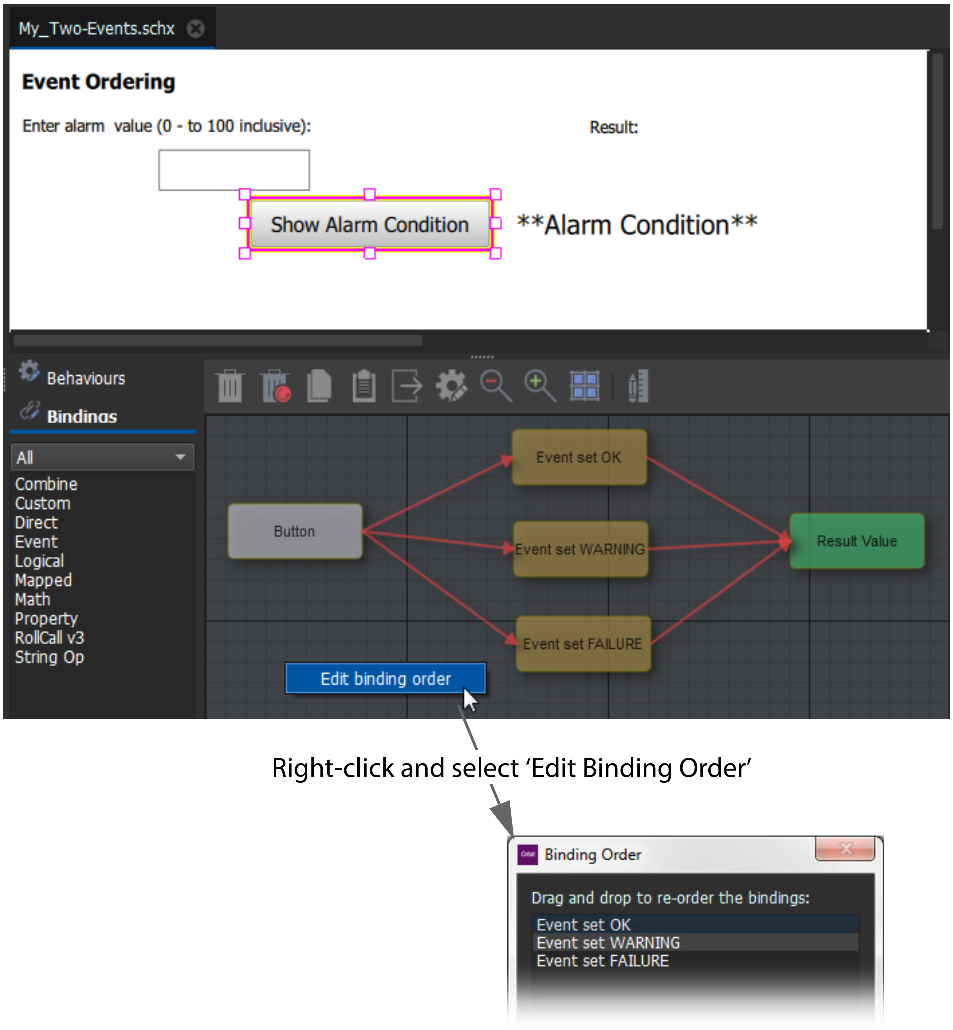


Fig. 7-46: Edit Binding Order Dialog

The order of execution of the Bindings is determined by the **Binding Order** dialog.

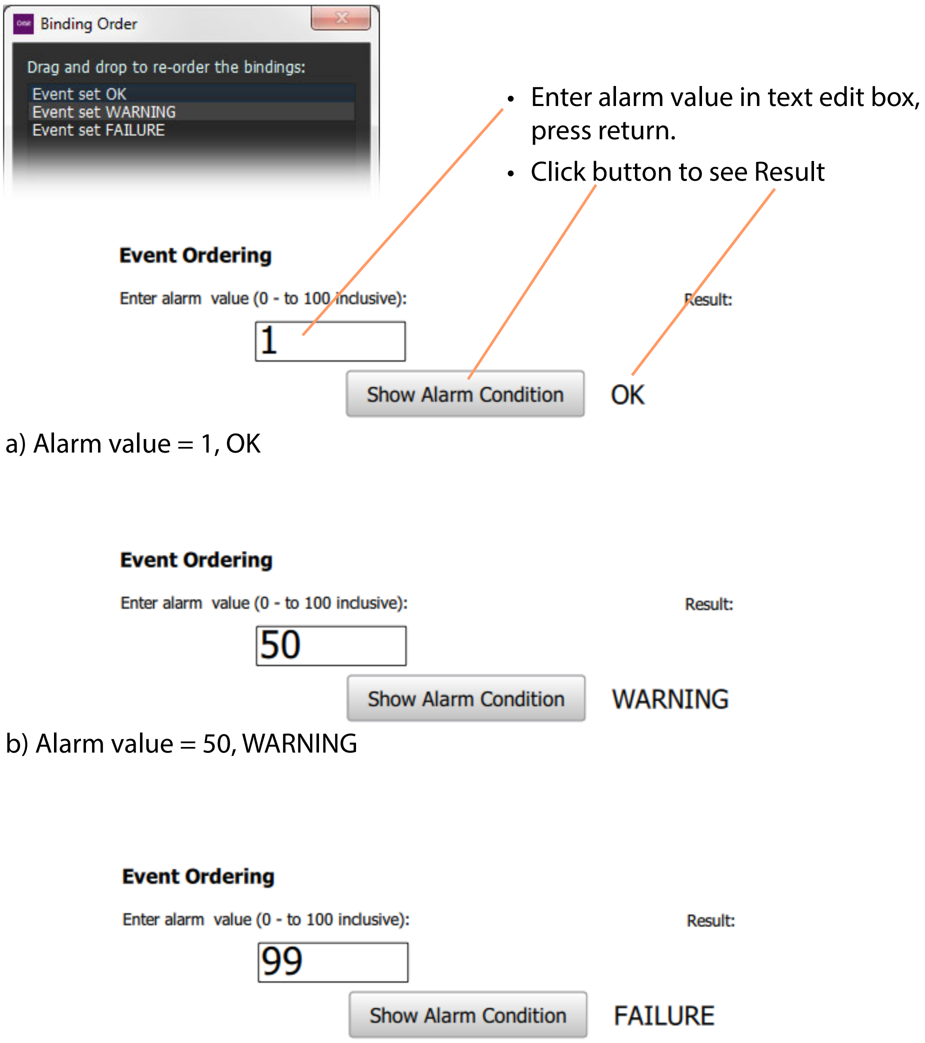
4 Drag and drop the listed items to re-order the **Binding Order** list as required.

(For this example, the order shown in Figure 7-46 is required.
The 'Event set FAILURE' Binding must the last to be executed and will overwrite the Result Value if the alarm value indicates a failure.)

Note: This example illustrates a point about Binding order and uses three **Event** Bindings.
The example itself would normally be done with a **Mapped** Binding to convert a value into one of the three states 'OK', 'Warning' or 'Failure'.

Exercising the Example

Figure 7-47 shows the example being exercised for different entered alarm values.



Binding Order

Drag and drop to re-order the bindings:

Event set OK
Event set WARNING
Event set FAILURE

- Enter alarm value in text edit box, press return.
- Click button to see Result

Event Ordering

Enter alarm value (0 - to 100 inclusive): 1

Result: OK

a) Alarm value = 1, OK

Event Ordering

Enter alarm value (0 - to 100 inclusive): 50

Result: WARNING

b) Alarm value = 50, WARNING

Event Ordering

Enter alarm value (0 - to 100 inclusive): 99

Result: FAILURE

c) Alarm value = 99, FAILURE

Fig. 7-47: Exercising the Event Binding Order Example

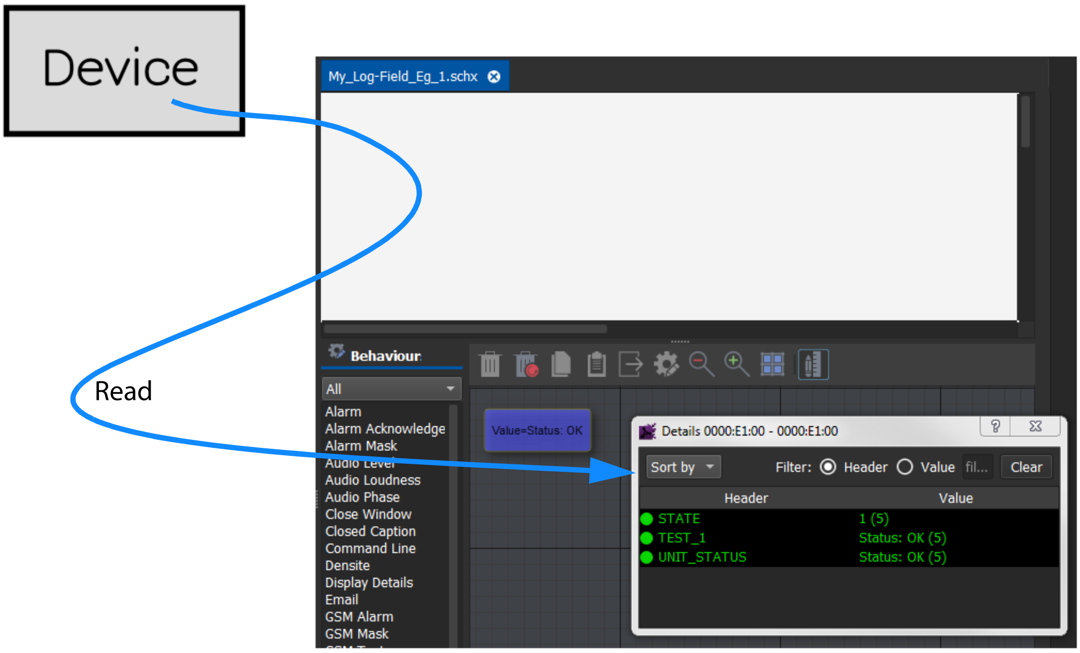
8 Examples with Behaviours

Summary

Examples with Behaviours

This chapter presents some MapView-specific examples showing the use of Behaviours with Bindings.

Example - Read a Device’s Log Field



Details 0000:E1:00 - 0000:E1:00

Header	Value
● STATE	1 (5)
● TEST_1	Status: OK (5)
● UNIT_STATUS	Status: OK (5)

plus Using the #Hash Field# Syntax

This example reads and displays a Log Field header with Orbit MapView from a RollCall-enabled device with RollCall address ‘0000:E1:00’. It assumes a Log Sever and Orbit Services are running and monitoring devices in a system.

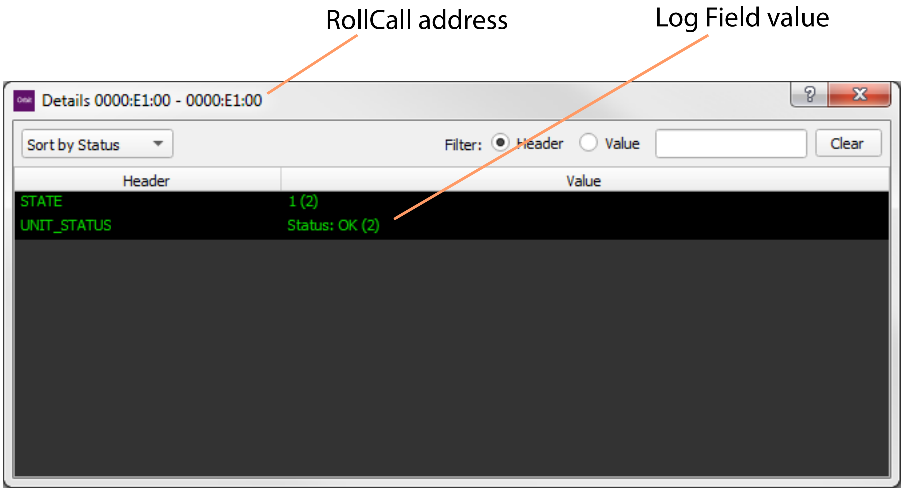
This example uses:

- Widgets - none.
- Bindings - none.
- Behaviours - **Log Field**.
- the **Hash Field** syntax (# #)
(see Using the #Hash Field# Syntax, on page 232 below).

Log Field Source Data

In the Orbit Network View pane:

1. Right-click on a RollCall-enabled device (our example uses RollCall address 0000:E1:00) and select ‘Details’.
2. A Details window is shown which shows the live log messages (Header-Value) being issued by the RollCall-enabled device.



RollCall address

Log Field value

Details 0000:E1:00 - 0000:E1:00

Header	Value
STATE	1 (2)
UNIT_STATUS	Status: OK (2)

Fig. 8-1: Log Field Behaviour

Read a Log Field

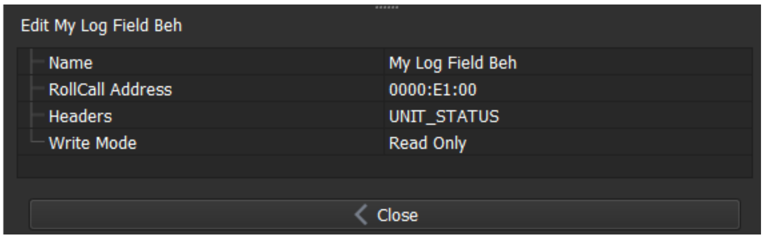
1 Open a new Orbit MapView screen and show the 'Behaviour and Bindings' graphical editor.

2 Add a **Log Field** Behaviour.

3 Edit the Behaviour properties to be:

- RollCall Address: 0000:E1:00 (or any RollCall address of a device issuing log messages.)
- Headers: UNIT_STATUS (I.e. any Log Field header name being issued by the RollCall address.)
- Write Mode: Read Only

See Figure 8-2.



Edit My Log Field Beh

Name	My Log Field Beh
RollCall Address	0000:E1:00
Headers	UNIT_STATUS
Write Mode	Read Only

Fig. 8-2: Log Field Behaviour Settings

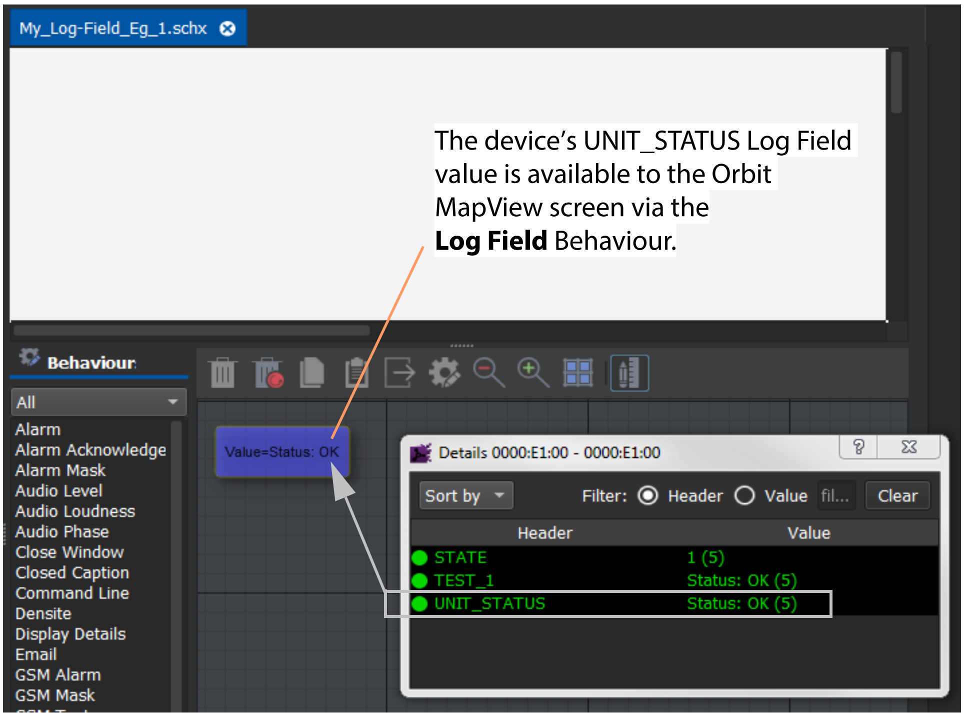
4 Click **Close**.

5 Click **Save** to save the screen schematic.

Exercise Log Field Reading

1 Click the **Test** button ( ) to exercise the screen.

The Log Field Behaviour value reflects the UNIT_STATUS log header value from the RollCall address. See Figure 8-3.



Header	Value
● STATE	1 (5)
● TEST_1	Status: OK (5)
● UNIT_STATUS	Status: OK (5)

The device’s UNIT_STATUS Log Field value is available to the Orbit MapView screen via the **Log Field** Behaviour.

Fig. 8-3: Log Field Value Shown by Log Field Behaviour

2 Click the **Test** button again to stop exercising the screen.

Using the #Hash Field# Syntax

In Read a Log Field, on page 231, a **Log Field** Behaviour is explicitly used when a Log Field value is required on a MapView screen. For simple cases where property values are simply set to Log Field values, a ‘hash field’ syntax short-hand can be used.
(Refer to #Hash Field# Syntax, on page 164.)

A hash field, #LOG_FIELD#, in a property value string invokes a **Log Field** Behaviour ‘behind-the-scenes’ in Orbit.

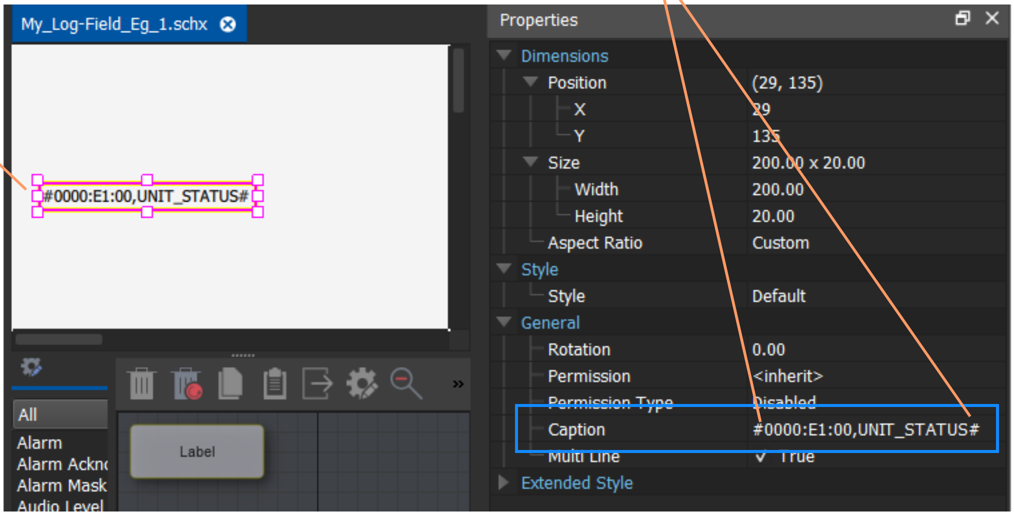
To use the Hash Field Syntax:

1 Add a **Label** to the screen.

2 Set the ‘Caption’ property to #0000:E1:00,UNIT_STATUS# , see Figure 8-4.

Hash field syntax, # #, used in caption property value

Label widget



Property	Value
Dimensions	
Position	(29, 135)
X	29
Y	135
Size	200.00 x 20.00
Width	200.00
Height	20.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	#0000:E1:00,UNIT_STATUS#
Multi Line	✓ True
Extended Style	

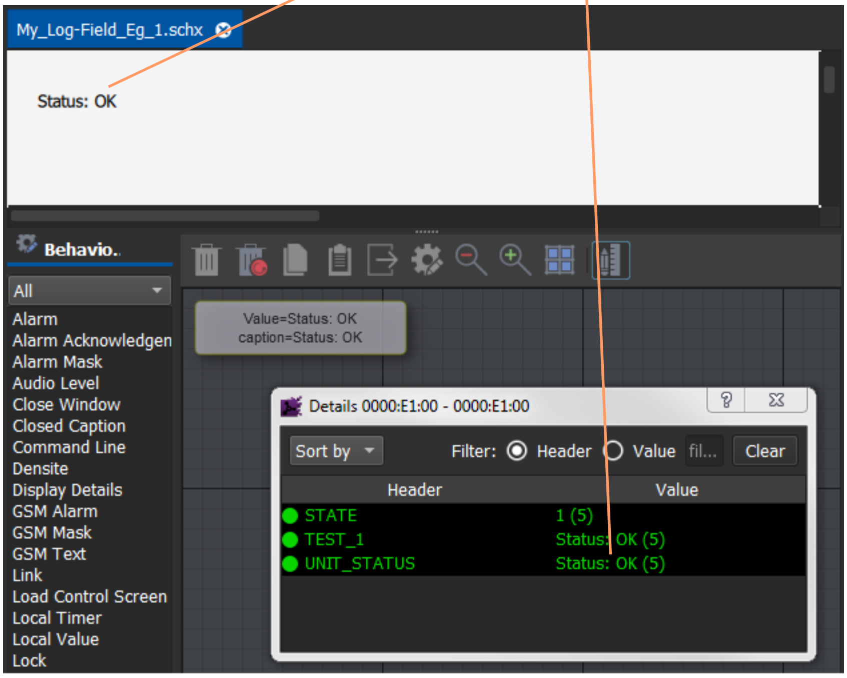
Fig. 8-4: Hash Field Syntax

3 Click **Save**.

Exercise Hash Field

1 Click the 'Test Behaviour/Bindings' icon ( ) to exercise the screen.

A device's UNIT_STATUS header value is shown on a label on-screen.



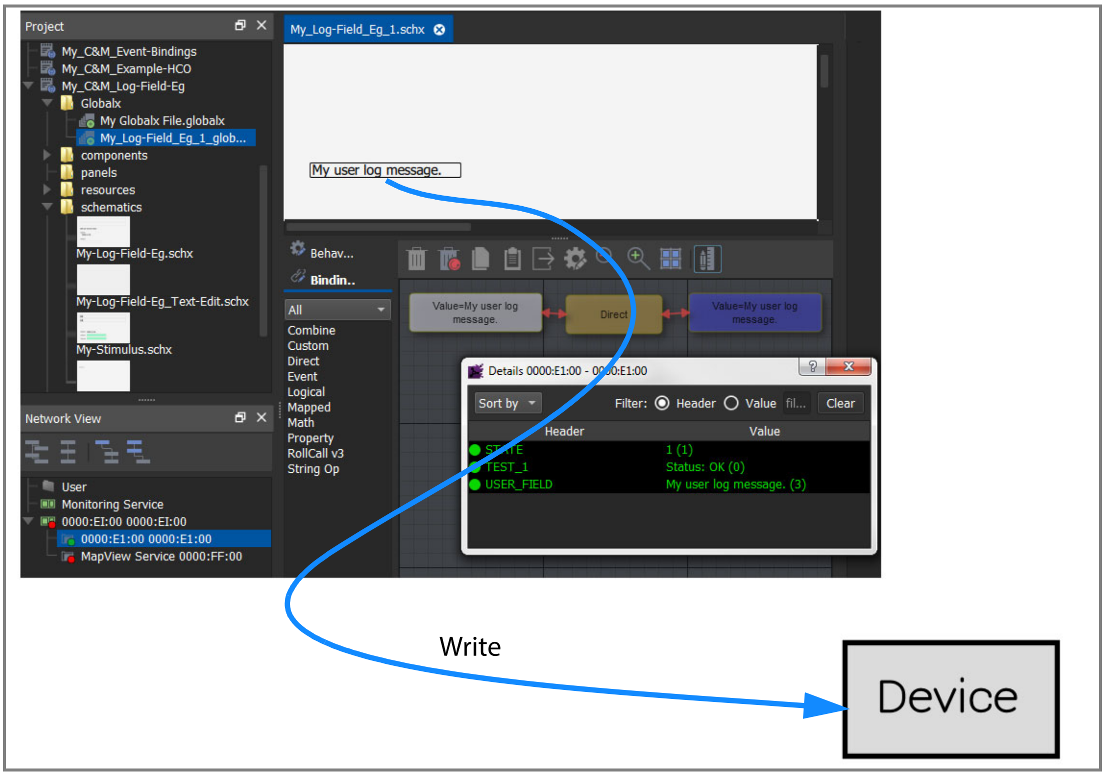
Value=Status: OK
caption=Status: OK

Header	Value
STATE	1 (5)
TEST_1	Status: OK (5)
UNIT_STATUS	Status: OK (5)

Fig. 8-5: Log Field Value Shown on Label and by Log Field Behaviour

2 Click the **Test** button again to stop exercising the screen.

Example - Write to a Log Field



Header	Value
STATE	1 (1)
TEST_1	Status: OK (0)
USER_FIELD	My user log message. (3)

This example writes to a **Log Field** header with Orbit MapView from a RollCall-enabled device with RollCall address ‘0000:E1:00’.

This example uses:

- Widgets - **Label, Text Edit**.
- Bindings - **Direct**.
- Behaviours - **Local Value**.

Log Field header values may also be written to from an Orbit MapView screen. Here, the example has a **Text Edit** widget added and connected up to a **Log Field** header value (USER_FIELD).

Write to a Log Field

The steps below follow on from Example - Read a Device’s Log Field, on page 230 above:

1 Add a **Text Edit** widget to the MapView screen.

> Note: A **Text Edit** widget can be configured to require the user to press Enter (i.e. the Return key) or not - by setting the ‘Requires Return Key’ property to true.)

2 Select the **Text Edit** widget and open the ‘Behaviour and Bindings’ graphical editor.

3 Add a **Log Field** Behaviour and a **Direct** Binding.

4 Configure the **Log Field** Behaviour:

- Name: ‘Log Field Write Beh’.
- RollCall Address: ‘0000:E1:00’.
- Headers: ‘USER_FIELD’.
- Write Mode: ‘Write Only’.

5 Configure the **Direct** Binding to use the ‘Log Field Write Beh’ Behaviour as source its Behaviour.

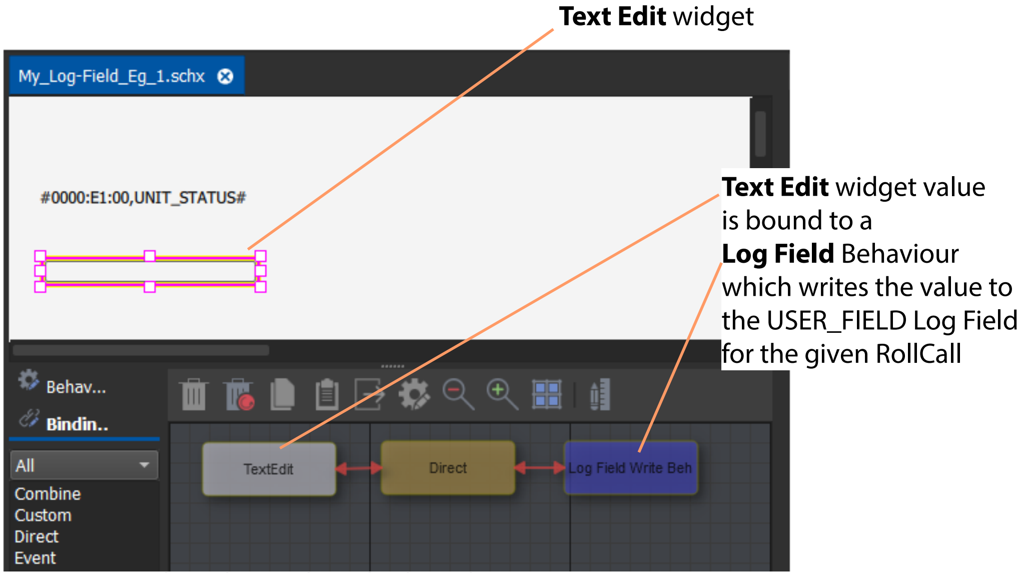


Fig. 8-6: Text Edit Widget and Local Value Behaviour

6 Click **Close**.

7 Click **Save**.

Exercise Writing to Log Field

1 Click the **Test** button ( ) to exercise the screen.

2 Enter text into the **Text Edit** widget and press Enter.

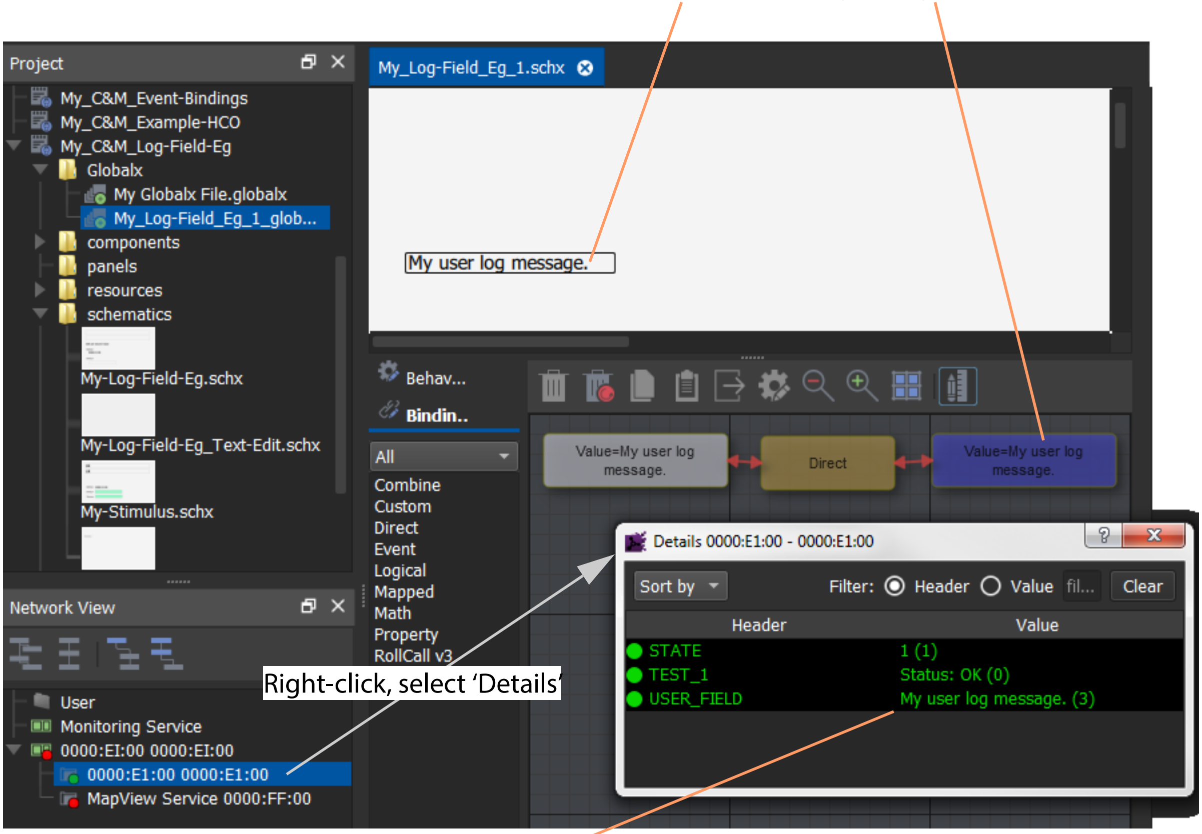
The MapView screen writes the text string to the USER_FIELD Log Field of the device at RollCall address 0000:E1:00 via the **Log Field** Behaviour

To see this, in the **Network View** pane:

3 Right-click on the device with RollCall address 0000:E1:00 and select ‘Details’ to show a Details window.

4 Figure 8-7 shows the entered text string.

User-entered text string in **Log Field** Behaviour



Header	Value
● STATE	1 (1)
● TEST_1	Status: OK (0)
● USER_FIELD	My user log message. (3)

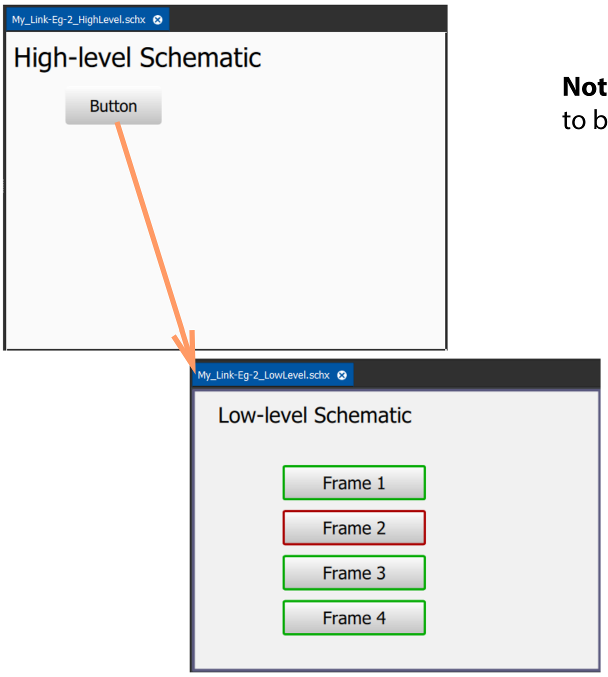
Right-click, select 'Details'

Text string in 'Details' window as a Log Field header value

Fig. 8-7: User-Entered Value Written to a Log Field Value

5 Click the **Test** Button again to stop exercising the screen.

Example - Linking from One Screen to Another



Note: the top-level screen must be set to be the **Home Screen.**

A link between two screens can be created by adding a **Link** Behaviour to a screen along with a Button widget. This allows the user to navigate between screens when the screen is running.

This example uses:

- Widgets - **Button**.
- Bindings - **Event**.
- Behaviours - **Link**.

Overall Screen Hierarchy and Home Screen

In this example, two isolated screens will be created and used. However, in practice, screens will be part of some overall screen hierarchy and, in this case, the top-level screen must be set as the home screen. This is done so that Orbit Services know where the top level of the screen hierarchy is. The services can then traverse the hierarchy from that point.

To set a top-level screen as the home screen:

- Open the top-level screen.
- Click **Project** > **Set xxx as Home** in the main menu.

Link Behaviour with Button Widget Example

1. Create two blank screens,
   one which shall be our low-level screen and one our high-level screen.
   ('My_Link-Eg-2_HighLevel' and 'My_Link-Eg-2_LowLevel').
2. Open the high level screen and add a **Button** widget.
3. Click **Project** > **Set xxx as Home** in the main menu.
   This sets the higher-level screen to be the home screen for this 2-screen example.
4. Select the **Button** widget and open the 'Behaviour and Bindings' graphic editor by clicking the **Edit Behaviours** button.
5. Add a **Link** Behaviour.

Automatically, an **Event** Binding is also added connecting the Link Behaviour to the clicking of the button. (For example, see Figure 8-8.)

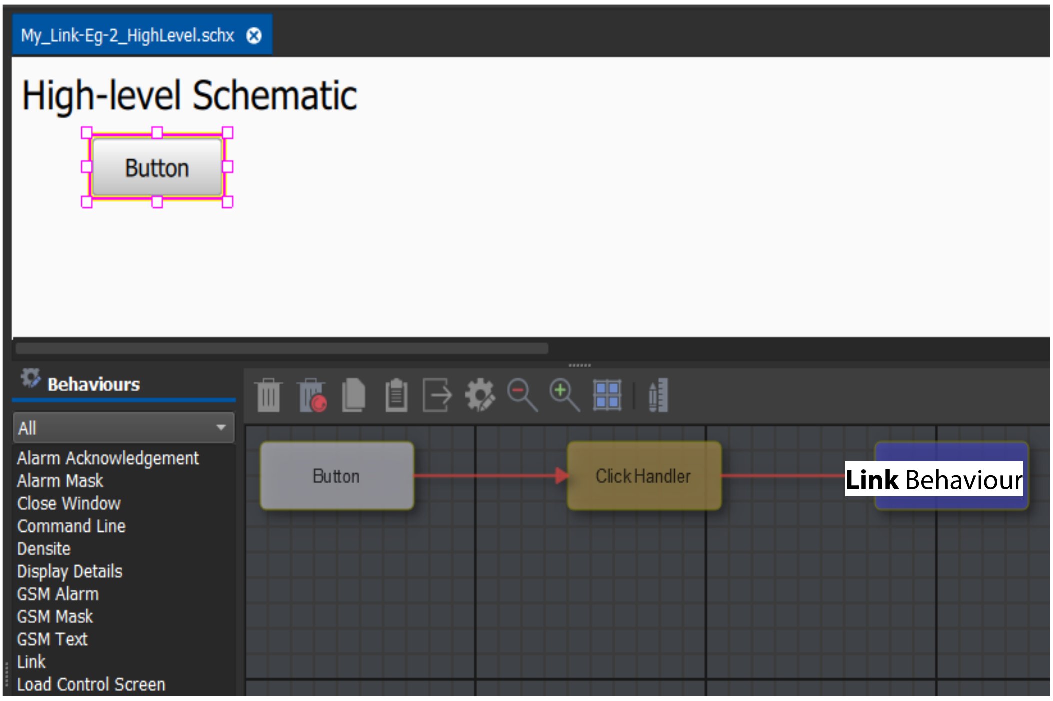
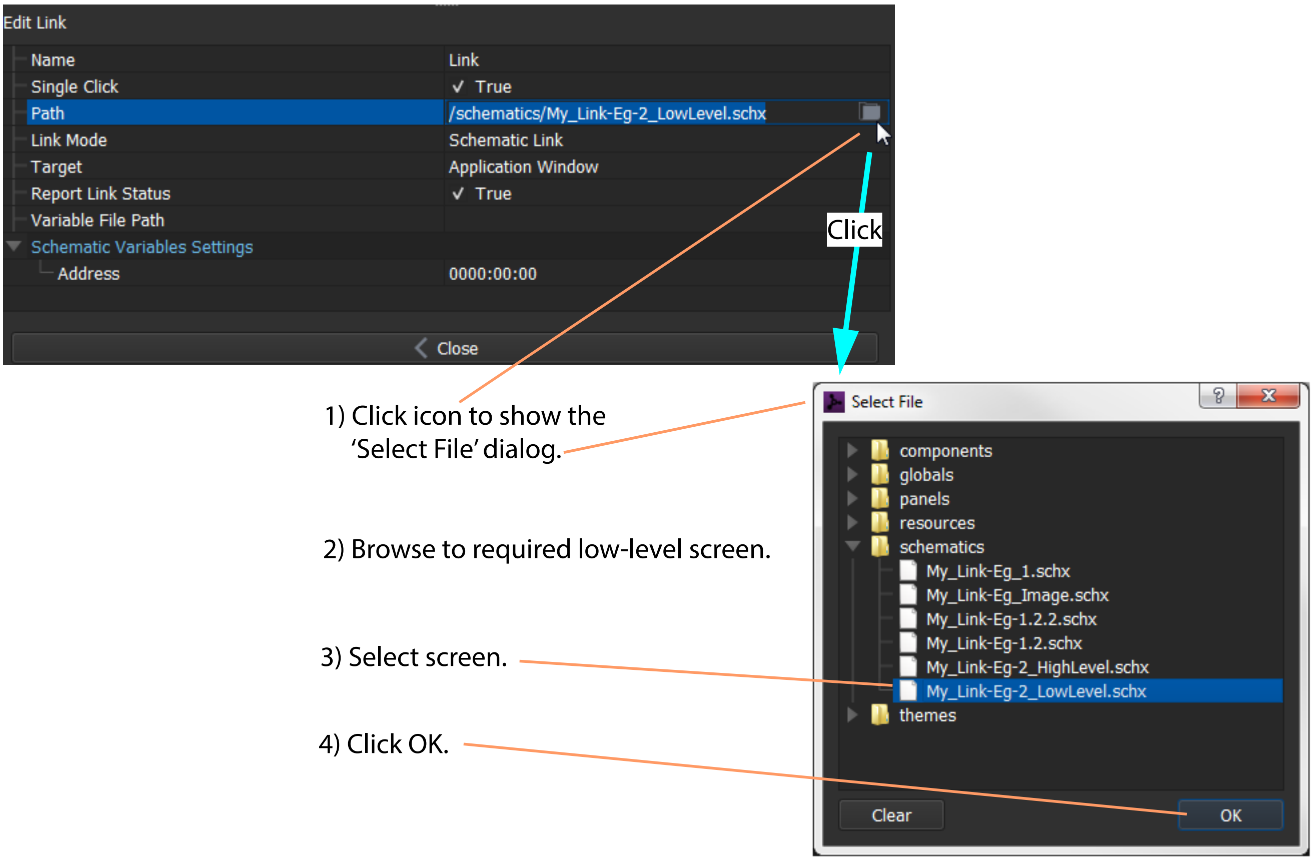


Fig. 8-8: High-level Screen

6 Edit the **Link** Behaviour properties
and set the **Path** property value to be the path to the low-level screen. See Figure 8-9.

And Figure 8-10 shows the final **Link** Behaviour properties.



Edit Link

Name	Link
Single Click	✓ True
Path	/schematics/My_Link-Eg-2_LowLevel.schx
Link Mode	Schematic Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	
Schematic Variables Settings	
Address	0000:00:00

1) Click icon to show the 'Select File' dialog.

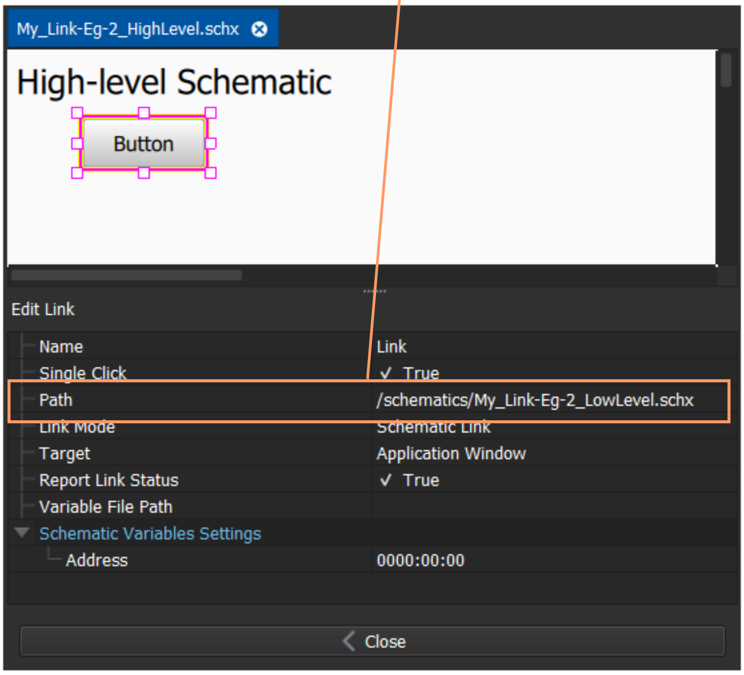
2) Browse to required low-level screen.

3) Select screen.

4) Click OK.

Fig. 8-9: Setting Path Property

Path set to the low-level screen in the Orbit MapView project.



Edit Link

Name	Link
Single Click	✓ True
Path	/schematics/My_Link-Eg-2_LowLevel.schx
Link Mode	Schematic Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	
Schematic Variables Settings	
Address	0000:00:00

Fig. 8-10: Link Behaviour Properties

7 Click **Close**.

8 Click **Save** to save the screen file.

9 Click **Save Project**.

Exercise the Link to a Screen

The Link example screen can now be exercised:

1 Open the high-level screen.

2 To start exercising the screen,
click the **Run Mode** icon in the main tool bar ( ).

Orbit enters 'Run Mode'. See Figure 8-11.

3 To exercise the **Link** functionality,
click the button on the Orbit MapView screen to open the low-level screen.
See Figure 8-11.

4 Click the **Back** icon (<) in the main tool bar to go back to the high-level screen.

To stop the exercising the screen:

5 Click the 'Run Mode' icon again.

Note: The icon is annotated 'Design Mode' when in 'Run Mode'.

And vice-versa.

1) Click **Run Mode** icon
to exercise the screen.

2) Click **Button** widget
to show the low-level screen.

3) Click **Back** icon
to go back to the high-level screen.

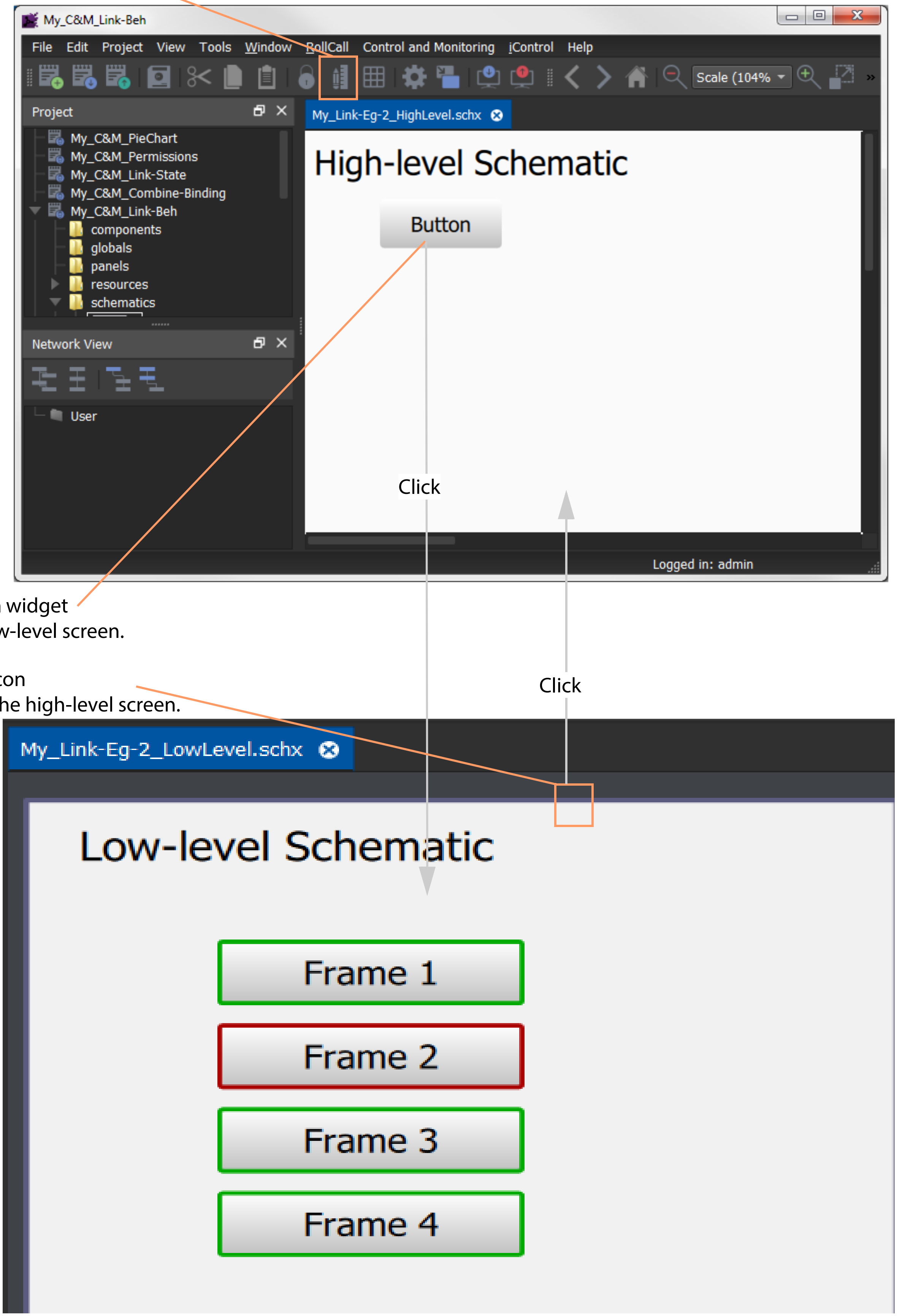


Fig. 8-11: Exercising the Link Behaviour

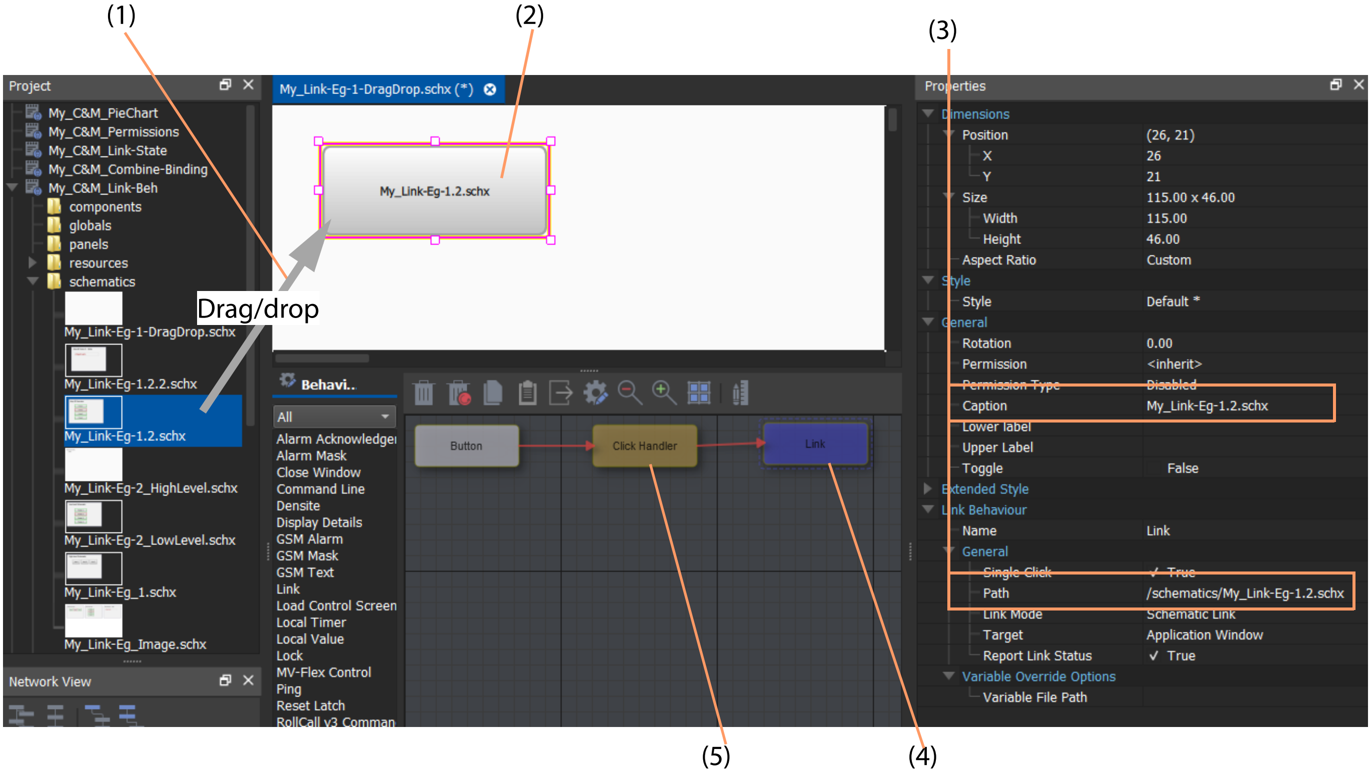
Drag-Drop Link Method

There is a quick way to add a **Link** Behaviour with a **Button** widget to link between two screens. A button can be created on a (higher-level) screen by drag-dropping a (lower-level) screen from the **Project View** in Orbit onto the (higher-level) screen open in a schematic editor. A **Link** Behaviour is automatically created and configured with a **Button** widget.

1 Drag-drop (lower-level) screen (see Figure 8-12) from the **Project View** pane (1) onto an open screen schematic.

This creates:

- a **Button** widget (2) with its 'Caption' and 'Path' properties already set (3);
- a **Link** Behaviour (4) bound to the Button with an **Event** Binding (5)



Dimensions
Position (26, 21)
X 26
Y 21
Size 115.00 x 46.00
Width 115.00
Height 46.00
Aspect Ratio Custom
Style
Style Default *
General
Rotation 0.00
Permission <inherit>
Permission Type Disabled
Caption My_Link-Eg-1.2.schx
Lower label
Upper Label
Toggle False
Extended Style
Link Behaviour
Name Link
General
Single Click ✓ True
Path /schematics/My_Link-Eg-1.2.schx
Link Mode Schematic Link
Target Application Window
Report Link Status ✓ True
Variable Override Options
Variable File Path

Fig. 8-12: Drag-Dropped Lower-Level Screen

Example - Screen Link States and Screen Re-Use with Variable Files

This example demonstrates **Link States** of an Orbit MapView project.

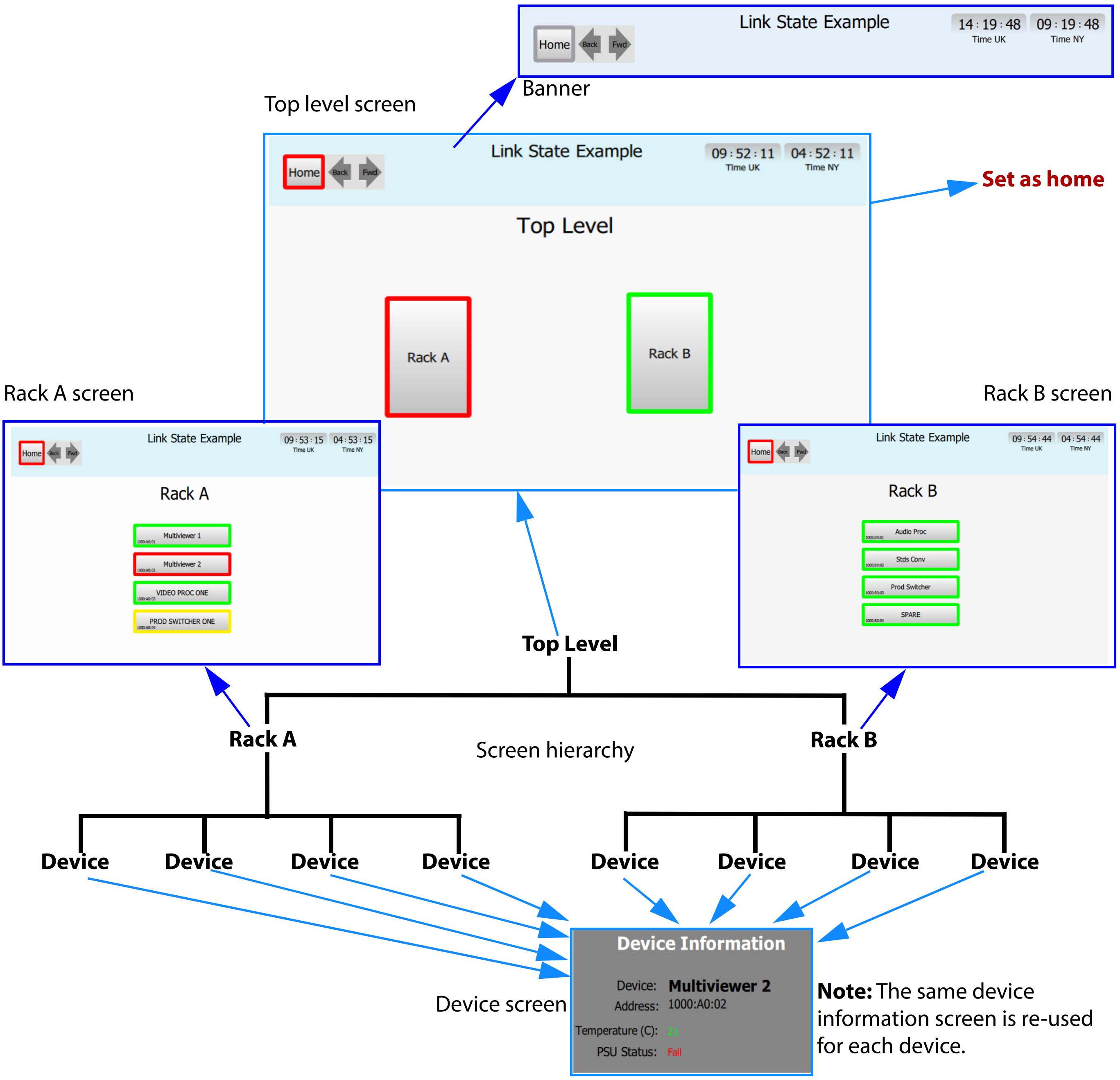
This example uses:

- Widgets - **Label, Button**, **Image Button**, **Digital Clock**.
- Bindings - **Direct**.
- Behaviours - **Link**.

And **Variable Files** are introduced, and Orbit Services are used.

Introduction

The example uses the Orbit Services (Orbit MapView service) to update **Link State** for a simple screen hierarchy (shown in Figure 8-13). Additionally, a **Variable File** is used to enable one screen to be re-used.



Note: The same device information screen is re-used for each device.

Fig. 8-13: Screen Hierarchical Tree Structure (Example Simple Orbit MapView Project)

Preliminaries

Assumptions:

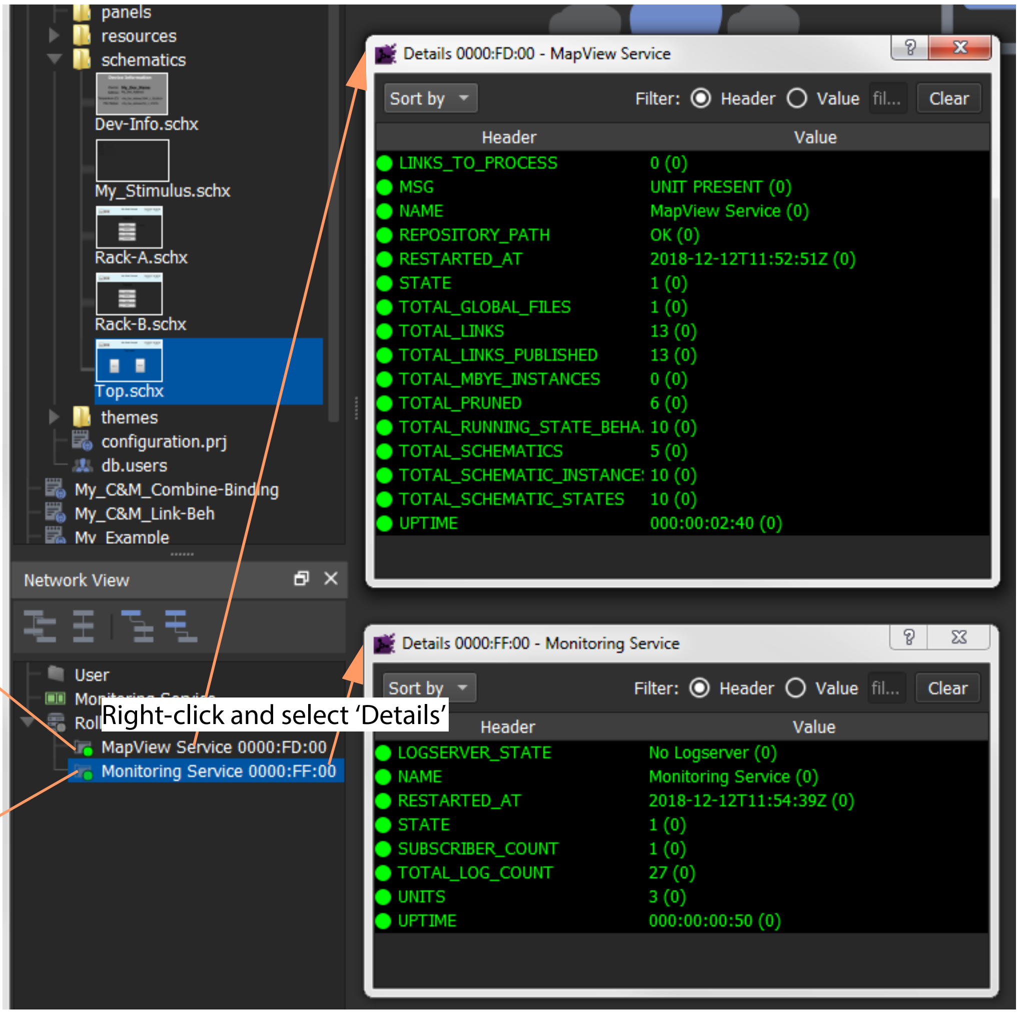
- The top-level screen is set as the 'Home' screen' ('Project > Set xxx as Home' in the main menu.)
- Orbit MapView service is running on a server,
  with RollCall address 0000:FD:00, domain 100.
- Orbit Monitoring service is running on a server,
  with RollCall address 0000:FF:00, domain 100.

  See Figure 8-14.

And:

- Orbit MapView set to domain 100.
- Device RollCall addresses used:
  - Rack A: 1000:A0:01 to 1000:A0:04.
  - Rack B: 1000:B0:01 to 1000:B0:04.

  See Figure 8-15.


panels
resources
schematics
Dev-Info.schx
My_Stimulus.schx
Rack-A.schx
Rack-B.schx
Top.schx
themes
configuration.prj
db.users
My_C&M_Combine-Binding
My_C&M_Link-Beh
My_Example
Network View
User
Monitoring Service
RollCall
MapView Service 0000:FD:00
Monitoring Service 0000:FF:00
Right-click and select 'Details'
Details 0000:FD:00 - MapView Service
Sort by
Filter: ● Header ○ Value fil... Clear
Details 0000:FF:00 - Monitoring Service
Sort by
Filter: ● Header ○ Value fil... Clear

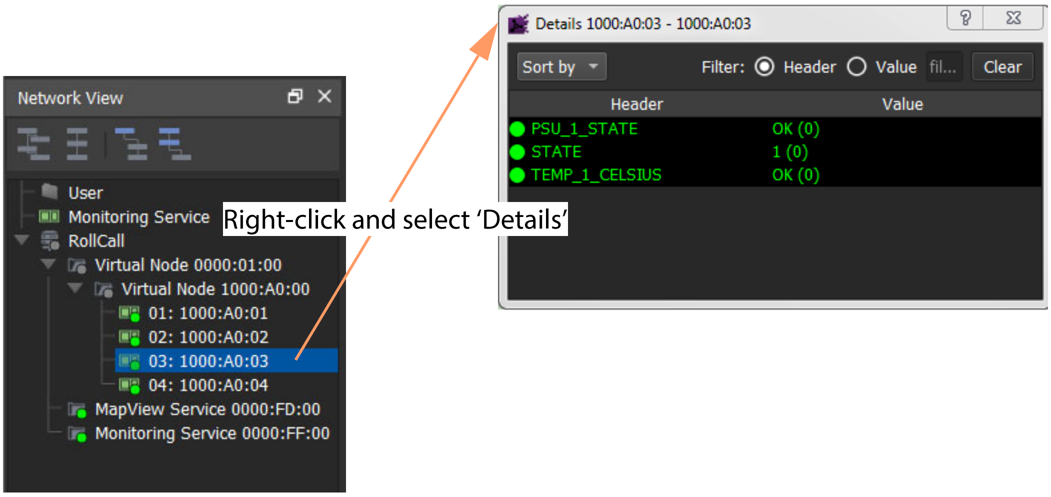
Header	Value
● LINKS_TO_PROCESS	0 (0)
● MSG	UNIT PRESENT (0)
● NAME	MapView Service (0)
● REPOSITORY_PATH	OK (0)
● RESTARTED_AT	2018-12-12T11:52:51Z (0)
● STATE	1 (0)
● TOTAL_GLOBAL_FILES	1 (0)
● TOTAL_LINKS	13 (0)
● TOTAL_LINKS_PUBLISHED	13 (0)
● TOTAL_MBYE_INSTANCES	0 (0)
● TOTAL_PRUNED	6 (0)
● TOTAL_RUNNING_STATE_BEHA.	10 (0)
● TOTAL_SCHEMATICS	5 (0)
● TOTAL_SCHEMATIC_INSTANCE:	10 (0)
● TOTAL_SCHEMATIC_STATES	10 (0)
● UPTIME	000:00:02:40 (0)

Header	Value
● LOGSERVER_STATE	No Logserver (0)
● NAME	Monitoring Service (0)
● RESTARTED_AT	2018-12-12T11:54:39Z (0)
● STATE	1 (0)
● SUBSCRIBER_COUNT	1 (0)
● TOTAL_LOG_COUNT	27 (0)
● UNITS	3 (0)
● UPTIME	000:00:00:50 (0)

MapView service at RollCall address 0000:FD:00

Monitoring service at RollCall address 0000:FF:00

Fig. 8-14: Network View Showing Orbit MapView Service and Orbit Monitoring Service Running



Header	Value
● PSU_1_STATE	OK (0)
● STATE	1 (0)
● TEMP_1_CELSIUS	OK (0)

Fig. 8-15: Some Devices Showing in Orbit Network View

Create the Project, a Top Level Screen and Set the Home Screen

1. Create a new control and monitoring Orbit project (C&M Project).
2. Create a new screen and leave it empty, for now. (Top.schx)
3. In the main menu,
   click '**Project > Set xxx as Home**' to set this screen to be the 'home' screen.
4. Click **Save**.
5. Click **Save Project**.

This is a (blank) top-level screen; it will be completed in a later sub-section below.

Build a Banner Component

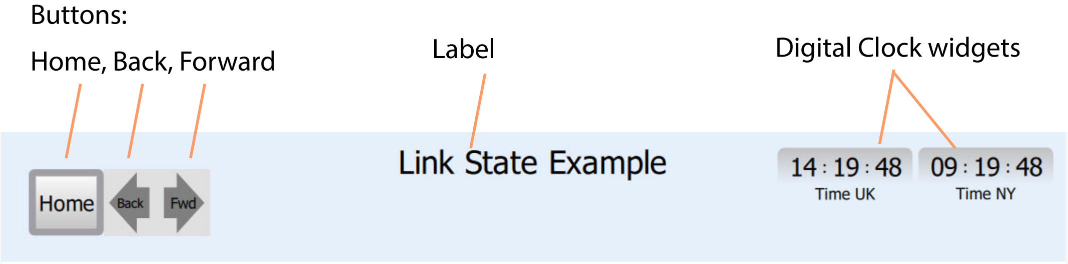


Fig. 8-16: Banner

To build the banner component:

1. Create a new component. (Banner.cptx)
   - Set the banner height and width in the **Properties** window.
   - Set background color.
2. Add a **Label** widget:
   - Static text, simply the title of our example, 'Link State Example'.
3. Add two **Digital Clock** widgets:
   - Configure these to show the time required.

4 Add three buttons comprising a **Button** widget and two **Image Button** widgets:

- 'Home' button - **Button** widget.
    - Set border width to 5.
    - Set caption to 'Home'.
    - Set **Report Link Status** to 'False'. (Omits button from 'Link State' processing.)
    - Add a **Link** Behaviour:
        - set Behaviour **Link Mode** property to 'schematic link'; and
        - set **Path** property to point at the top-level screen.
- 'Back' button - **Image Button** widget.
    - Set caption to 'Back'.
    - Set **Normal Image** path to a 'left-arrow' image, .
    - Add a **Link** Behaviour and set Behaviour **Link Mode** property to 'back link'.
- 'Forward' button - **Image Button** widget.
    - Set caption to 'Forward'.
    - Set **Normal Image** path to a 'right-arrow' image, .
    - Add a **Link** Behaviour and set Behaviour **Link Mode** property to 'forward link'.

5 Click **Save**.

Build a Device Information Screen (Low-Level)

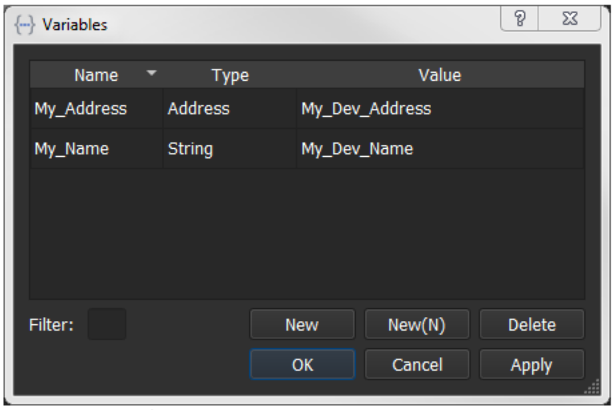
1 Create a new screen (Dev-Info.schx) and enter **Design Mode** in Orbit.

2 Right-click on the screen background and select 'Variables...'.

3 Add screen variables as shown in Figure 8-17a.

4 Add labels as shown in Figure 8-17b.

5 Click **Save**.

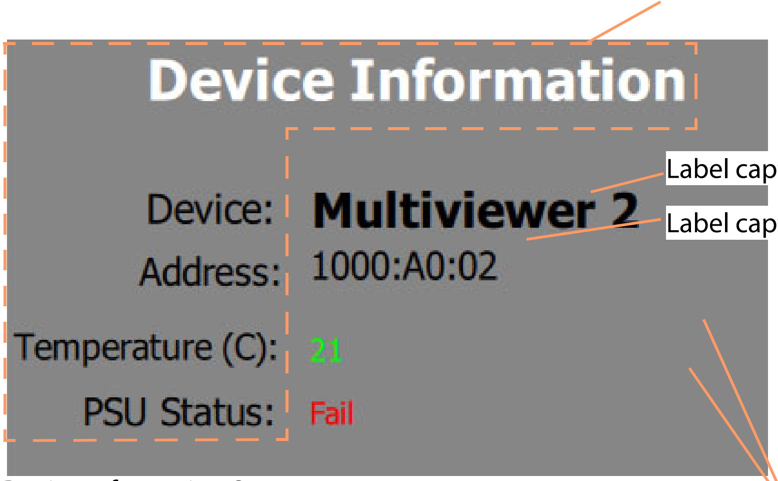


Variables

Name	Type	Value
My_Address	Address	My_Dev_Address
My_Name	String	My_Dev_Name

a) Screen Variables

Static label text annotation



Device Information

Device: Multiviewer 2

Address: 1000:A0:02

Temperature (C): 21

PSU Status: Fail

Label caption = {My_Name}

Label caption = {My_Address}

b) Device Information Screen

Label caption= <{My_Address},LOG_HEADER_NAME>

Fig. 8-17: Device Information Screen

Build Two Rack Screens (Mid-level)

Rack A Screen

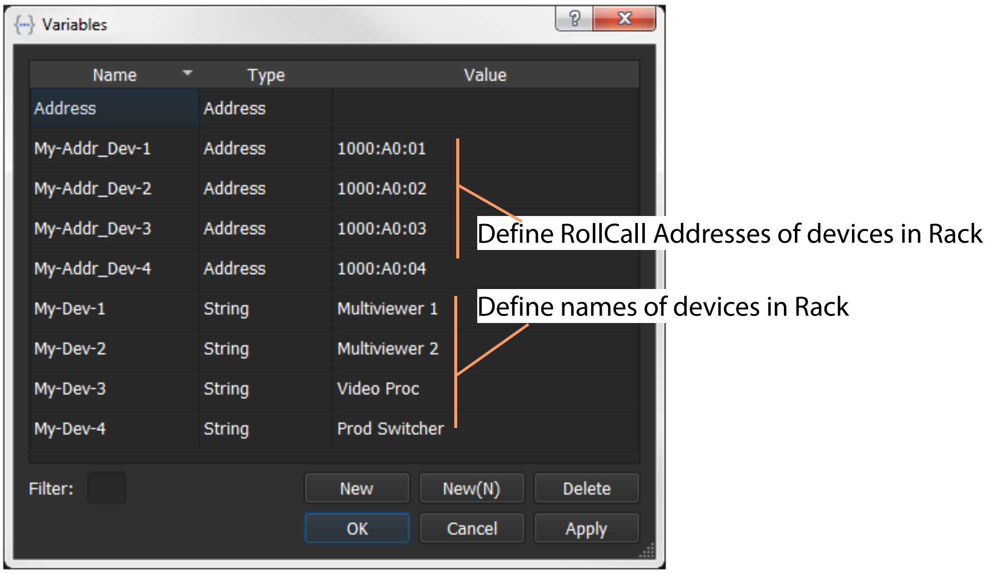
1 Create a new screen and open it in **Design Mode** in Orbit.

2 Add the 'Banner' component (built earlier) along the top.

3 Right-click on the screen background and select 'Variables...'.

4 Enter screen variables as shown in Figure 8-18a.

5 Click **OK**.

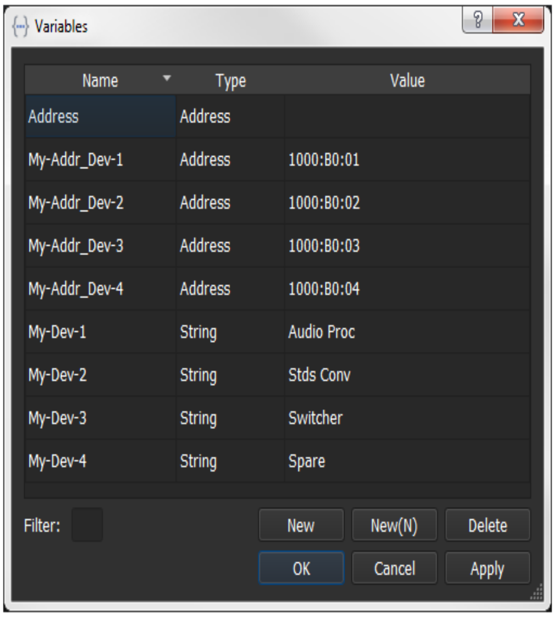


Name	Type	Value
Address	Address	
My-Addr_Dev-1	Address	1000:A0:01
My-Addr_Dev-2	Address	1000:A0:02
My-Addr_Dev-3	Address	1000:A0:03
My-Addr_Dev-4	Address	1000:A0:04
My-Dev-1	String	Multiviewer 1
My-Dev-2	String	Multiviewer 2
My-Dev-3	String	Video Proc
My-Dev-4	String	Prod Switcher

Define RollCall Addresses of devices in Rack

Define names of devices in Rack

a) Rack A screen variables



Name	Type	Value
Address	Address	
My-Addr_Dev-1	Address	1000:B0:01
My-Addr_Dev-2	Address	1000:B0:02
My-Addr_Dev-3	Address	1000:B0:03
My-Addr_Dev-4	Address	1000:B0:04
My-Dev-1	String	Audio Proc
My-Dev-2	String	Stds Conv
My-Dev-3	String	Switcher
My-Dev-4	String	Spare

b) Rack B screen variables

Fig. 8-18: Screen Variables:
a) Rack A.
b) Rack B.

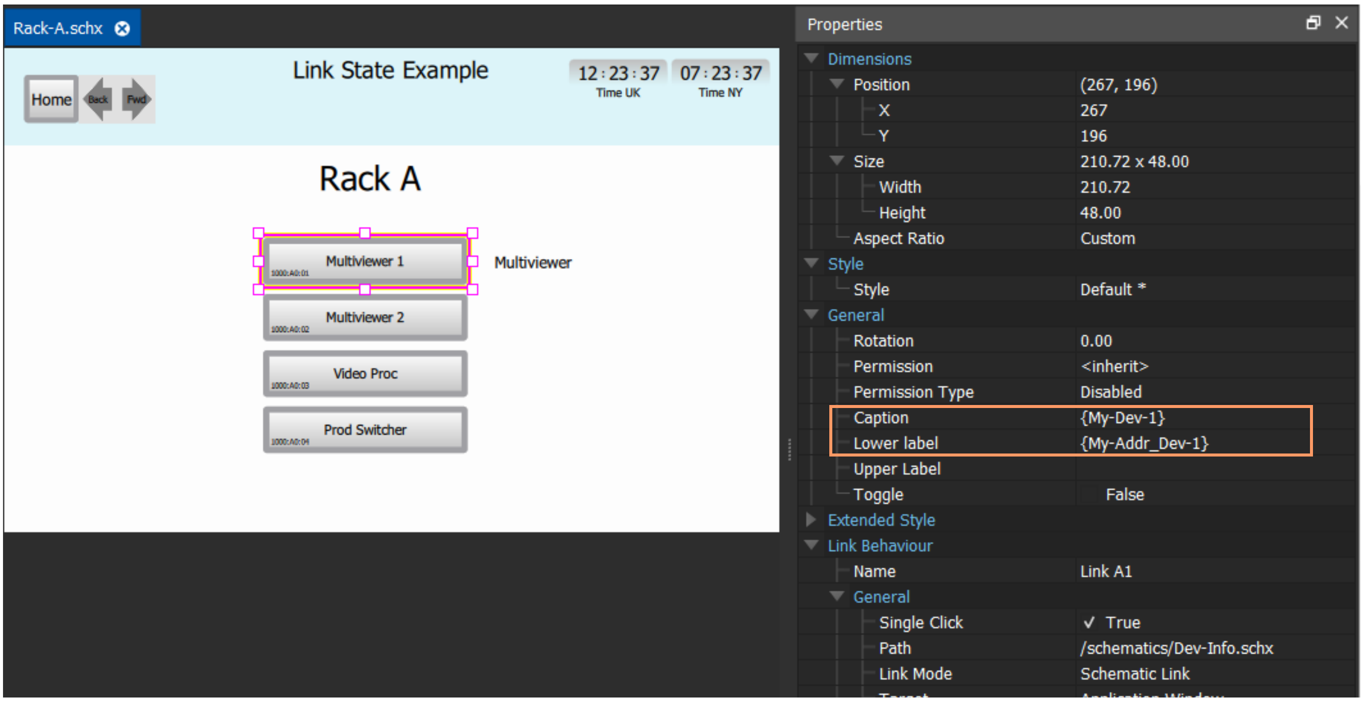
Note: The screen variables defined here provide a default RollCall address and name of each device in the equipment rack covered by this screen.

6 Add four **Button** widgets.

For each **Button** widget:

7 Set the Button’s **Caption** property to be the screen variable for the device name. See Figure 8-19.

8 Set the Button’s **Lower label** property to be the screen variable for the device RollCall address. See Figure 8-19.



Dimensions	
Position	(267, 196)
X	267
Y	196
Size	210.72 x 48.00
Width	210.72
Height	48.00
Aspect Ratio	Custom
Style	
Style	Default *
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Caption	{My-Dev-1}
Lower label	{My-Addr_Dev-1}
Upper Label	
Toggle	False
Extended Style	
Link Behaviour	
Name	Link A1
General	
Single Click	✓ True
Path	/schematics/Dev-Info.schx
Link Mode	Schematic Link
Target	Application Window

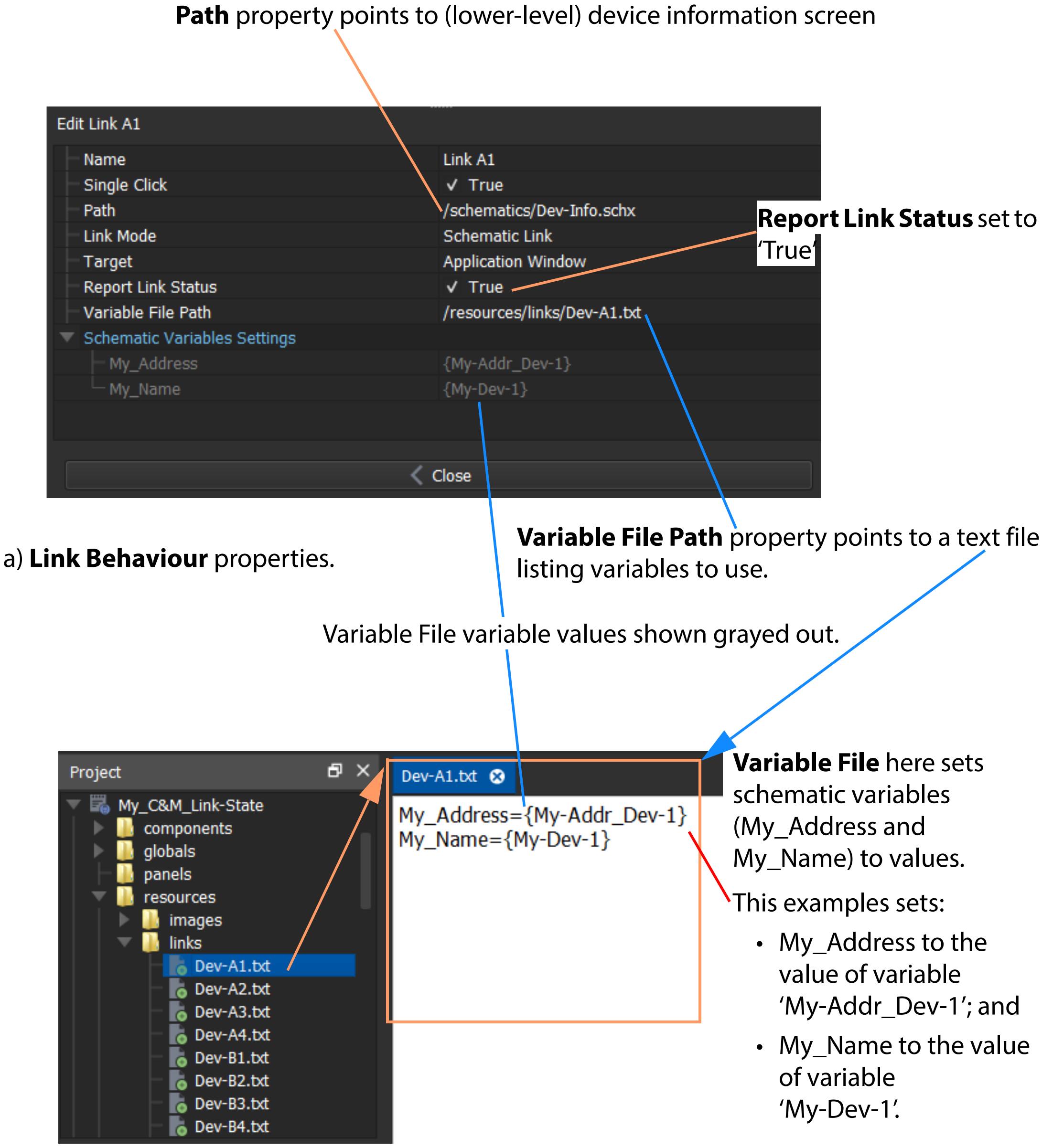
Fig. 8-19: Button Configuration

9 Add a **Link** Behaviour and configure as shown in Figure 8-20a:

- Configure **Path** to point at the device information screen.
- Set **Report Link Status** to ‘True’.
(The link state of the button will contribute to the overall state of the screen.)
- Configure **Variable File Path** to point at a text file listing the screen variable values to use in the target screen (via this **Link** Behaviour).

  Each **Variable File** used must be a text file containing the RollCall address and name of a device in the rack. (See Figure 8-20b for an example variable file.)

10 Repeat for other **Button** widgets.



Path property points to (lower-level) device information screen

Edit Link A1

Name	Link A1
Single Click	✓ True
Path	/schematics/Dev-Info.schx
Link Mode	Schematic Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	/resources/links/Dev-A1.txt
Schematic Variables Settings	
My_Address	{My-Addr_Dev-1}
My_Name	{My-Dev-1}

Report Link Status set to 'True'

a) **Link Behaviour** properties.

Variable File Path property points to a text file listing variables to use.

Variable File variable values shown grayed out.

```
My_Address={My-Addr_Dev-1}
My_Name={My-Dev-1}
```

Variable File here sets schematic variables (My_Address and My_Name) to values.

This examples sets:

- My_Address to the value of variable 'My-Addr_Dev-1'; and
- My_Name to the value of variable 'My-Dev-1'.

b) Example **Variable File**

Fig. 8-20: Link Behaviour Configuration:
a) Properties.
b) Example Variable File.

11 Click **Save**.

This has created a screen for one device rack, Rack A.

Rack B Screen

To create the second rack (Rack B) screen:

1 Create a new screen similar to Rack A, and using variables in Figure 8-18b.

2 Click **Save**.

This has created two rack screens, see Figure 8-21.

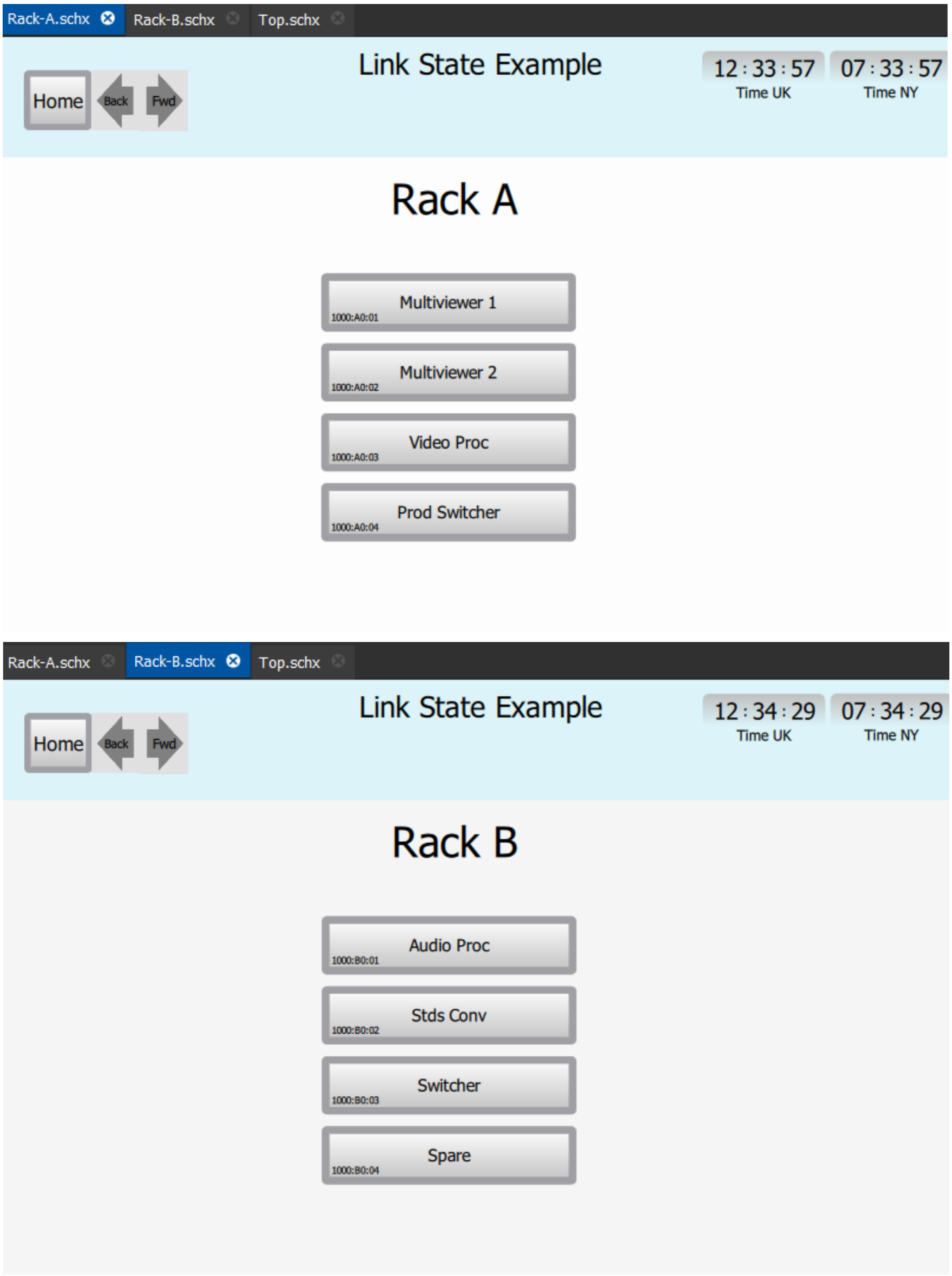


Fig. 8-21: Rack A and Rack B Screens - Showing Device Names from the Respective Rack Screens

Build the Top Level Screen

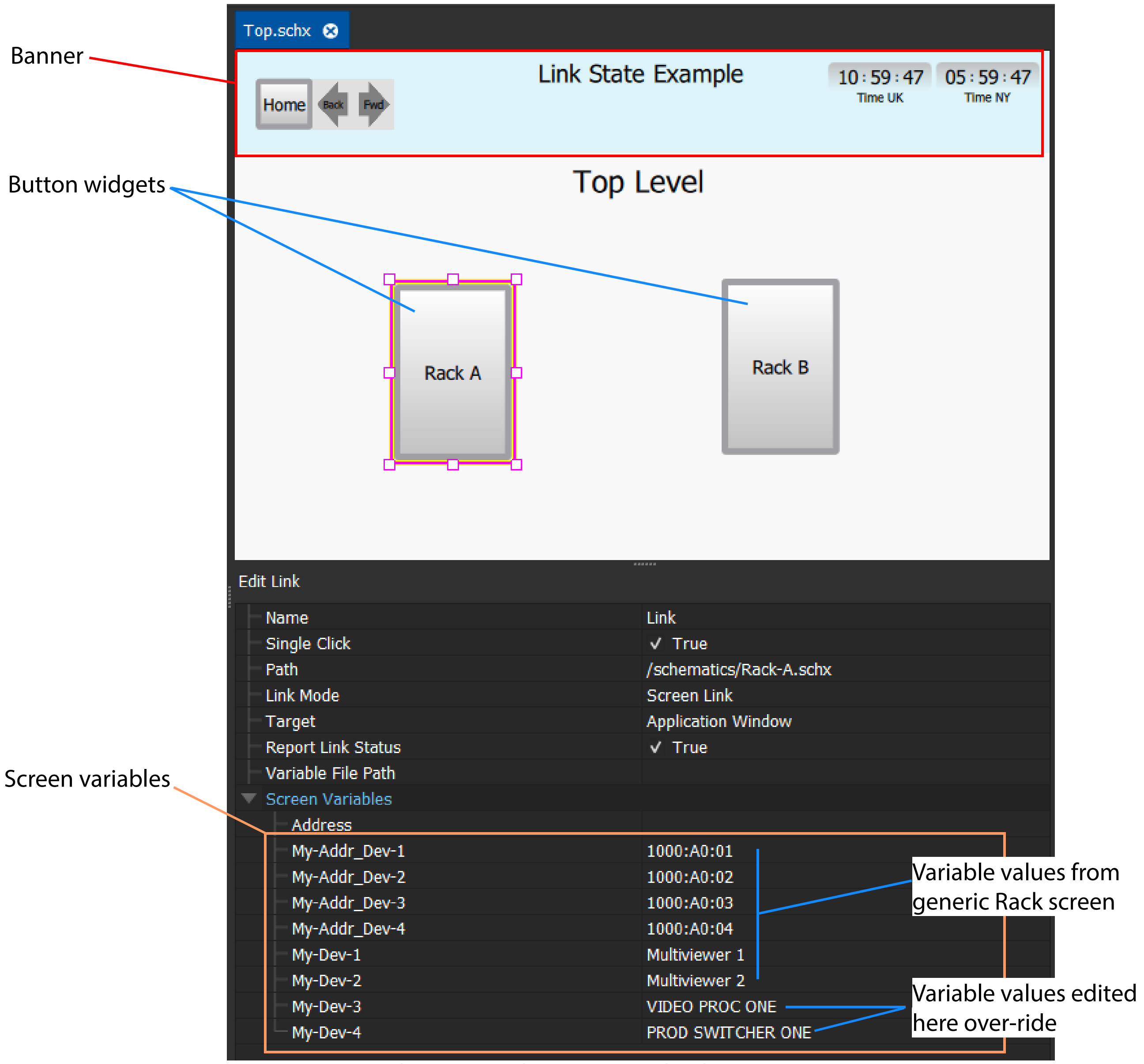
1. Open the top-level screen (created earlier) in **Design Mode**.
2. Add the 'Banner' component (built earlier) along the top.
3. Add two **Button** widgets.
   One button will be configured to link to the Rack A screen, the other to the Rack B screen.

Screen Variables for Rack A Over-ridden at Link Behaviour

For Rack A's **Button** widget:

1. Add a **Link** Behaviour and configure:
   - **Path** property to point at to rack screen A.
   - **Report Link Status** set to 'True'.

   The screen variables as shown come from the Rack A screen itself. These **Link** Behaviour settings may be edited (and over-ridden) here. See Figure 8-22.
2. Click **Close**.



Banner

Button widgets

Screen variables

Variable values from generic Rack screen

Variable values edited here over-ride

Edit Link

Name	Link
Single Click	✓ True
Path	/schematics/Rack-A.schx
Link Mode	Screen Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	
Screen Variables	
Address	
My-Addr_Dev-1	1000:A0:01
My-Addr_Dev-2	1000:A0:02
My-Addr_Dev-3	1000:A0:03
My-Addr_Dev-4	1000:A0:04
My-Dev-1	Multiviewer 1
My-Dev-2	Multiviewer 2
My-Dev-3	VIDEO PROC ONE
My-Dev-4	PROD SWITCHER ONE

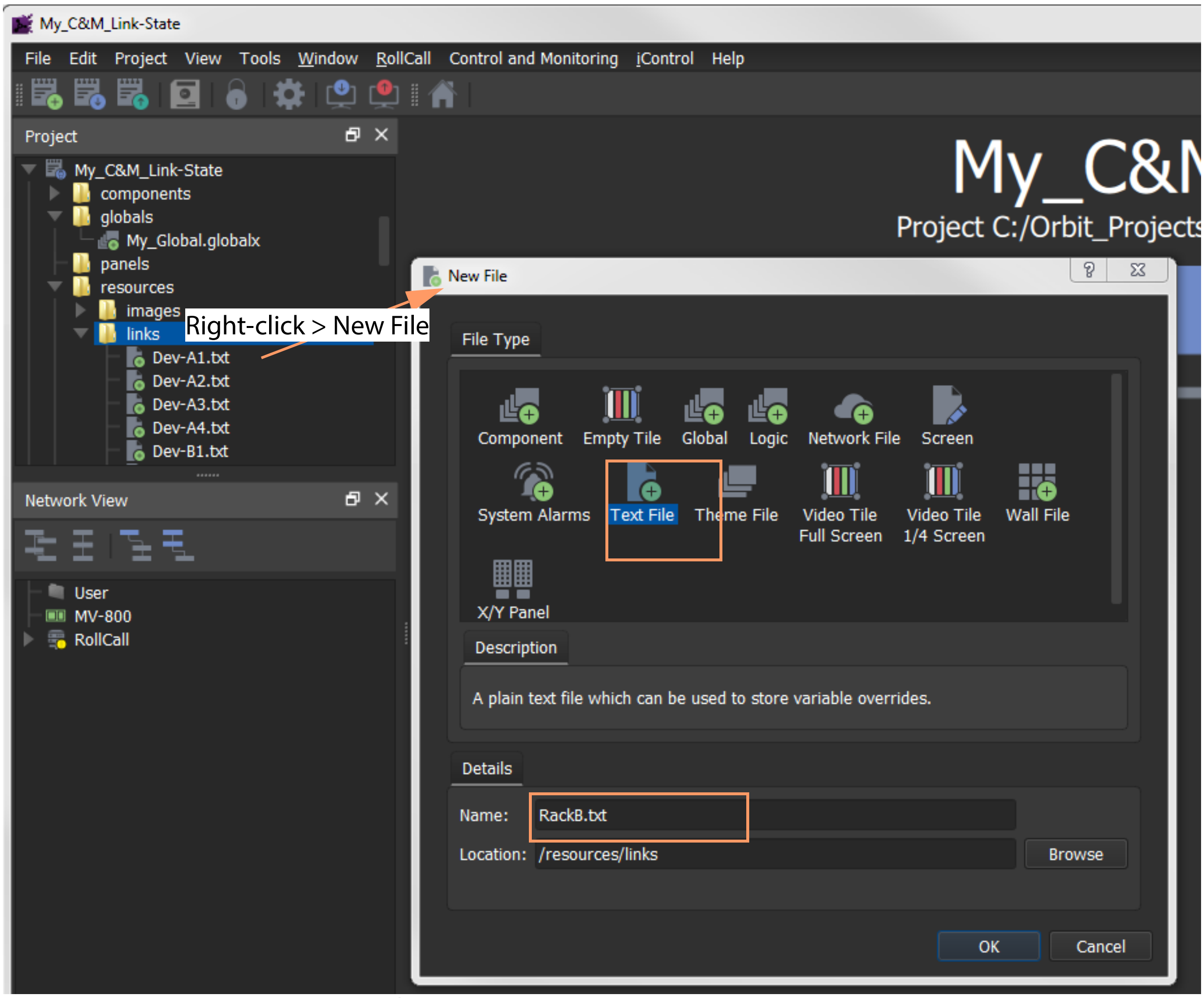
Fig. 8-22: Top Level Screen - Rack A Button's Link Behaviour Settings

Schematic Variables for Rack B Defined in a New Text File

For Rack B, a new text file will be used to define the schematic variables used on the Rack B screen, over-riding those on the generic Rack screen:

1 Create a new text file from the Project View.
See Figure 8-23.

Click **OK**.



Right-click > New File

A plain text file which can be used to store variable overrides.

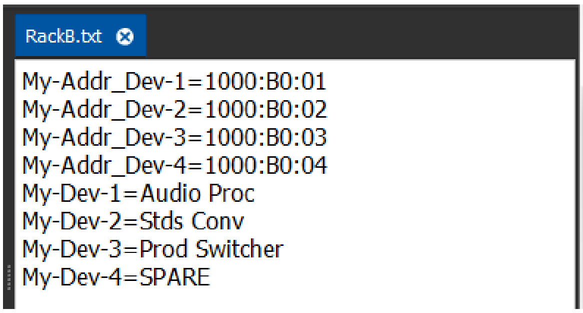
Name: RackB.txt

Location: /resources/links

Fig. 8-23: New Text File

2 Open the new text file. It opens in a text editor.
And enter the required schematic variable values. See Figure 8-24.

3 Click **Save**.



```
My-Addr_Dev-1=1000:B0:01
My-Addr_Dev-2=1000:B0:02
My-Addr_Dev-3=1000:B0:03
My-Addr_Dev-4=1000:B0:04
My-Dev-1=Audio Proc
My-Dev-2=Stds Conv
My-Dev-3=Prod Switcher
My-Dev-4=SPARE
```

Fig. 8-24: Edit Text File

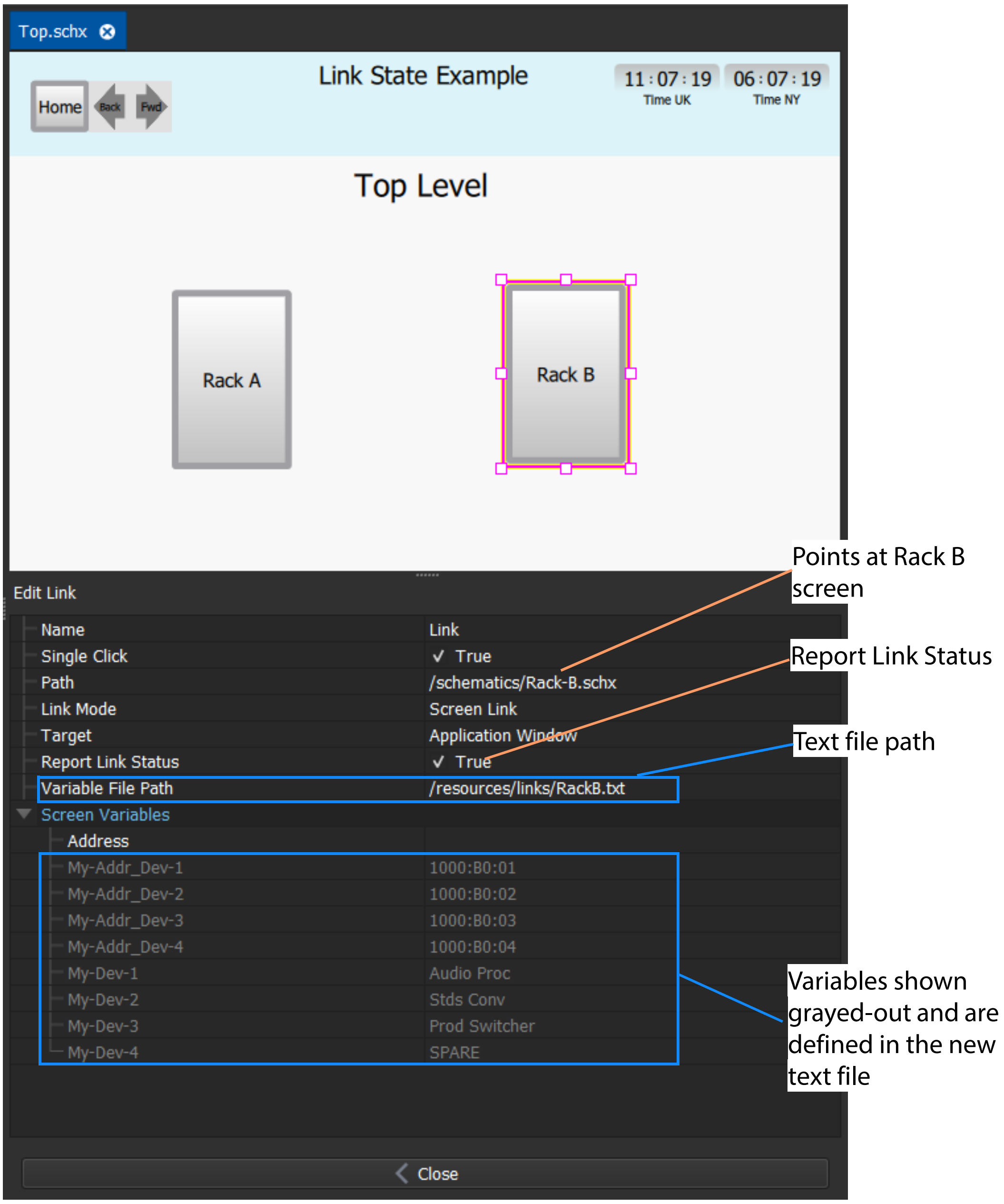
4 Open the top level screen.

For rack B's **Button** widget:

5 Add a **Link** Behaviour and configure:

- **Path** property to point at the rack screen B.
- **Report Link Status** set to 'True'.
- **Variable File Path** set to the project path to the Rack B text file containing variable values.

See Figure 8-22.



Name	Link
Single Click	✓ True
Path	/schematics/Rack-B.schx
Link Mode	Screen Link
Target	Application Window
Report Link Status	✓ True
Variable File Path	/resources/links/RackB.txt
Screen Variables	
Address	
My-Addr_Dev-1	1000:B0:01
My-Addr_Dev-2	1000:B0:02
My-Addr_Dev-3	1000:B0:03
My-Addr_Dev-4	1000:B0:04
My-Dev-1	Audio Proc
My-Dev-2	Stds Conv
My-Dev-3	Prod Switcher
My-Dev-4	SPARE

Points at Rack B screen

Report Link Status

Text file path

Variables shown grayed-out and are defined in the new text file

Fig. 8-25: Top Level Screen - Rack B Button's Link Behaviour Settings

6 Click **Close**.

7 Click **Save**.

8 Click **Save Project**.

Restart Orbit MapView Service

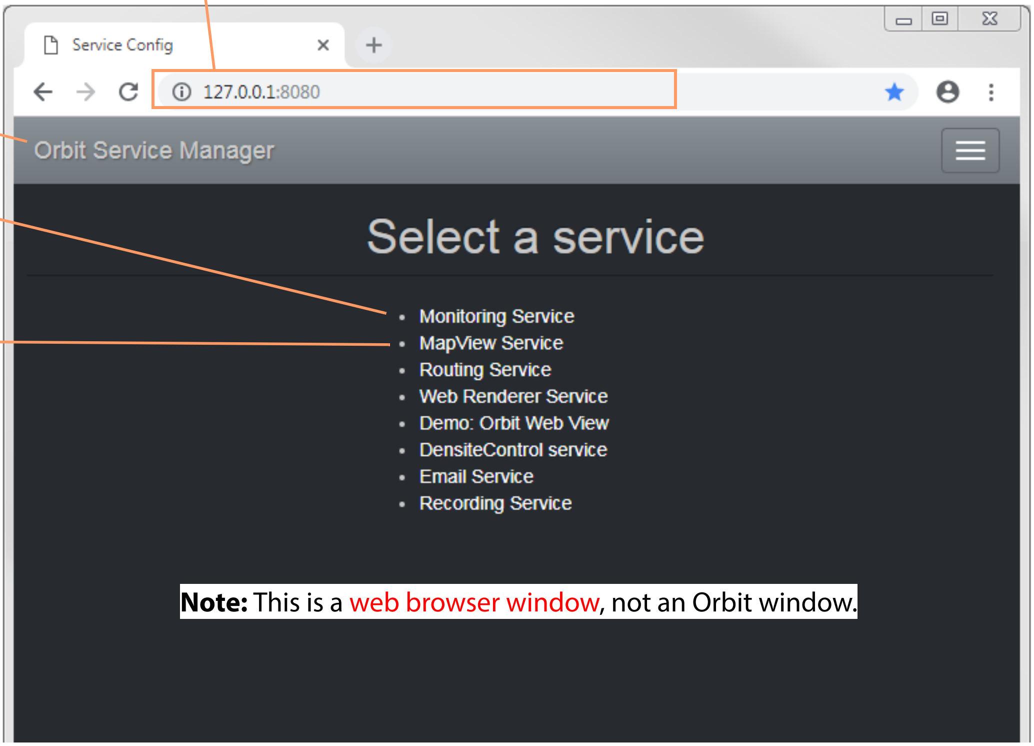
The Orbit Monitoring and Orbit MapView services should be running, as outlined in Preliminaries, on page 244.

1 Open the Orbit Service Manager web page in a web browser and click on 'MapView Service'. See Figure 8-26.
The MapView Service page is shown. See Figure 8-27.

Browser window

IP address and IP port of the installed Orbit Services.

Note: Orbit Services are typically installed on a server PC and the IP address and IP port number entered into the browser are for the services installed on the server PC.
The address shown in this figure is for Orbit Services installed on the same PC that is running the Orbit Client.



Note: This is a web browser window, not an Orbit window.

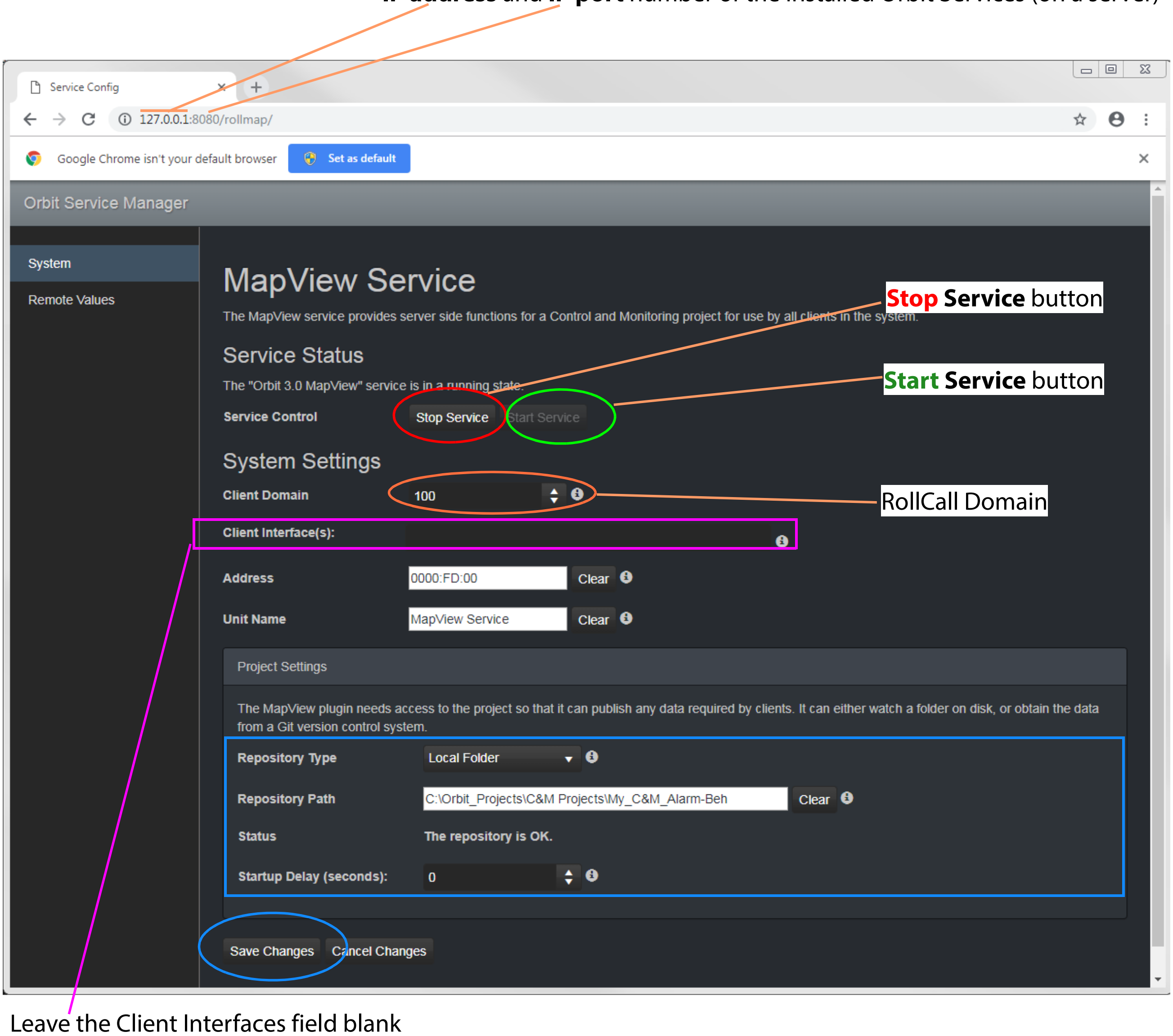
Click heading to view this page

Click to view the Monitoring Service page

Click to view the MapView Service page

Fig. 8-26: Orbit Service Manager Web Page

IP address and **IP port** number of the installed Orbit Services (on a server)



MapView Service

The MapView service provides server side functions for a Control and Monitoring project for use by all clients in the system.

Service Status

The "Orbit 3.0 MapView" service is in a running state.

Service Control	Stop Service, Start Service

System Settings

Client Domain	100
Client Interface(s):	
Address	0000:FD:00
Unit Name	MapView Service

Project Settings

The MapView plugin needs access to the project so that it can publish any data required by clients. It can either watch a folder on disk, or obtain the data from a Git version control system.

Repository Type	Local Folder
Repository Path	C:\Orbit_Projects\C&M Projects\My_C&M_Alarm-Beh
Status	The repository is OK.
Startup Delay (seconds):	0

Stop Service button

Start Service button

RollCall Domain

Leave the Client Interfaces field blank
- an empty Client Interfaces list will use all available interfaces.

Fig. 8-27: Orbit MapView Service Web Page

2 To stop the service, click the **Stop Service** button.

3 Point the Orbit MapView service at the latest version of the Orbit MapView project, by setting **Repository Type** and **Repository Path**.

4 Click **Save Changes**.

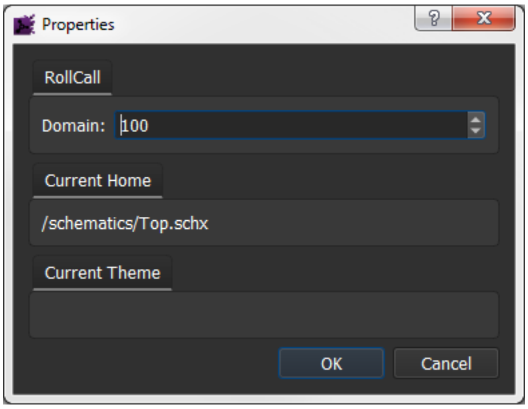
5 Click **Start Service** to restart the service

Quick Check of Orbit MapView Project

Finally, check that the Orbit MapView project is using the same RollCall domain and that its top-level screen is set as the home screen.
With Orbit running on the client computer and with the Orbit MapView project open:

1 In the Orbit main menu, click ‘Control and Monitoring > Properties’.

2 A properties dialog is shown. See Figure 8-28.



Properties

RollCall

Domain: 100

Current Home

/schematics/Top.schx

Current Theme

Fig. 8-28: Properties Dialog

3 Check that the displayed RollCall domain number is the same as used by the Orbit Services.

4 Check that the top level screen is set as the current home screen (**Project > Set XXX as Home** in main menu.).

IMPORTANT
Top Level Screen as Home Screen
The top-level screen must be set as the home screen. This is done so that Orbit Services know where the top level of the screen hierarchy is. The services can then traverse the hierarchy from that point.

To set a top-level screen as the home screen:

- Open the top-level screen.
- Click **Project > Set xxx as Home** in the main menu.

Exercise the Example

Run the Orbit MapView Project

1. Open the Orbit MapView project.
2. Open the top-level screen and enter **Run Mode**.

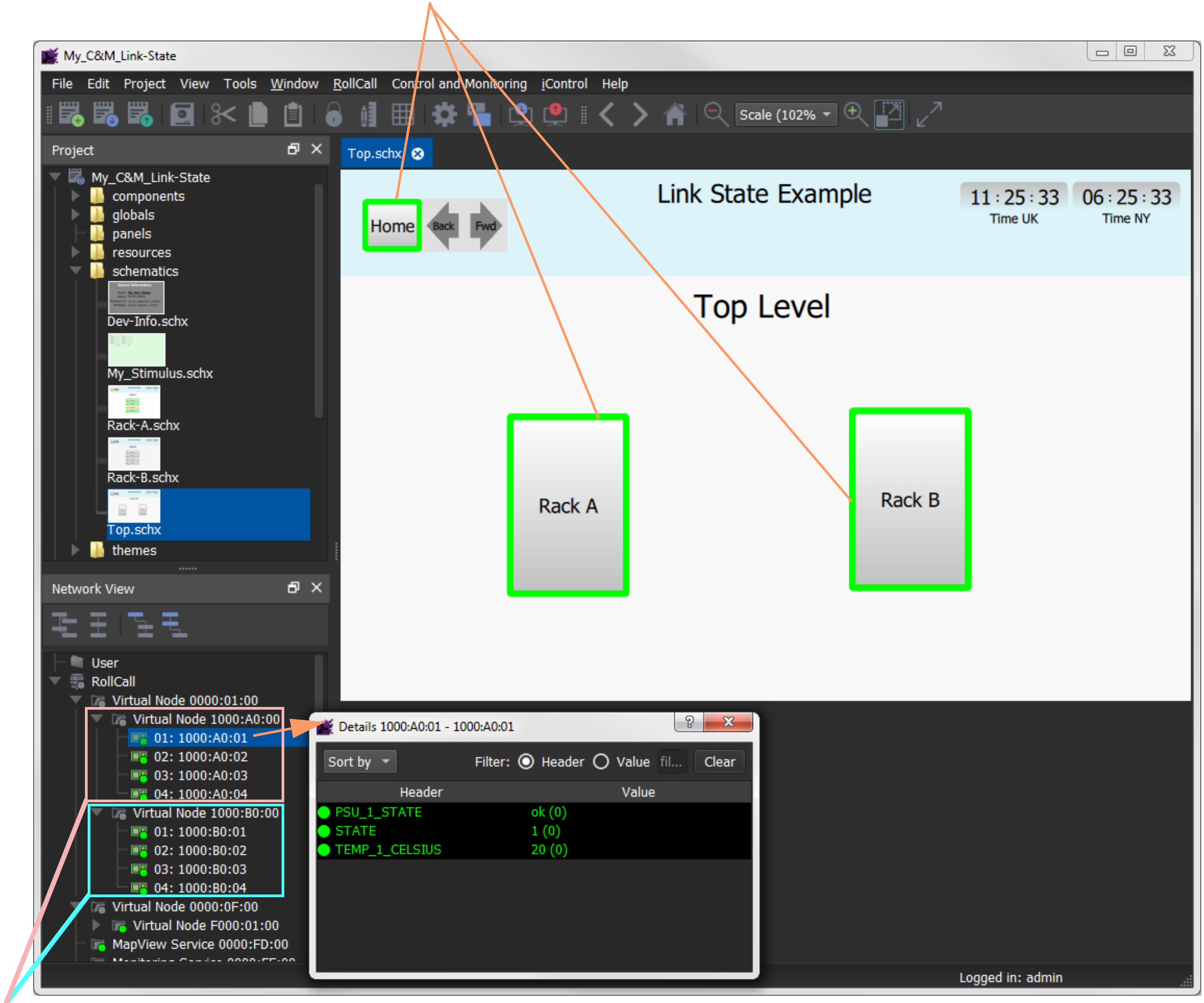
The following is happening:

- The Orbit MapView project is:
  - running on the client computer;
  - monitoring eight devices (four in rack A and four in rack B);
  - reporting the state of devices on buttons in the MapView screen hierarchy; and
  - indicating the state of devices in the **Network View**.

- The Orbit MapView Service is:
  - running on a server computer;
  - executing the same Orbit MapView project in parallel to the client computer; and
  - providing updated MapView project ‘Link State’ information to the client.

See Figure 8-29.

Button border color indicates the state of the lower-level screen



Header	Value
● PSU_1_STATE	ok (0)
● STATE	1 (0)
● TEMP_1_CELSIUS	20 (0)

System devices are shown in the **Network View** grouped under 'Virtual Node' names according to device RollCall address.

Fig. 8-29: Top-Level Screen Running with All Devices 'OK'

If the state of monitored devices changes, and hence the state of lower-level screens changes, then this is reflected in the running Orbit MapView project. See Figure 8-30 to Figure 8-32 below.

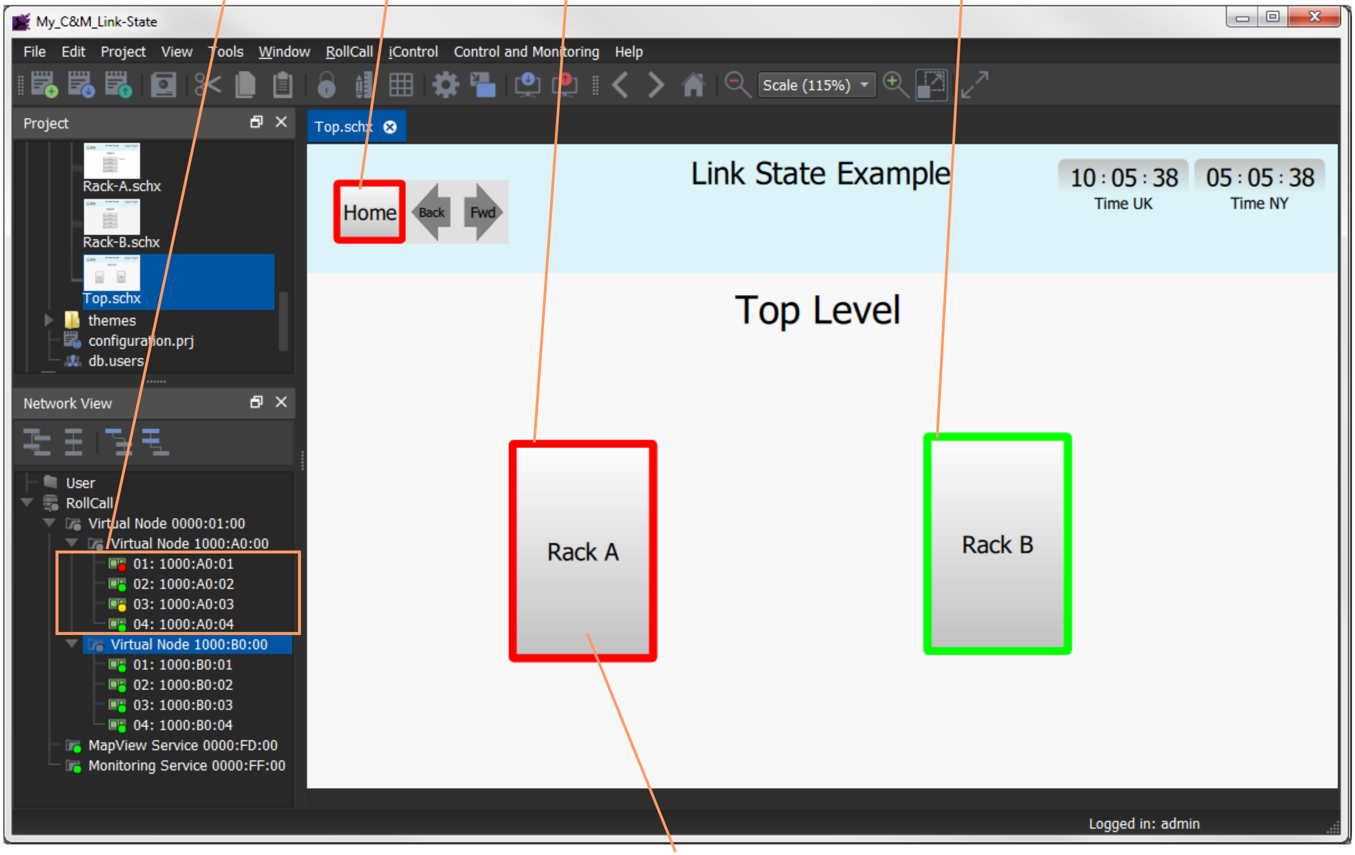
Error State Shown at Top Level

Network View indicates one (or more) error states in devices.

Home button indicates:
One or more error states (and zero or more warning states) on the Top screen.

Rack button indicates:
One or more error states (and zero or more warning states) on the Rack A screen.

Rack button indicates:
All OK on the Rack B screen.



Click button to open Rack A screen. See Figure 8-31.

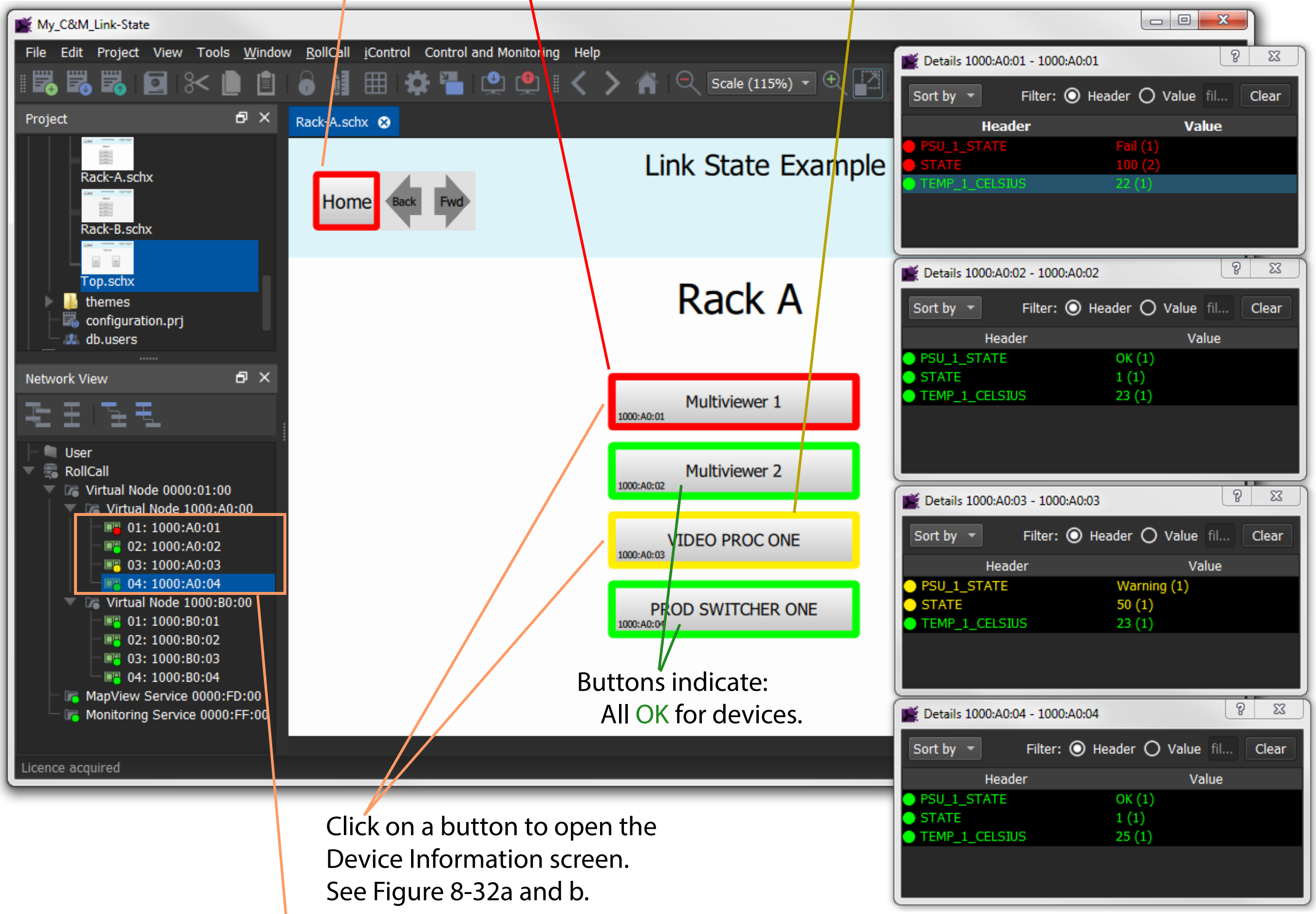
Fig. 8-30: Top-Level Screen - Error State on Rack A Screen

Error State Shown at a Lower (Rack) Level

Home button still indicates:
One or more error states and zero or more warning states
on the top screen.

Device button indicates that there are:
One or more error states and zero or more warning states
for device 'Multiviewer 1'.

Device button indicates that there are:
One or more warning states
for device 'Video Proc'.



Details 1000:A0:01 - 1000:A0:01

Sort by Filter: ● Header ○ Value fil... Clear

Header	Value
🟥 PSU_1_STATE	Fail (1)
🟥 STATE	100 (2)
🟩 TEMP_1_CELSIUS	22 (1)

Details 1000:A0:02 - 1000:A0:02

Sort by Filter: ● Header ○ Value fil... Clear

Header	Value
🟩 PSU_1_STATE	OK (1)
🟩 STATE	1 (1)
🟩 TEMP_1_CELSIUS	23 (1)

Details 1000:A0:03 - 1000:A0:03

Sort by Filter: ● Header ○ Value fil... Clear

Header	Value
🟨 PSU_1_STATE	Warning (1)
🟨 STATE	50 (1)
🟩 TEMP_1_CELSIUS	23 (1)

Details 1000:A0:04 - 1000:A0:04

Sort by Filter: ● Header ○ Value fil... Clear

Header	Value
🟩 PSU_1_STATE	OK (1)
🟩 STATE	1 (1)
🟩 TEMP_1_CELSIUS	25 (1)

Buttons indicate:
All OK for devices.

Click on a button to open the
Device Information screen.
See Figure 8-32a and b.

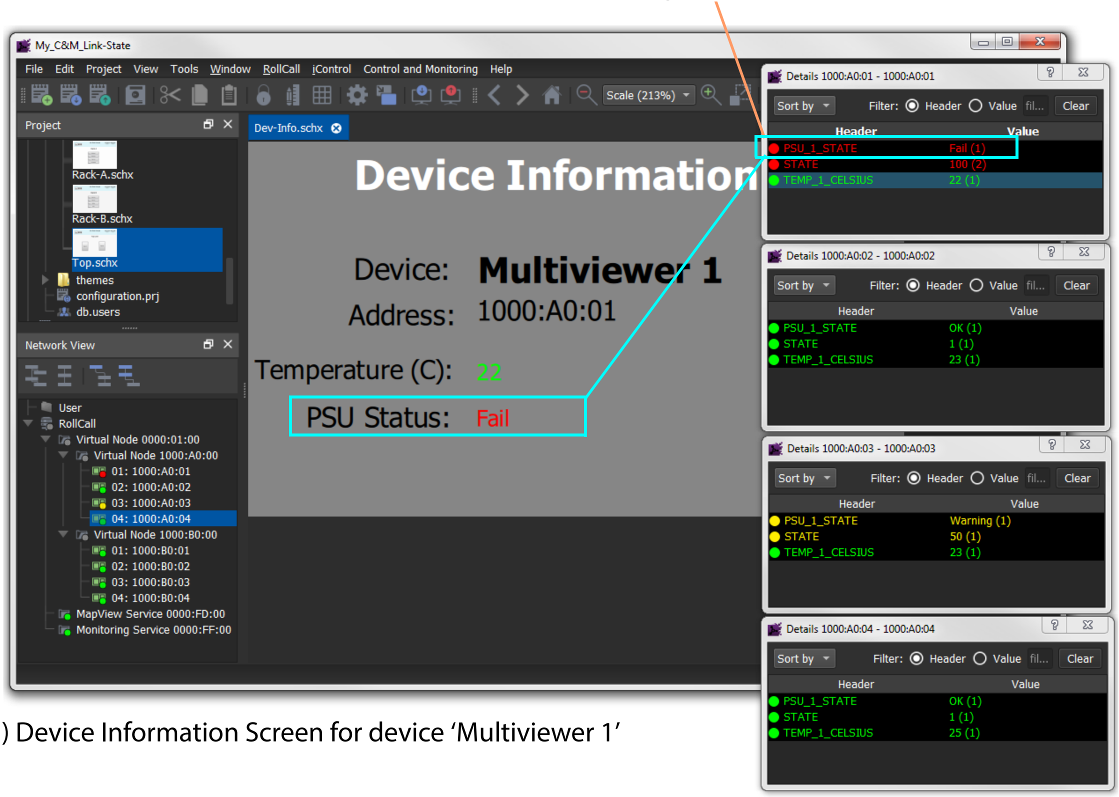
In the **Network View** pane, to show Log Fields for each device:

- Right-click on a device item and select 'Details'.
  The **Details** window for the device is shown.

Fig. 8-31: Rack A Screen and Device Details Windows

Error State Shown at Lowest (Device) Level

Device information screen shows Log Fields whose values are also shown in the corresponding 'Details' window.



Device Information

Device: Multiviewer 1

Address: 1000:A0:01

Temperature (C): 22

PSU Status: Fail

Details 1000:A0:01 - 1000:A0:01

Header	Value
● PSU_1_STATE	Fail (1)
● STATE	100 (2)
● TEMP_1_CELSIUS	22 (1)

Details 1000:A0:02 - 1000:A0:02

Header	Value
● PSU_1_STATE	OK (1)
● STATE	1 (1)
● TEMP_1_CELSIUS	23 (1)

Details 1000:A0:03 - 1000:A0:03

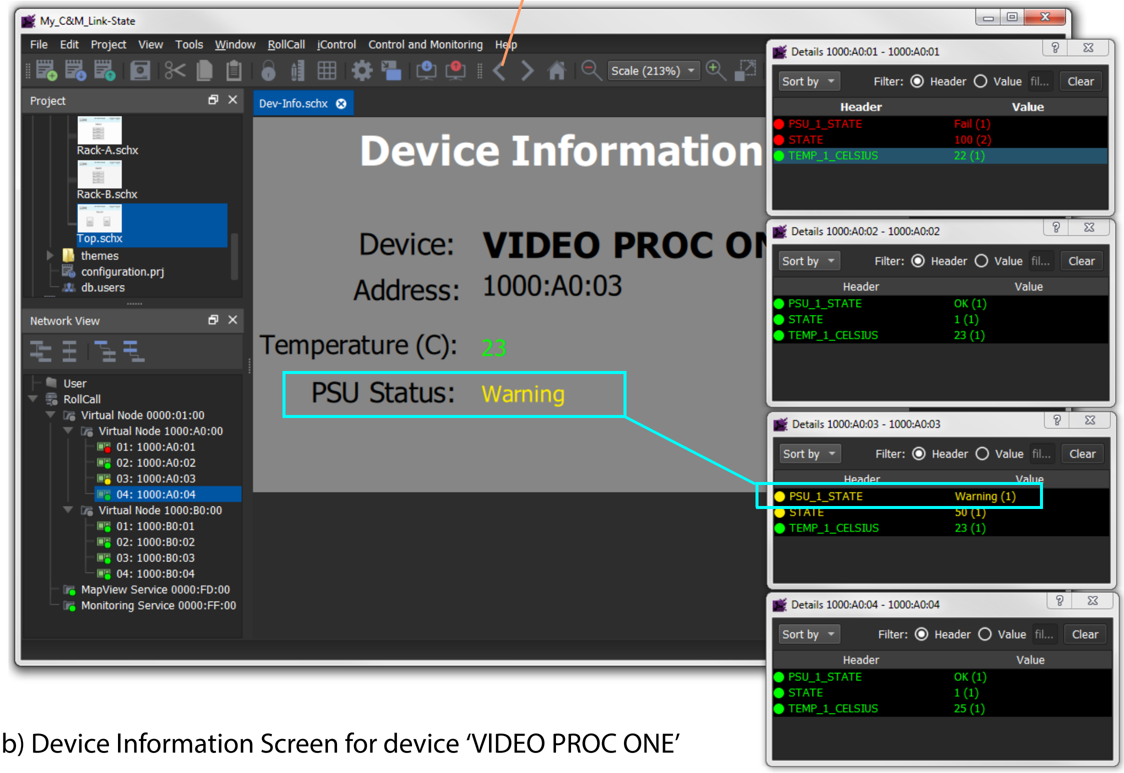
Header	Value
● PSU_1_STATE	Warning (1)
● STATE	50 (1)
● TEMP_1_CELSIUS	23 (1)

Details 1000:A0:04 - 1000:A0:04

Header	Value
● PSU_1_STATE	OK (1)
● STATE	1 (1)
● TEMP_1_CELSIUS	25 (1)

a) Device Information Screen for device 'Multiviewer 1'

Click to go back



Device Information

Device: VIDEO PROC ON

Address: 1000:A0:03

Temperature (C): 23

PSU Status: Warning

Details 1000:A0:01 - 1000:A0:01

Header	Value
● PSU_1_STATE	Fail (1)
● STATE	100 (2)
● TEMP_1_CELSIUS	22 (1)

Details 1000:A0:02 - 1000:A0:02

Header	Value
● PSU_1_STATE	OK (1)
● STATE	1 (1)
● TEMP_1_CELSIUS	23 (1)

Details 1000:A0:03 - 1000:A0:03

Header	Value
● PSU_1_STATE	Warning (1)
● STATE	50 (1)
● TEMP_1_CELSIUS	23 (1)

Details 1000:A0:04 - 1000:A0:04

Header	Value
● PSU_1_STATE	OK (1)
● STATE	1 (1)
● TEMP_1_CELSIUS	25 (1)

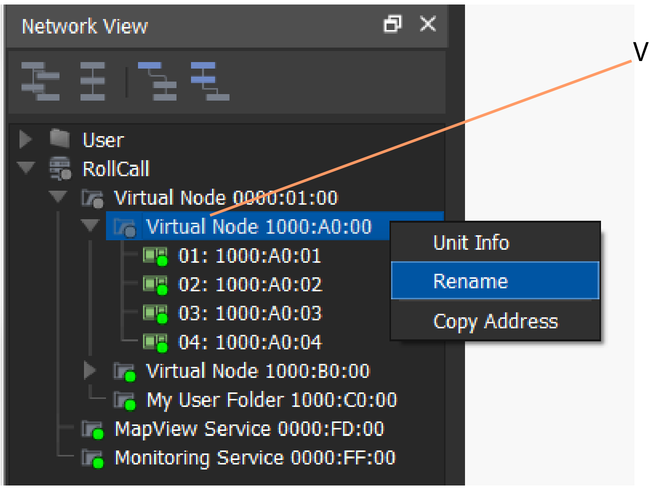
b) Device Information Screen for device 'VIDEO PROC ONE'

Fig. 8-32: Device Information Screens with Device Details Windows:
a) Multiviewer 1 Device.
b) Video Proc Device.

Renaming Virtual Nodes in Network View

While Orbit Services are running,
any 'Virtual Node' names may be renamed to more friendly names:

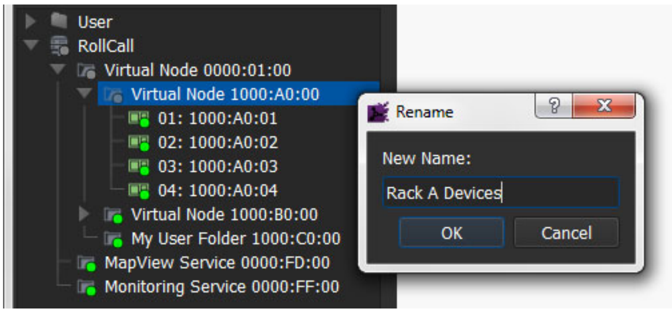
1 Right-click on the node name and select 'Rename'.



Virtual Node

Fig. 8-33: Rename Virtual Node in Network View

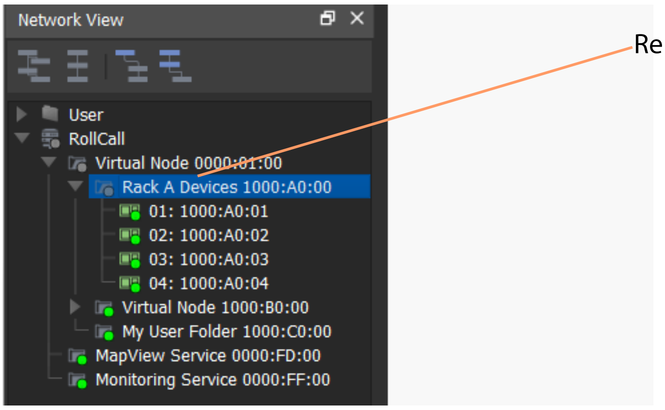
2 Enter a new name in the **Rename** dialog and click **OK**.



New Name: Rack A Devices

Fig. 8-34: Rename Dialog

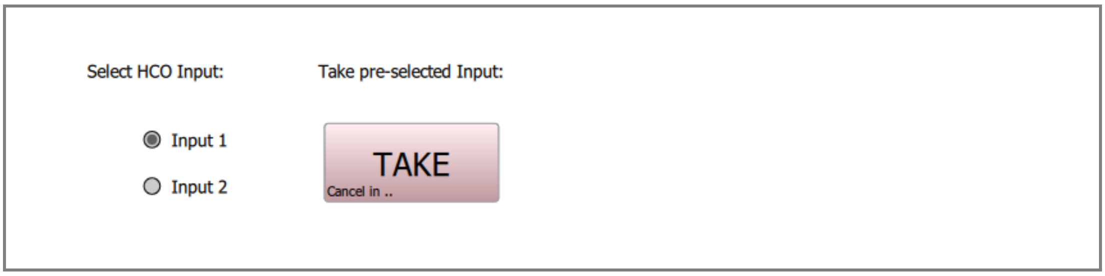
The Network View node shows its new name. See Figure 8-35.



Renamed node in **Network View**

Fig. 8-35: Renamed Network View Node.

Example - 'Control with Take' for a Manual HCO Switch-Over



This example shows how the controls for an advanced user-controlled video switch-over can be easily created for Orbit MapView with no programming. Instead, two 'radio button' widgets, one 'Take' button widget and some simple logic are used.

Introduction

Simple Manual Switch-over Case

A simple manual user-controlled switch-over could be created with two **Radio Button** widgets, each button causing a change-over command to be sent to an external HCO device using **Event** Bindings and Command Behaviours (e.g. RollCall v3 Command Behaviour).

However, this example will synthesize a more advanced switch-over arrangement:

Advanced Switch-over

In this example, a 'Take' button will be used to apply a video change-over on a Grass Valley HCO device. A user pre-selects the required HCO signal source and then applies it by clicking one ('Take') button.

Furthermore, the 'Take' button will only be visible when a signal input source is selected by the user on the screen. The 'Take' button will be hidden again after 5 seconds and the time remaining for the 'Take' button will be displayed on the button face.

A summary of our requirements:

- Two radio buttons to pre-select the HCO input
- A 'Take' button to apply the pre-selected input to the HCO device.
- 'Take' button to be hidden when nothing is pre-selected.
- 'Take' button should be hidden after a 5 seconds timeout period and any pre-selected input is undone.
- The 'Take' button 'time remaining' should be displayed on the button face.

Graphically, the example is shown in Figure 8-36.

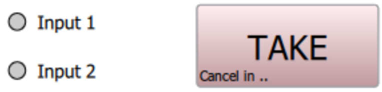


Fig. 8-36: HCO Switch-Over Example

Preliminaries

In this example, a Grass Valley Densité HCO-3901 card is to be controlled. This device is assumed to be working under a Grass Valley iControl server.

Add Widgets

1 Open a new screen (C&M project).

2 Add:

- two **Radio Button** widgets; and
- one **Button** widget.

See Figure 8-37.

3 Click **Save**.

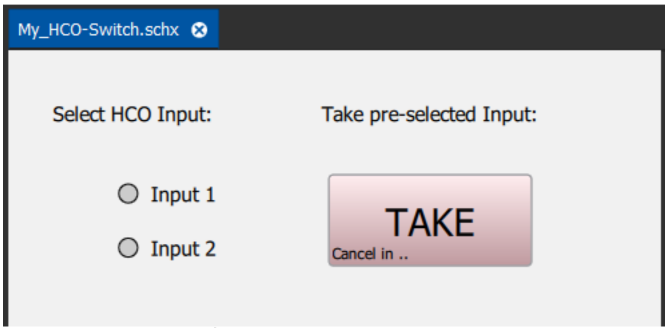


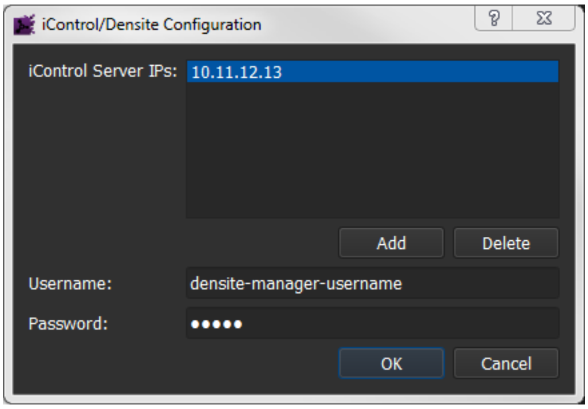
Fig. 8-37: HCO Switch-Over Screen

The next step is to build the ‘behind-the-scenes’ logic that resides within the screen. This is best built up in stages and tested in stages.

Configure Orbit to Connect to an iControl Server

To connect the Orbit tool to an iControl server:

1 Click **iControl** > **Configuration** in the Orbit main menu.
The iControl/Densité Configuration dialog is shown. See Figure 8-38



iControl Server IPs: 10.11.12.13

Username: densite-manager-username

Password: ●●●●●

Fig. 8-38: iControl/Densité Configuration Dialog

2 Add the IP address of the target iControl server.

3 Enter the **Username** and **Password** for the Densité Manager.

4 Click **OK**.

This configures Orbit to connect to an iControl server.

Accessible units are shown in the **Network View** pane of Orbit.

Add a Behaviour to Control the HCO Device

A **Densité** Behaviour is required to connect to a parameter on the HCO-3901 device.
To add the Behaviour, either:

- Select the background of the screen and display the ‘Behaviour/Bindings’ graphical editor.
- Add a **Densité** Behaviour.
- Configure the Behaviour’s ‘Name’, ‘Card Id’ and ‘Parameter Id’ properties for the target HCO-3901 device.

Or:

- Use the **Network View** method, described in Add Behaviour with Network View Method, on page 266.

Add Behaviour with Network View Method

This method allows a user to get settings for a Behaviour directly from a device. The **Network View** shows the device network that Orbit can see.

The **Network View** method is described below:

1 Explore the device network in the **Network View**.
And locate the device required.
(For our example, locate an iControl server and the required Densité device within it.)

2 Right-click on the device item
and select ‘Display Card Parameters’.

A list of device parameters is shown in a tab.

(For our example, a list of HCO-3901 Densité device parameters is shown. See Figure 8-39.)

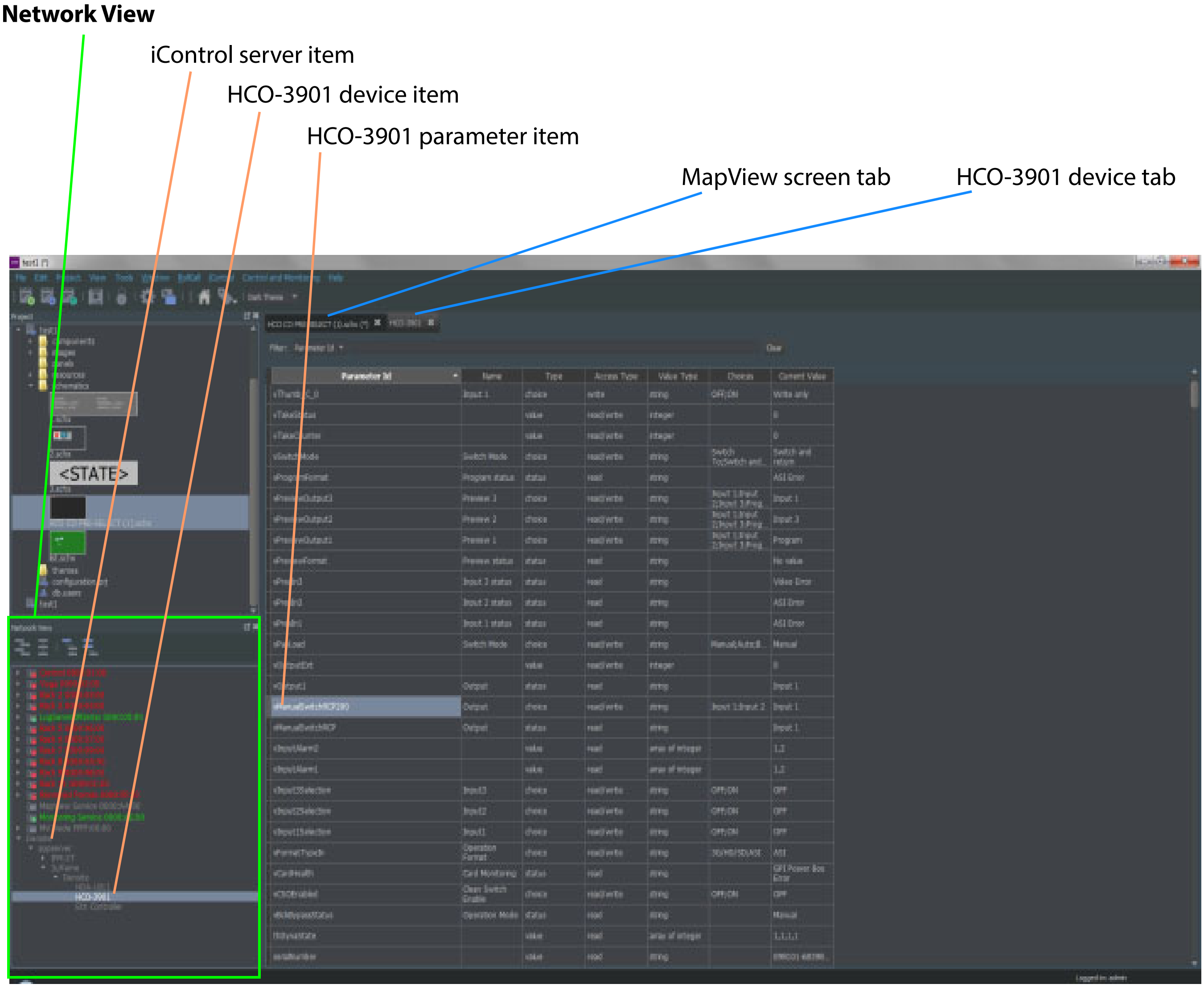


Fig. 8-39: Device Tab

In the device (HCO-3901) tab:

3 Right-click on the HCO-3901's 'vManualSwitchRCP2000' parameter
(i.e. the selected HCO input)
and select '**Copy as Behaviour**'.

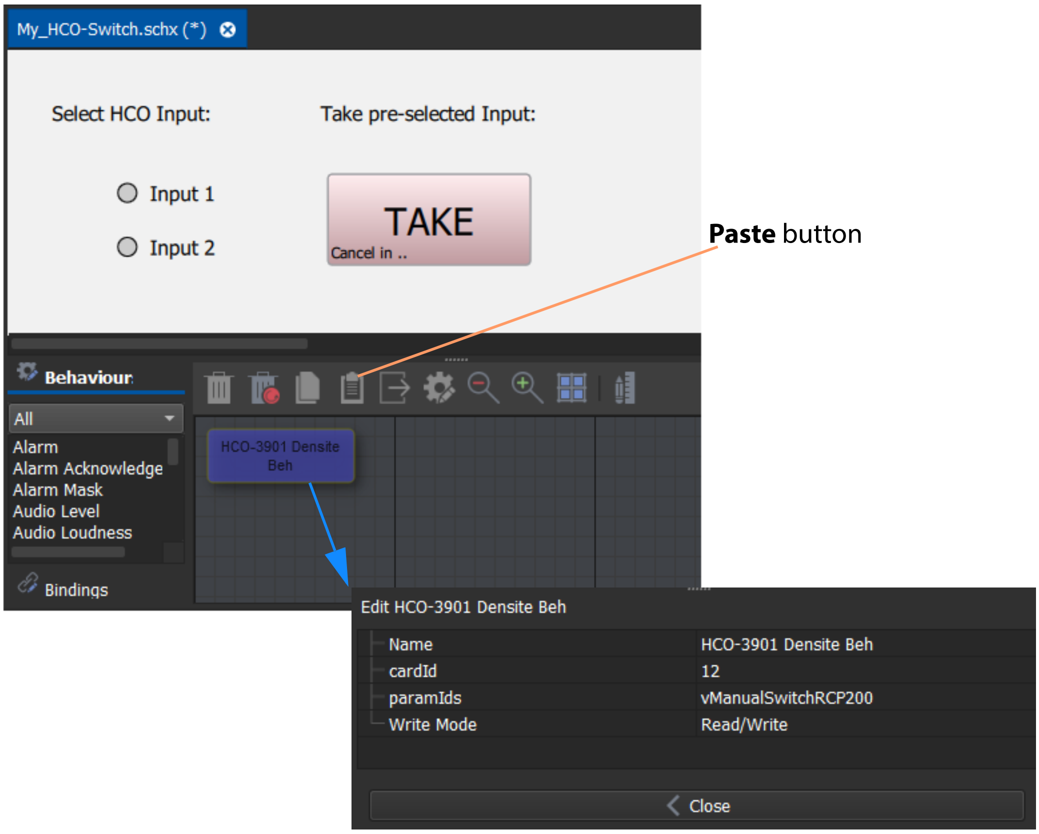
Return to the Orbit MapView screen (click on the MapView screen's tab).

In the Orbit MapView screen:

4 Click the **Edit Behaviours** button to show the 'Behaviour and Binding' graphical editor. And click the **Paste** button.

For our example, this adds a **Densité** Behaviour configured to interface to the HCO-3901's parameter. See Figure 8-40.

The **Densité** Behaviour will now control and reflect the HCO-3901's selected input.



Paste button

Edit HCO-3901 Densite Beh

Name	HCO-3901 Densite Beh
cardId	12
paramIds	vManualSwitchRCP200
Write Mode	Read/Write

Fig. 8-40: HCO-3901 Densité *Behaviour Pasted In*

Configure Radio Buttons

The **Densité** Behaviour value may now be changed on this Orbit MapView screen and reflected on the HCO-3901 device. To do this via our radio buttons:

1 Select the “Input 1” **Radio Button** widget and add a **Radio Button** Binding.

2 Configure the Binding:

- to use the **Densité** Behaviour as its source Behaviour; and
- set ‘Checked when value is equal to:’ to `“Input 1”`.

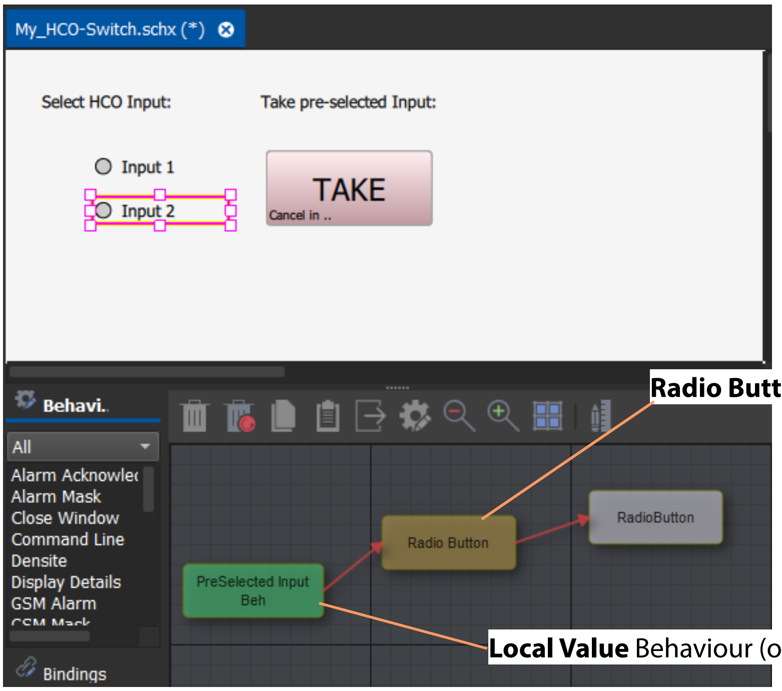
3 Select the “Input 2” **Radio Button** widget and add a **Radio Button** Binding.

4 Configure the Binding:

- to use the **Densité** Behaviour as its source Behaviour; and
- set ‘Checked when value is equal to:’ to `“Input 2”`.

5 Click **Close**.

6 Click **Save**.



Radio Button Binding

Local Value Behaviour (on the screen background)

Fig. 8-41: Orbit MapView Screen So Far

Exercise the Radio Buttons

1 Click the **Test** button and exercise the radio buttons.

The **Densité** Behaviour value changes according to which radio button is selected. And the HCO-3901 card changes input accordingly. (If Orbit is connected to the HCO-3901.)

2 Click the **Test** button again to halt exercising.

Configure Pre-select for TAKE Button

To implement a pre-select capability, the resulting value from the radio buttons must be stored locally in the Orbit MapView screen and only passed to the **Densité** Behaviour when the user clicks the TAKE button.

On the Orbit MapView screen with the ‘Behaviour and Binding’ graphical editor visible:

1 Select the Orbit MapView screen background.

2 Add a **Local Value** Behaviour named ‘PreSelected Input Beh’.

And:

3 Edit the **Radio Button** Bindings to now use ‘PreSelected Input Beh’ as their source Behaviour.

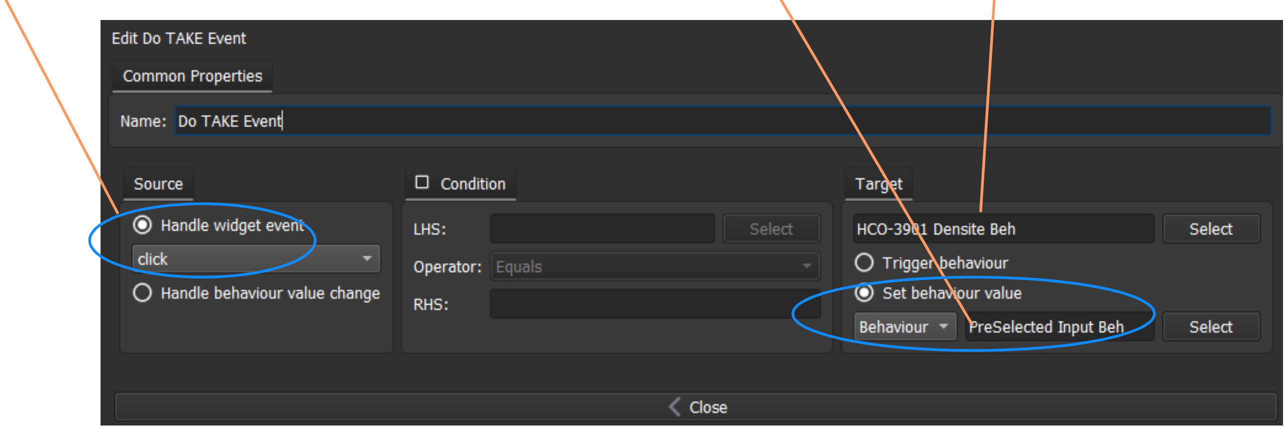
Then:

4 Select the TAKE button widget.

5 Add an **Event** Binding for the TAKE button and configure it as shown in Figure 8-42 below.

Use the value of this Behaviour to set this target Behaviour’s value

Act on a button click



Edit Do TAKE Event

Common Properties

Name: Do TAKE Event

Source

● Handle widget event

click

○ Handle behaviour value change

□ Condition

LHS: Select

Operator: Equals

RHS:

Target

HCO-3901 Densite Beh Select

○ Trigger behaviour

● Set behaviour value

Behaviour PreSelected Input Beh Select

Fig. 8-42: TAKE Button Event Binding

6 Click **Close**.

7 Click **Save**.

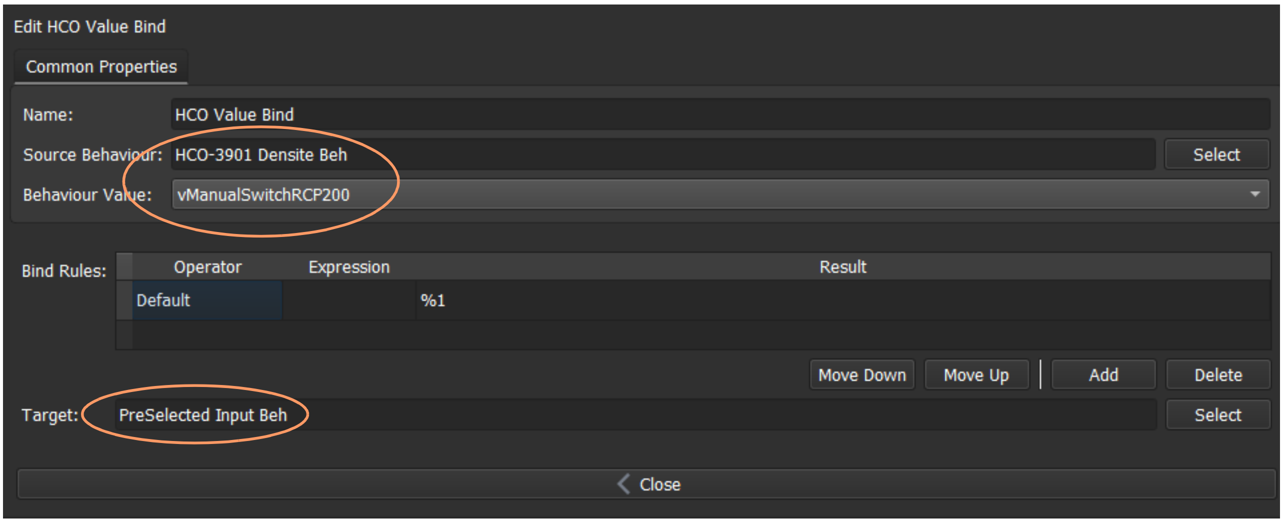
The resulting value from the pair of radio buttons will be sent to the **Densité** Behaviour to control the HCO-3901 device when TAKE is clicked.

However, if the HCO-3901 device is changed by some other means, then this must be reflected on our Orbit MapView screen. To implement this, on the Orbit MapView screen:

8 Select Orbit MapView screen background.

9 Add a **Mapped** Binding and configure it as shown in Figure 8-43.

10 Click **Close** and click **Save**.



Edit HCO Value Bind

Common Properties

Name: HCO Value Bind

Source Behaviour: HCO-3901 Densite Beh [Select]

Behaviour Value: vManualSwitchRCP200

Bind Rules:

Operator	Expression	Result
Default		%1

Move Down | Move Up | Add | Delete

Target: PreSelected Input Beh [Select]

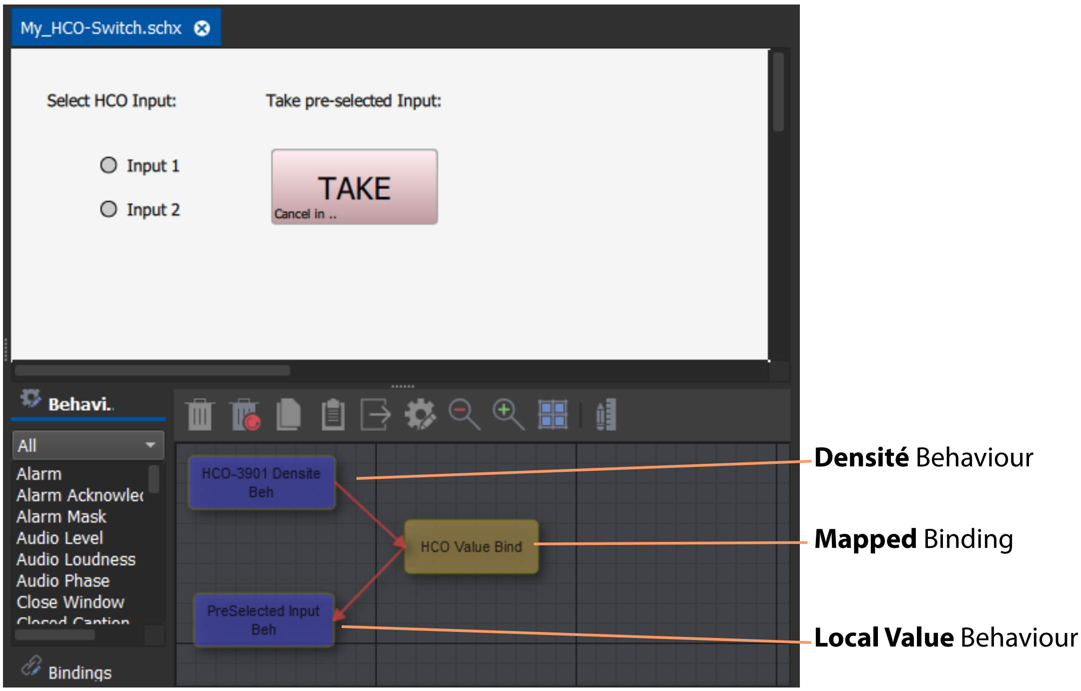
Close

Fig. 8-43: Mapped Binding Configured

This will ensure the local value is updated if the HCO-3901 changes.

Note: Unlike a **Property** Binding, a **Mapped** Binding requires a default rule to be in-place. (Because at least one rule must match for the Binding to pass a value.)
A **Property** Binding with no rules will still pass a value:
A **Mapped** Binding will not.

The Orbit MapView screen so far is shown in Figure 8-44.



Densité Behaviour

Mapped Binding

Local Value Behaviour

Fig. 8-44: Orbit MapView Screen So Far

TAKE Button Visibility

The TAKE button shall be visible when the pre-selected value is different to that on the HCO-3901 device.

On the Orbit MapView screen, in the 'Behaviour and Binding' graphical editor:

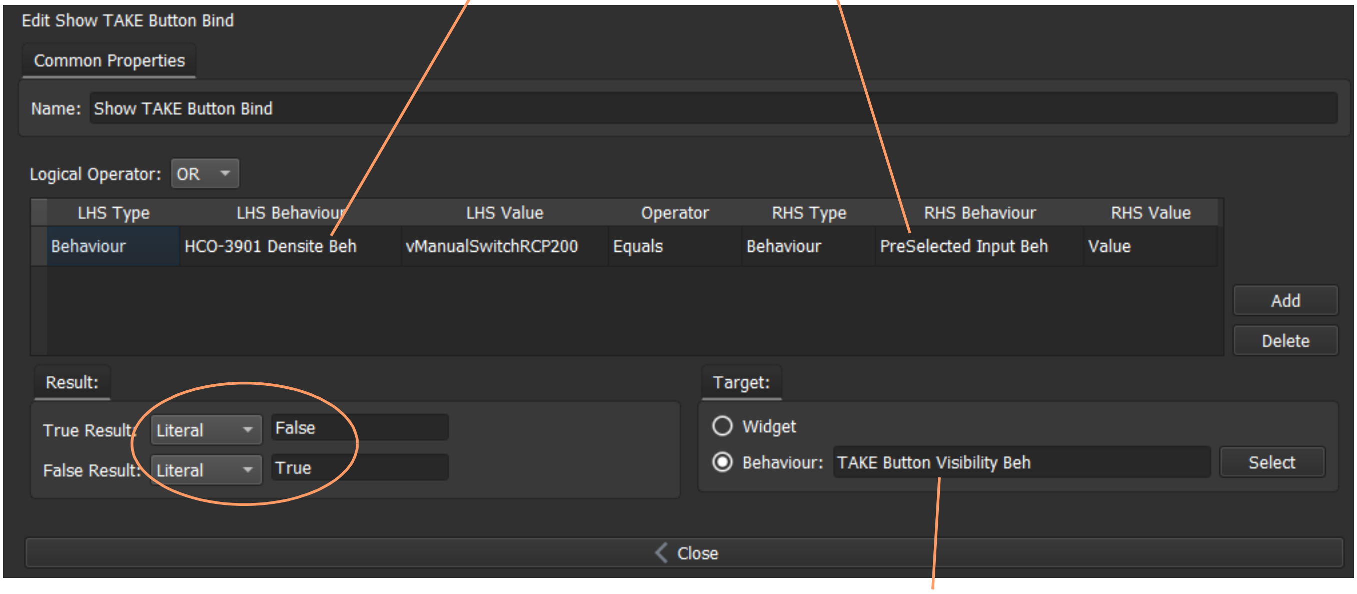
1 Select the TAKE button widget
and add a **Logical** Binding named 'Show TAKE Button Bind.

This will be used to compare two values.

2 Edit the **Logical** Binding settings, see Figure 8-45.

Add a row and configure the Binding as shown in Figure 8-45.

'Input' Behaviours = 'HCO-3901 Densité Beh' and 'PreSelected Input Beh'



Edit Show TAKE Button Bind

Common Properties

Name: Show TAKE Button Bind

Logical Operator: OR

LHS Type	LHS Behaviour	LHS Value	Operator	RHS Type	RHS Behaviour	RHS Value
Behaviour	HCO-3901 Densite Beh	vManualSwitchRCP200	Equals	Behaviour	PreSelected Input Beh	Value

Result:

True Result: Literal False

False Result: Literal True

Target:

○ Widget

● Behaviour: TAKE Button Visibility Beh

New **Local Value** Behaviour as target

Fig. 8-45: Logical Binding Settings

When either 'HCO-3901 Densité Beh' or 'PreSelected Input Beh' Behaviour values (on the screen background) change, the **Logical** Binding will check and set the 'TAKE Button Visibility Beh' Behaviour value to:

- 'False' if they are equal; or to
- 'True' if they are not.

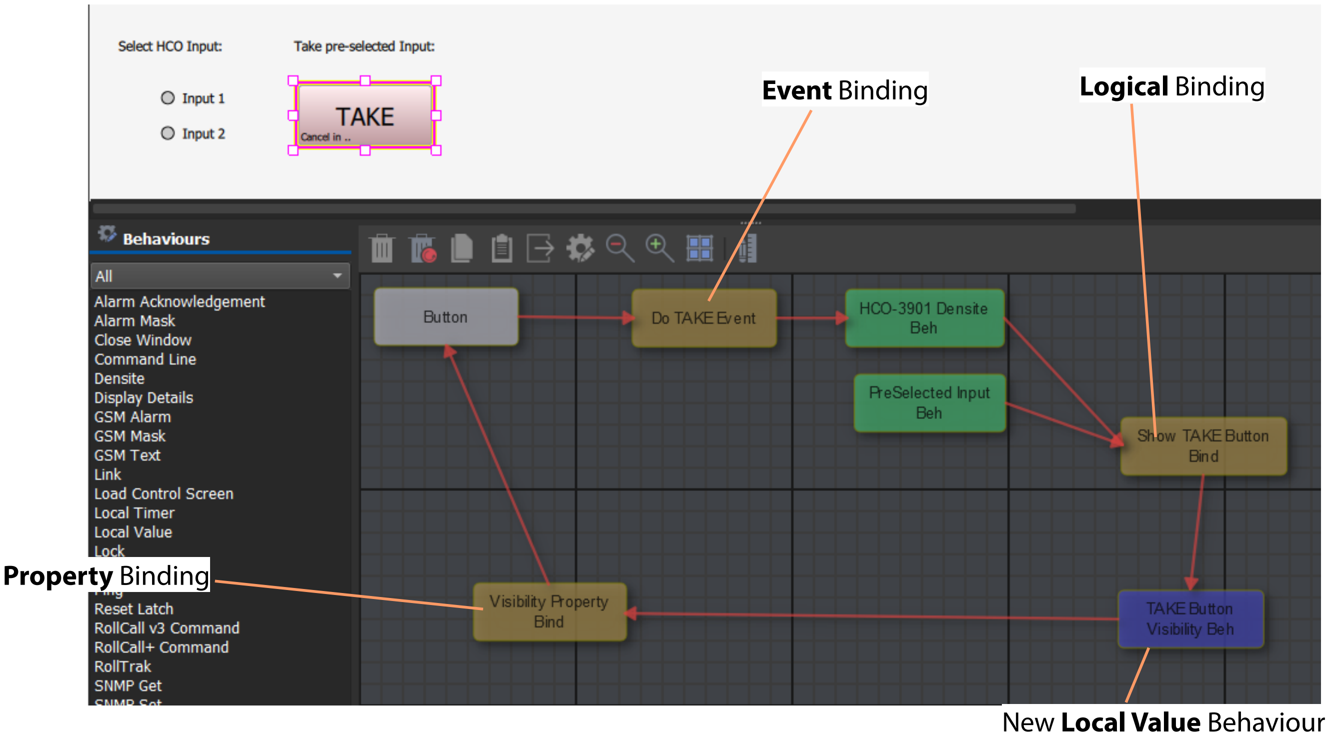
Then:

3 Add a **Property** Binding on the button.
And set its:

- 'Source Behaviour' to be the 'TAKE Button Visibility Beh' Behaviour; and
- 'Property to bind' to 'Visibility'.

4 Click **Close** and click **Save**.

The Bindings added so far are shown in Figure 8-46.



Event Binding

Logical Binding

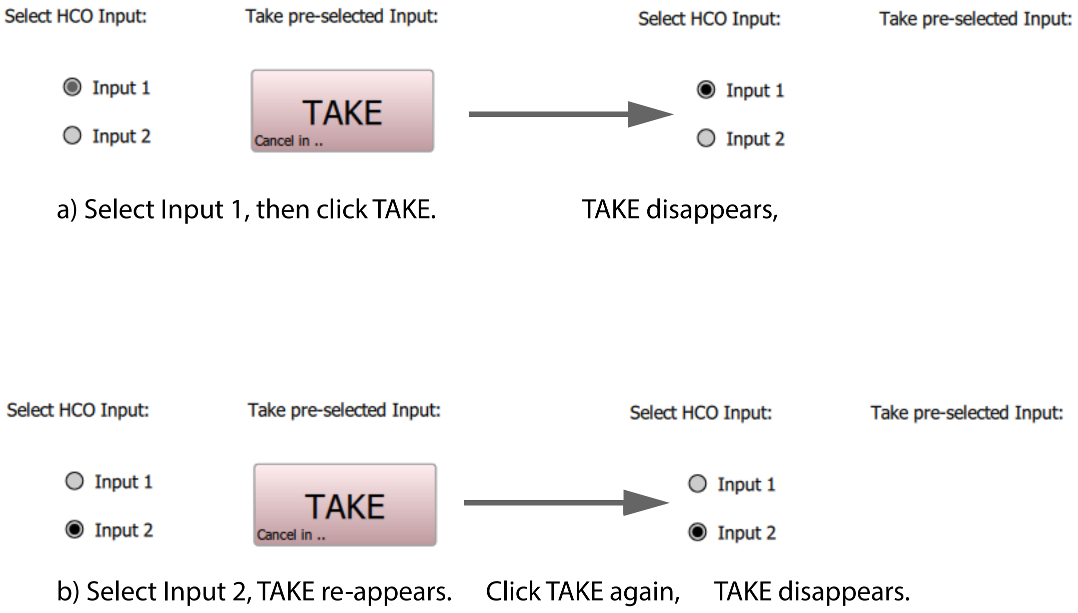
Property Binding

New **Local Value** Behaviour

Fig. 8-46: TAKE Button Bindings

Exercise the TAKE Button Visibility

1 Click the **Test** button and exercise the radio buttons. (See Figure 8-47.)



a) Select Input 1, then click TAKE. TAKE disappears,

b) Select Input 2, TAKE re-appears. Click TAKE again, TAKE disappears.

Fig. 8-47: Exercising TAKE Button Visibility

2 Select an input on the radio buttons.

3 Click TAKE.

The TAKE button disappears after it is clicked.

4 Select the other input.

The TAKE button appears.

To halt the exercising:

5 Click the **Test** button.

Cancel TAKE Pre-Select after 5 Seconds

In a real-life use-case, it is useful to cancel the pre-selected value automatically if it has not been ‘taken’ within a short time period (say within 5 seconds of when it was set).
To implement this, a timer be driven from the TAKE button visibility:

We can use a **Local Timer** Behaviour, driven off the TAKE button’s visibility value:

1 Select the TAKE Button widget
and view the ‘Behaviours and Bindings’ graphical editor.

2 Add a **Local Timer** Behaviour called ‘Cancel Timer Beh’.

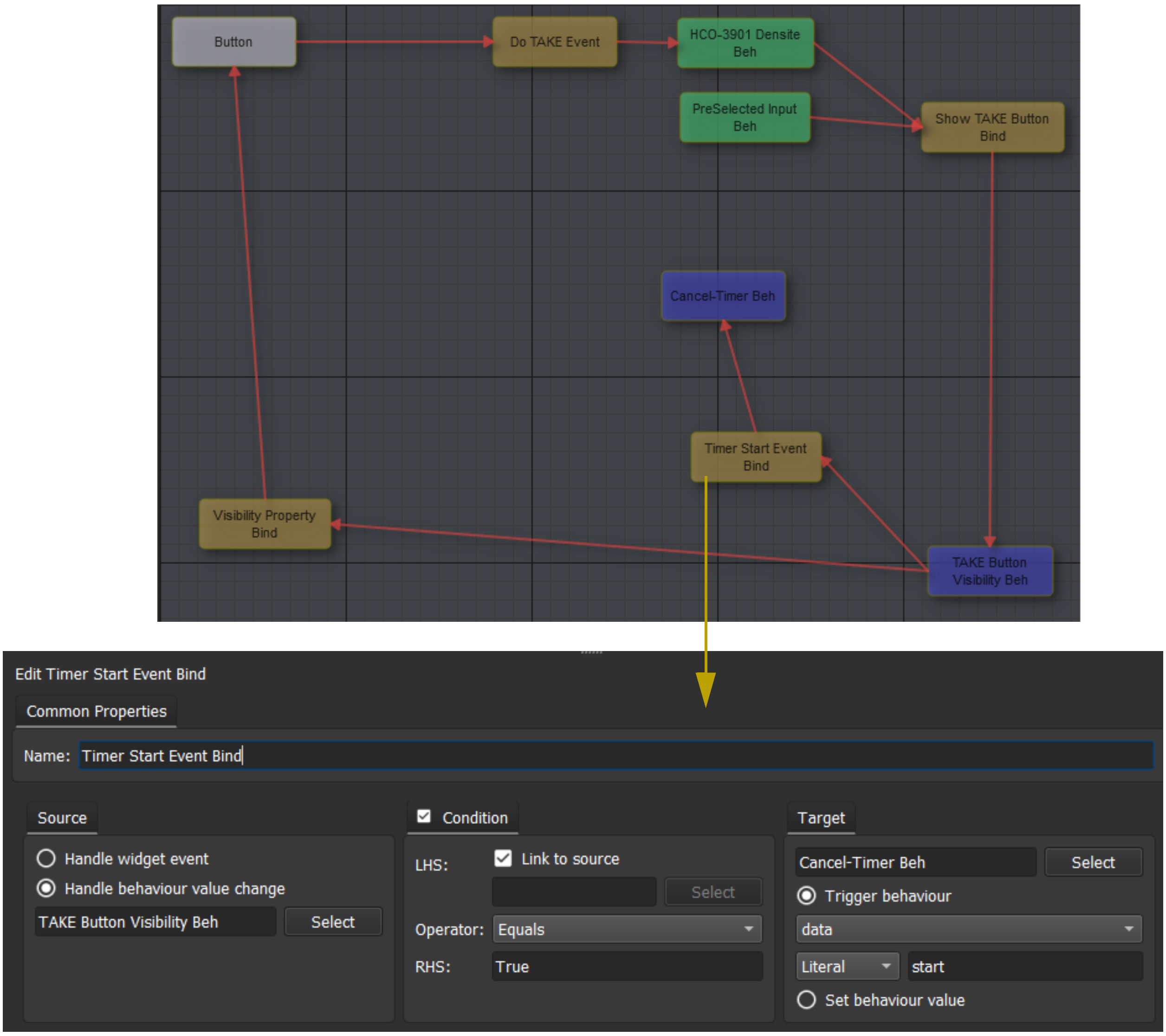
3 Add an **Event** Binding and configure it to:

- start the ‘Cancel Timer Beh’ when ‘TAKE Button Visibility Beh’ is set to ‘True’.

See Figure 8-48.

This **Event** Binding will handle *starting* the timer.

Note: The **Event** Binding is configured to execute the ‘Cancel-Timer’ Behaviour by passing it the argument ‘start’.



Edit Timer Start Event Bind

Common Properties

Name: Timer Start Event Bind

Source

○ Handle widget event

● Handle behaviour value change

TAKE Button Visibility Beh [Select]

☑ Condition

LHS: ☑ Link to source

[Select]

Operator: Equals

RHS: True

Target

Cancel-Timer Beh [Select]

● Trigger behaviour

data

Literal start

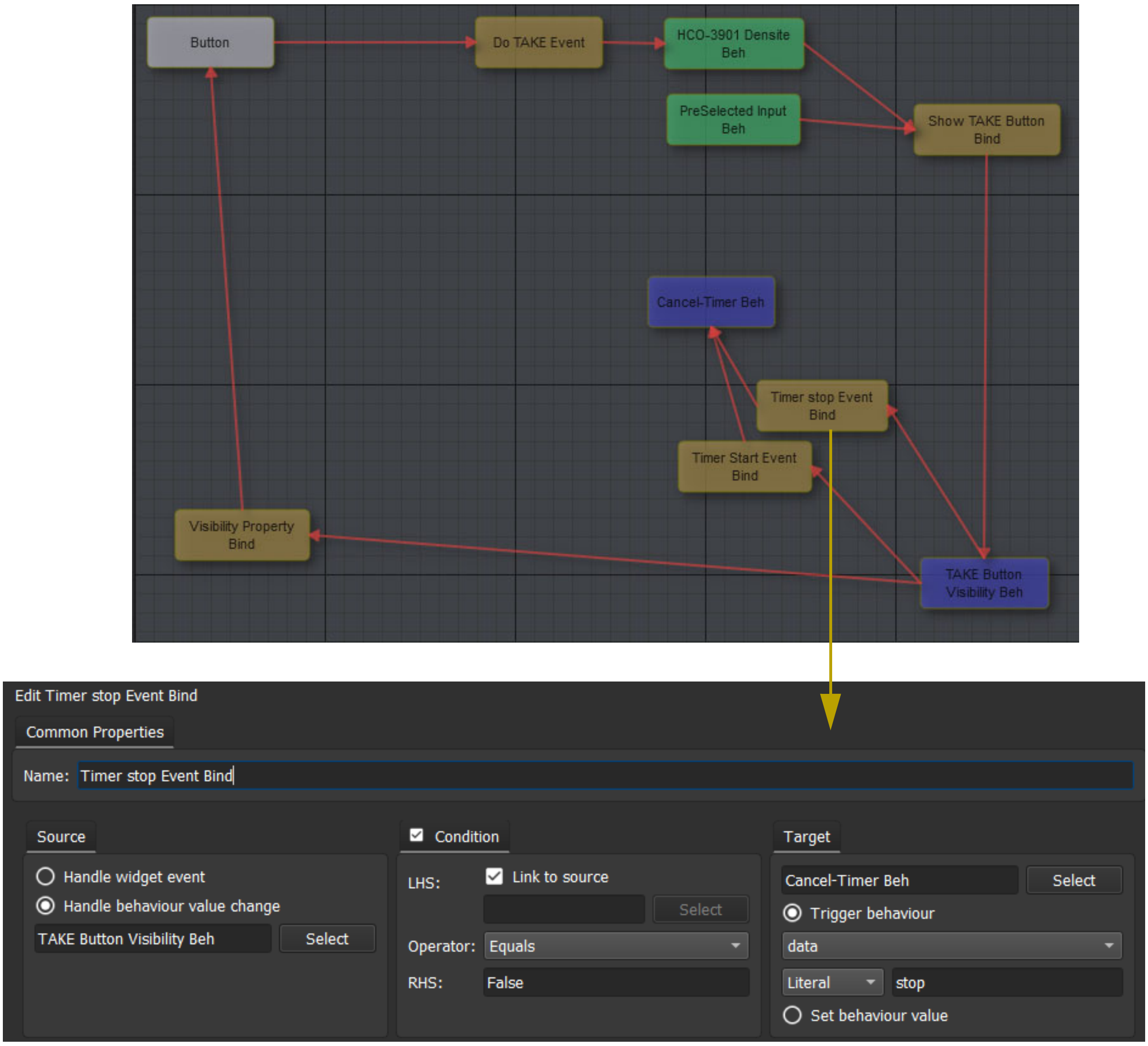
○ Set behaviour value

Fig. 8-48: Timer Start Event Binding and Settings

4 Add another **Event** Binding and configure it to stop the ‘Cancel Timer Beh’ when ‘TAKE Button Visibility Beh’ is set to ‘False’. See Figure 8-49.

This Binding will handle stopping the timer.

Note: The **Event** Binding is configured to execute the ‘Cancel-Timer’ Behaviour passing it the argument ‘stop’.



Edit Timer stop Event Bind

Common Properties

Name: Timer stop Event Bind

Source

○ Handle widget event
● Handle behaviour value change
TAKE Button Visibility Beh [Select]

☑ Condition

LHS: ☑ Link to source
[Select]
Operator: Equals
RHS: False

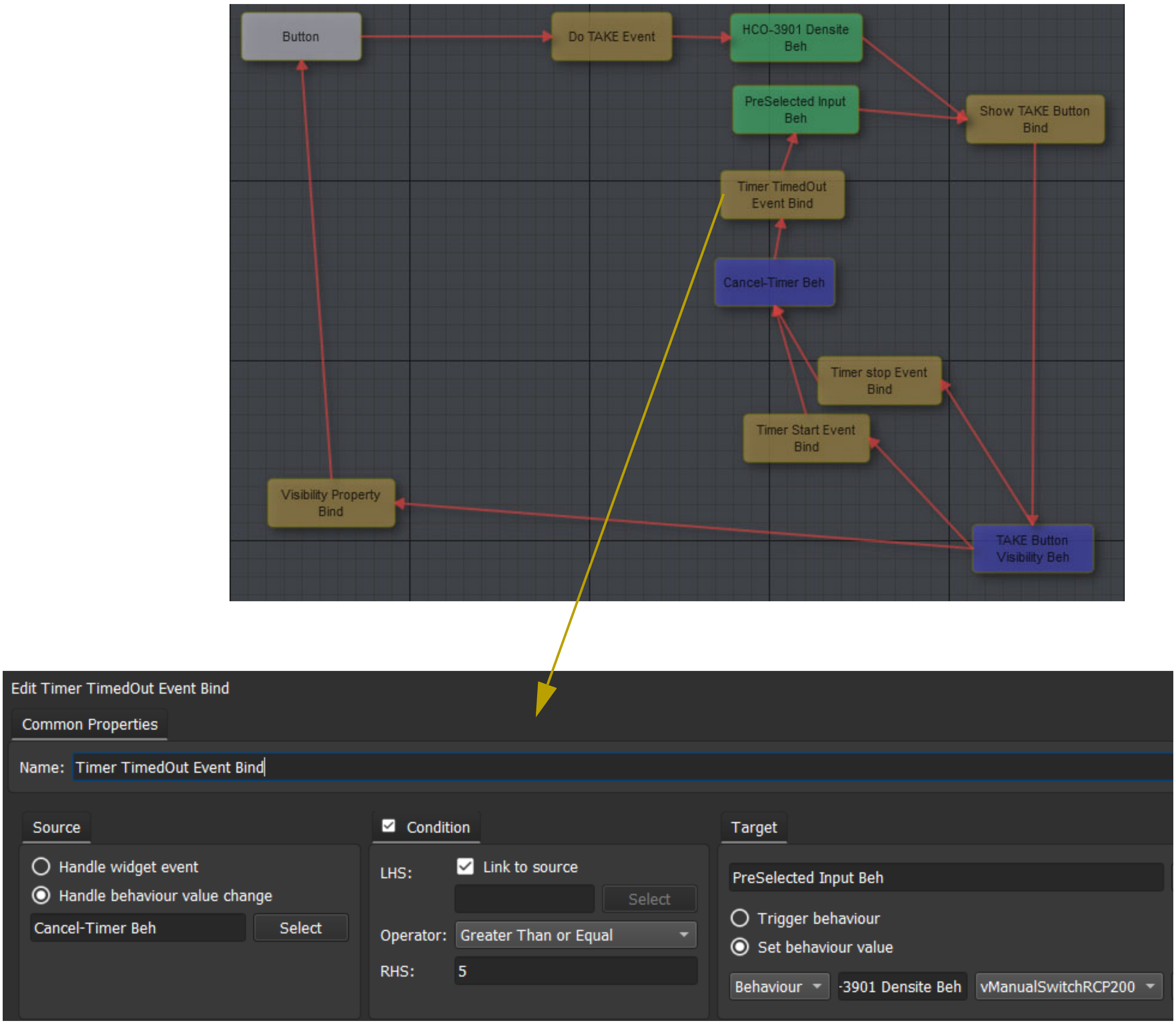
Target

Cancel-Timer Beh [Select]
● Trigger behaviour
data
Literal stop
○ Set behaviour value

Fig. 8-49: Timer Stop Event Binding and Settings

5 Add another **Event** Binding and configure this as shown in Figure 8-50.

This will be used to revert the value of the **PreSelected Input Beh** back to be the same as the HCO-3901 device after a 5 second timeout period. (Thus causing the TAKE button to disappear.)



Edit Timer TimedOut Event Bind

Common Properties

Name: Timer TimedOut Event Bind

Source
- ○ Handle widget event
- ● Handle behaviour value change
- Cancel-Timer Beh Select

☑ Condition
- LHS: ☑ Link to source Select
- Operator: Greater Than or Equal
- RHS: 5

Target
- PreSelected Input Beh
- ○ Trigger behaviour
- ● Set behaviour value
- Behaviour -3901 Densite Beh vManualSwitchRCP200

Fig. 8-50: Timer Timeout Event Binding Settings

6 Click **Close**.

7 Click **Save**.

Exercise the 'Cancel Pre-Select' Timeout

1 Click the **Test** button and exercise the radio buttons.

Five seconds after pre-selecting an input on the radio buttons, the TAKE button is hidden and the pre-selection reverts back.

To halt the exercising:

2 Click the **Test** button again.

Display the TAKE Timeout on the Button Face

As a useful aid to the user, the remaining time before the ‘pre-select’ is canceled etc. is to be shown on the TAKE button face. To do this, the remaining time needs to be calculated (i.e. 5 - ‘timer value’) and the value written onto the TAKE button.

1 Add a **Local Value** Behaviour called **Timeout Value** and set its initial value to 5.

  This is the (fixed) length of the timeout.

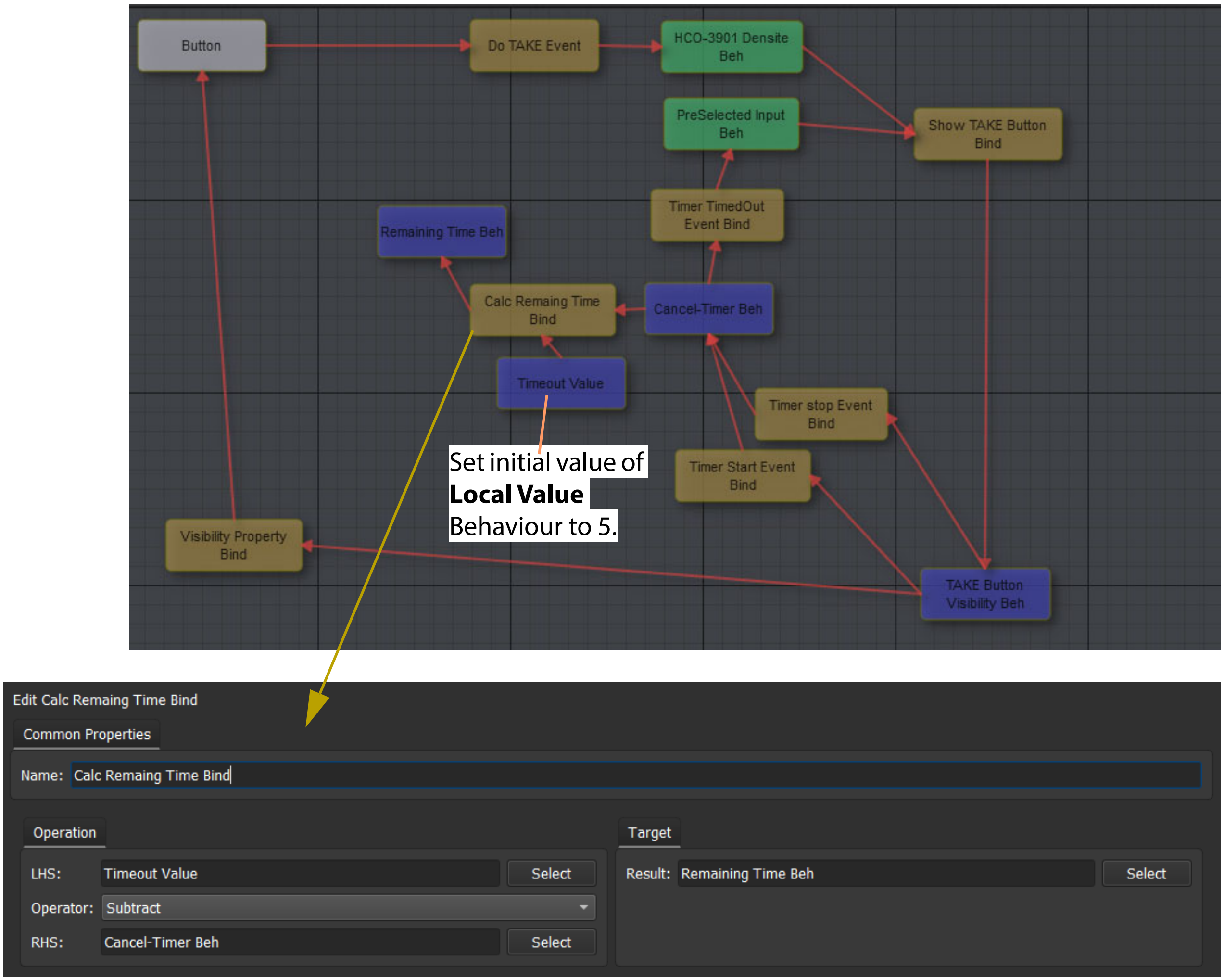
2 Add a **Local Value** Behaviour called **Remaining Time Beh** and set the initial value to 0.

  This will hold the time remaining.

3 Add a **Math** Binding and configure it to:

  - subtract **Cancel-Timer Beh** (timer value) from **Timeout Value**; and
  - place the result in **Remaining Time Beh**.

  See Figure 8-51.



Set initial value of **Local Value** Behaviour to 5.

Edit Calc Remaing Time Bind

Name: Calc Remaing Time Bind

LHS: Timeout Value

Operator: Subtract

RHS: Cancel-Timer Beh

Result: Remaining Time Beh

Fig. 8-51: Math Binding Settings

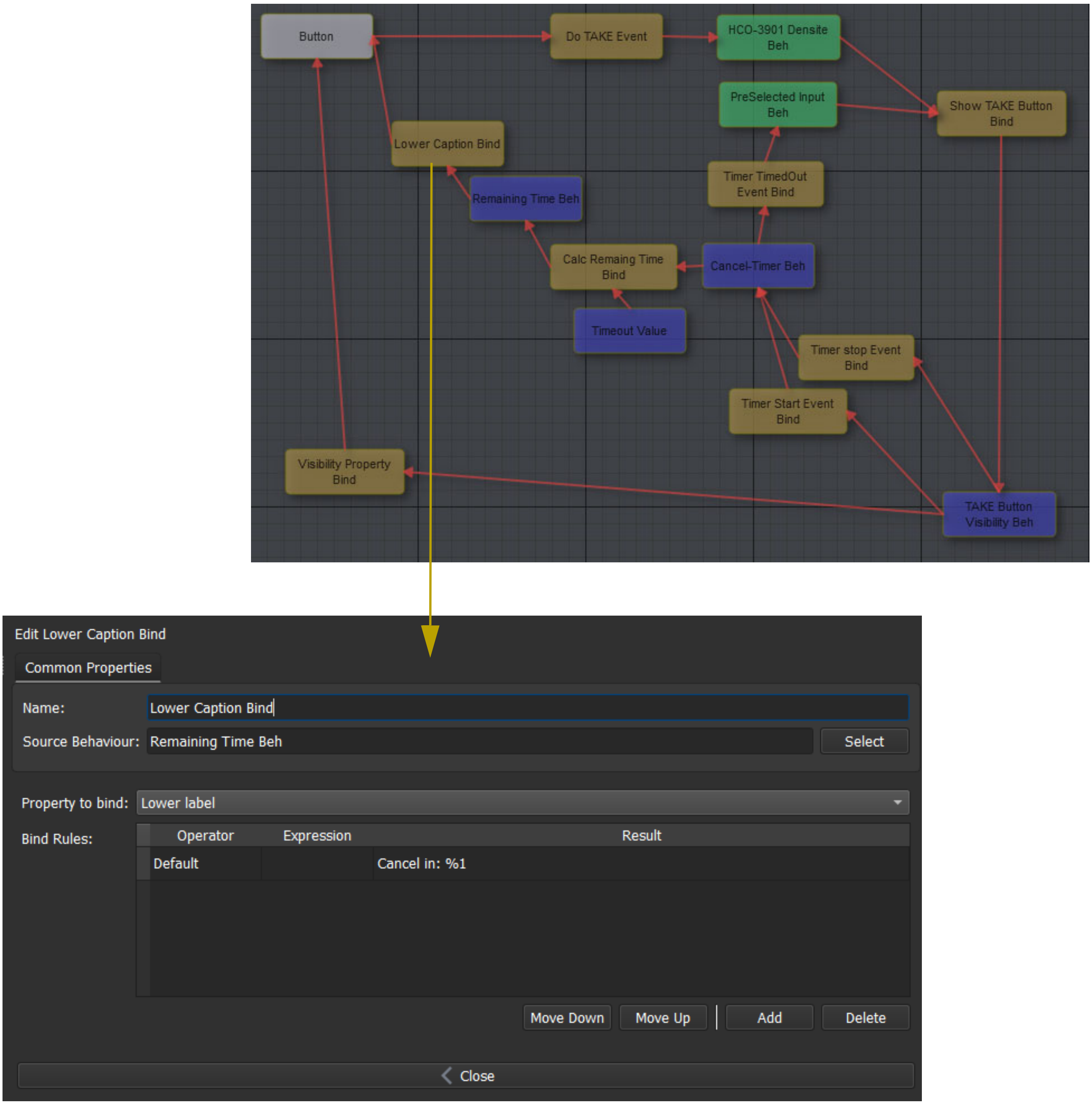
4 Add a **Property** Binding. Configure this to:

  - use **Remaining Time Beh** as the source Behaviour;
  - write to the button’s Lower Caption property; and to
  - use a string format with a prefix “Cancel in:”.

  See Figure 8-52.

5 Click **Close**.

6 Click **Save**.



Edit Lower Caption Bind

Common Properties

Name: Lower Caption Bind

Source Behaviour: Remaining Time Beh

Property to bind: Lower label

Bind Rules:

Operator	Expression	Result
Default		Cancel in: %1

Fig. 8-52: Configuring Lower Label Caption Property Binding

Exercise the Timeout Display on the Button Face

1 Click the **Test** button and exercise the radio buttons.

2 Select an input source on the radio buttons.
The TAKE button appears.

3 Click TAKE.
The TAKE button disappears.

4 Select another input on the radio buttons.
The TAKE button appears.

5 This time though, do not click the button.
The timeout is shown on the button face as a count down.
When the count down reaches 0, the button disappears and the radio button setting reverts.

To halt the exercising:

6 Click the **Test** button.

Bindings and Behaviours on the Take Button

The overall Behaviours and Bindings on the TAKE button are shown in Figure 8-53.

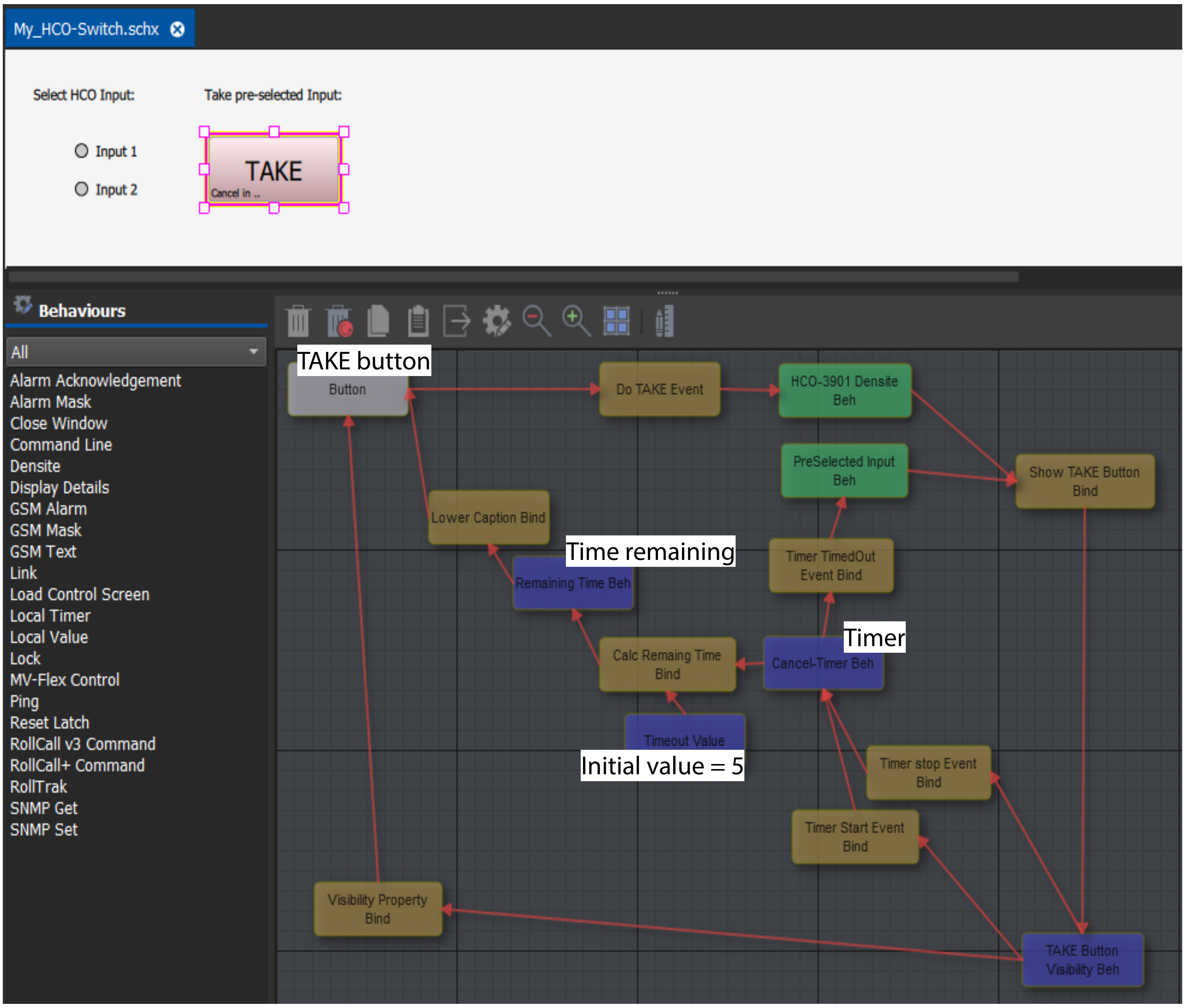


Fig. 8-53: TAKE Button Behaviours and Bindings

9 Server-side Processing Examples

Summary

Server-side Processing Examples

The Orbit MapView client application typically works with some other services which run on some system server computer. Services include: Log Server, iControl GSM, iControl Densité Manager and Orbit Services. These provide server-side processing for the Orbit MapView client application. For example, they can provide aggregated alarm messages formed from one or other combinations of device status messages.
(Please refer to separate documentation for information about the Log Server, Orbit Services and iControl.)

Additionally, an Orbit MapView project may define some server-side processing itself.

Orbit Global Files for Server-Side Processing

Log Field values are remote items, typically generated by system devices (for example a Densité module) or by some processing in the system (for example, a Log-Field-aggregating function may run by a Log Server or by an Orbit Monitoring Service).

On Orbit MapView Screens (Client-Side Processing)

Orbit MapView screens can access Log Field header-values via various Behaviours (for example, via **Log Field** and **Densité** Behaviours). These Behaviours can read or write Log Field header values, depending on how they are configured, and can interact with other screen widgets, Behaviours and Bindings.

MapView screen values written to a **Log Field** Behaviour ('Write Mode' property set to 'Write Only' or 'Read/Write') will be sent out to an Orbit Monitoring Service and onwards to the system. Thus system Log Field values may be changed by a MapView screen via some logic defined on the MapView screen.

This processing logic runs client-side in the MapView project on the client computer.

Server-Side Processing with Orbit MapView Projects

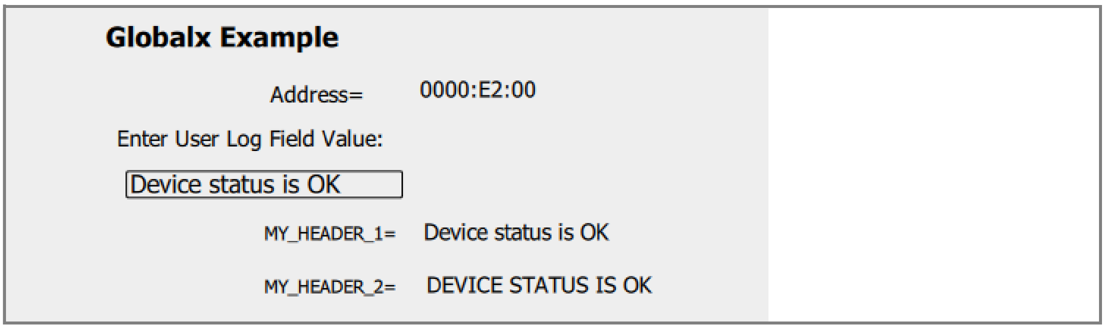
It is possible to specify some processing logic to be executed by the Orbit MapView Service, OMS, (i.e. server-side) and, hence, be independent of whether the client computer is powered up or down. The processing logic is specified in a .globalx file in an Orbit MapView project.

To do this, in the Orbit project:

- create a new file of global type (a.globalx file);
- place the required logic inside the.globalx file;
- save the file and save the entire project; and
- ensure the Orbit MapView Service is using the same Orbit MapView project. (This may require restarting the OMS.)

When the Orbit MapView project is subsequently run, the logic in the.globalx file is executed by the OMS.

Example - Simple Global File for Server-Side Processing



Globalx Example

Address= 0000:E2:00

Enter User Log Field Value:

Device status is OK

MY_HEADER_1= Device status is OK

MY_HEADER_2= DEVICE STATUS IS OK

This example will:

- use a virtual device with an unused RollCall address; and
- create a globalx file to process a Log Field of the new device.

And then use a MapView screen to:

- write a user-entered value to a Log Field of the new device; and
- read back and display Log Field values.

Orbit MapView Preliminaries

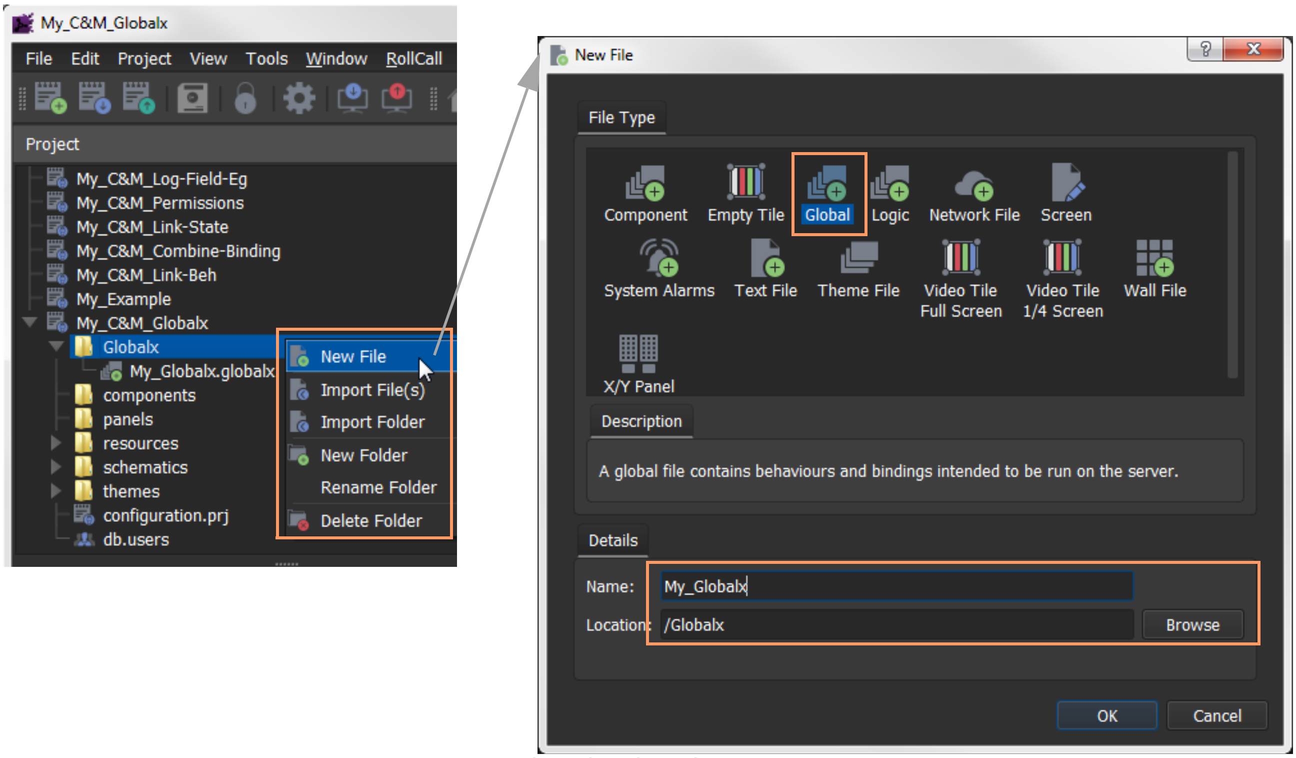
From the Orbit MapView project home screen:

1. Click ‘**Control and Monitoring** > **Properties**’ in the main menu.
2. Set the RollCall Domain.
   This must be identical to the domain used by the Orbit Services.
3. Click OK.
4. If the project is stored in a Git repository, click ‘**Tools** > **Options** > **Remotes**’
   and enter the URL of the repository holding the project.
5. Click ‘**RollCall** > **Network Map**’ to set up IP address of any Log server being used.

Build the Globalx File

In the Project pane, create a new.globalx file.

1. Create a new folder in the Orbit MapView project and call it ‘Globalx’.
   (Right-click on project name in Project View and select ‘New Folder’.)



A global file contains behaviours and bindings intended to be run on the server.

Name: My_Globalx

Location: /Globalx

Fig. 9-1: Creating a New File within the Orbit MapView Project

2 Create a new file of type 'global' in the new 'Globalx' folder.

3 Click **OK** in the New File dialog.

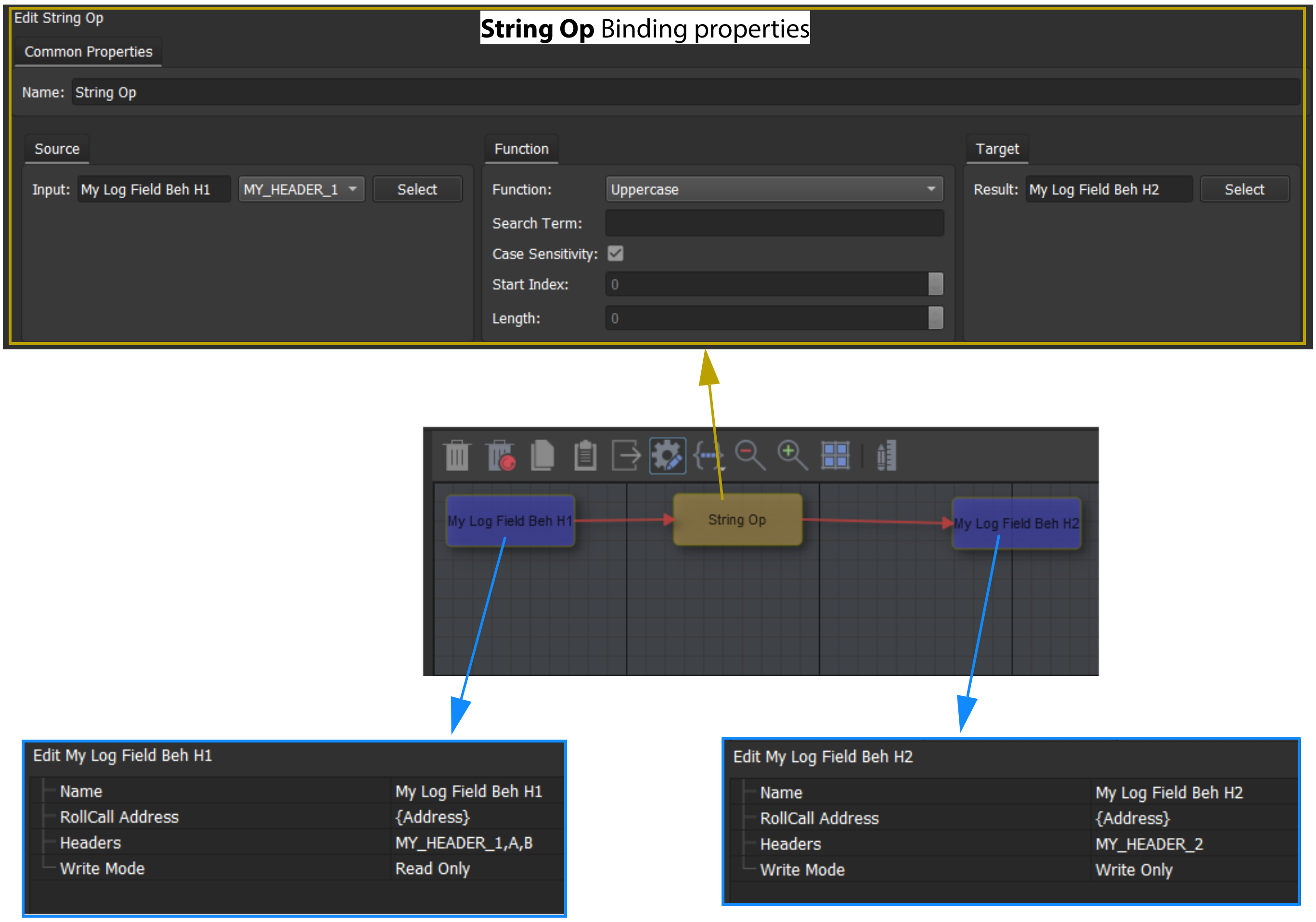
4 Open the new globalx file. (Double-click on it in the **Project View**.)

(The opened view is similar to the 'Behaviour and Bindings' graphical editor.)

5 Add two **Log Field** Behaviours and one **String Op** Binding,
and configure these as follow for an *unused* RollCall address (0000:E2:00 is used here):

- One **Log Field** Behaviour to read Log Field header 'MY_HEADER_1'.
- One **Log Field** Behaviour to write to Log Field header 'MY_HEADER_2'.
- **String Op** Binding to write an UPPER CASE version of MY_HEADER_1 value to MY_HEADER_2.

See Figure 9-2.



Edit String Op

String Op Binding properties

Common Properties

Name: String Op

Source

Input: My Log Field Beh H1 MY_HEADER_1 Select

Function

Function: Uppercase

Search Term:

Case Sensitivity: ☑

Start Index: 0

Length: 0

Target

Result: My Log Field Beh H2 Select

Edit My Log Field Beh H1

Name	My Log Field Beh H1
RollCall Address	{Address}
Headers	MY_HEADER_1,A,B
Write Mode	Read Only

Edit My Log Field Beh H2

Name	My Log Field Beh H2
RollCall Address	{Address}
Headers	MY_HEADER_2
Write Mode	Write Only

Fig. 9-2: Configuring Two Log Field Behaviours and one String Op Binding for Global File

6 Click **Save**.

This has created a file which will be executed by the Orbit MapView service when the project is run on the client PC. The globalx file will be executed in parallel to the project.

Note: The Orbit MapView service must be:

- pointed at the Orbit project; and
- restarted to pick up the latest Orbit project version.

Build a MapView Screen File

1. Create a new MapView screen file.
2. Add a **Text Edit** widget onto the screen schematic.
3. Select the **Text Edit** widget and add a **Local Value** Behaviour.
4. Deselect the **Text Edit** widget.
5. Right-click in the screen background and select 'Variables...'.
   The **Variables** dialog is shown.
6. Set the value of the variable named 'Address' to '0000:E2:00'.
7. Click **OK**.
8. Click **Save**.

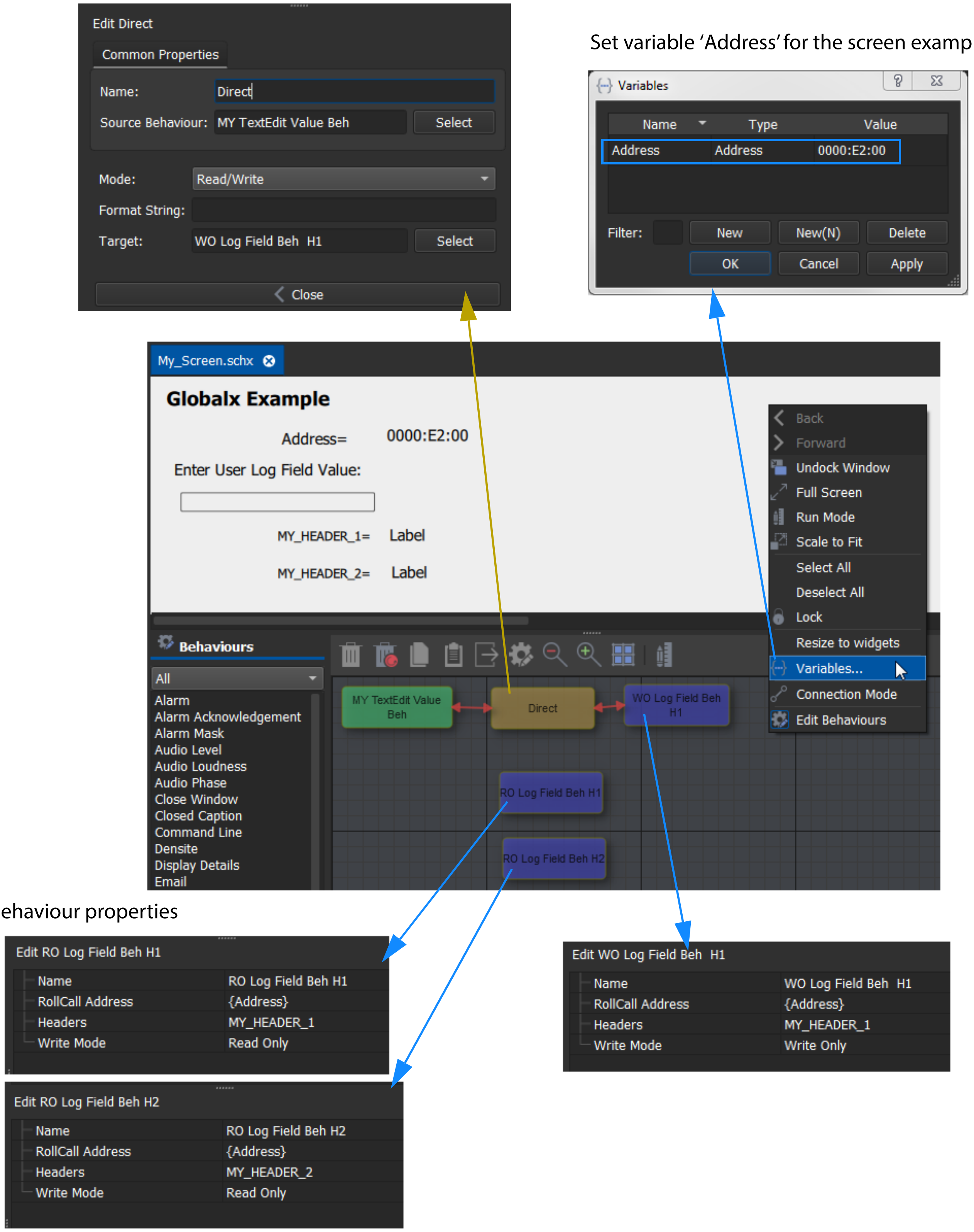
This has set up the RollCall address on the screen.

Next:

9. Add three **Log Field** Behaviours
   and configure these as follow for RollCall address 0000:E2:00:
   - One **Log Field** Behaviour to use the Text Edit value Behaviour and write its value to Log Field header 'MY_HEADER_1'.
   - One **Log Field** Behaviour to read Log Field header 'MY_HEADER_1'.
   - One **Log Field** Behaviour to read Log Field header 'MY_HEADER_2'.

   See Figure 9-3 on page 287.

10. Click **Save**.
11. Add two **Label** widgets
    and, using **Direct** Bindings, configure:
    - One label to show the value of Log Field MY_HEADER_1.
    - One label to show the value of Log Field MY_HEADER_2.
12. Click **Close**.
13. Click **Save**.



Set variable ‘Address’ for the screen example

Behaviour properties

Edit Direct

Common Properties

Name: Direct

Source Behaviour: MY TextEdit Value Beh

Mode: Read/Write

Format String:

Target: WO Log Field Beh H1

Variables

Name	Type	Value
Address	Address	0000:E2:00

Filter:

My_Screen.schx

Globalx Example

Address= 0000:E2:00

Enter User Log Field Value:

MY_HEADER_1= Label

MY_HEADER_2= Label

Edit RO Log Field Beh H1

Name	RO Log Field Beh H1
RollCall Address	{Address}
Headers	MY_HEADER_1
Write Mode	Read Only

Edit RO Log Field Beh H2

Name	RO Log Field Beh H2
RollCall Address	{Address}
Headers	MY_HEADER_2
Write Mode	Read Only

Edit WO Log Field Beh H1

Name	WO Log Field Beh H1
RollCall Address	{Address}
Headers	MY_HEADER_1
Write Mode	Write Only

Fig. 9-3: Configuring Address Variable and Three Log Field Behaviours

Configure Orbit Services

The Orbit services are set up via the Orbit Service Manager web page on the server computer.

Orbit Monitoring Service

1 Set **Client Domain** to the same domain as used by Orbit MapView.

2 Set a unique RollCall address.

3 If a Log Server is used, then set the Log Server Domain and IP address.

4 **Start** the service.

MapView Service

1 **Stop** the service.

2 Set **Client Domain** to the same domain as used by Orbit MapView.

3 Set a unique RollCall address.

If the Orbit project is stored in folders:

4 Set **Repository Type** to 'Local Folder'.

5 Set **Repository Path** to the path to the Orbit MapView project.

If the Orbit project is stored in a Git repository:

6 Set **Repository Type** to 'Git Repository'.

7 Set **Repository Path** to the Git repository URL to the Orbit MapView project.

For example, `http://172.10.70.101:8081/Git.Server/My_Project.git`

8 Click **Save Changes**.

9 **Start** the service.

Exercise the Globalx Example

The Orbit MapView screen can now be exercised:

1. When the Orbit services are pointed at the Orbit MapView project and started,
   the Orbit MapView service (server-side) loads the Orbit MapView project and executes the logic in the globalx file.

   For our example,
   this logic creates a virtual device within the Orbit MapView service with the specified RollCall address and with two Log Fields (MY_HEADER_1 and MY_HEADER_2).

Start exercising the MapView screen:

2. Click the 'Test Behaviour/Bindings' icon ( ).
3. Enter some (lower case) text into the Text Edit widget. Press Enter.

   The user-entered value is written to a Log Field of the virtual device.

   The globalx logic then writes an UPPER CASE version of the text to a second Log Field of the virtual device.

   Both Log Fields of the virtual device are reflected in the Map View screen.

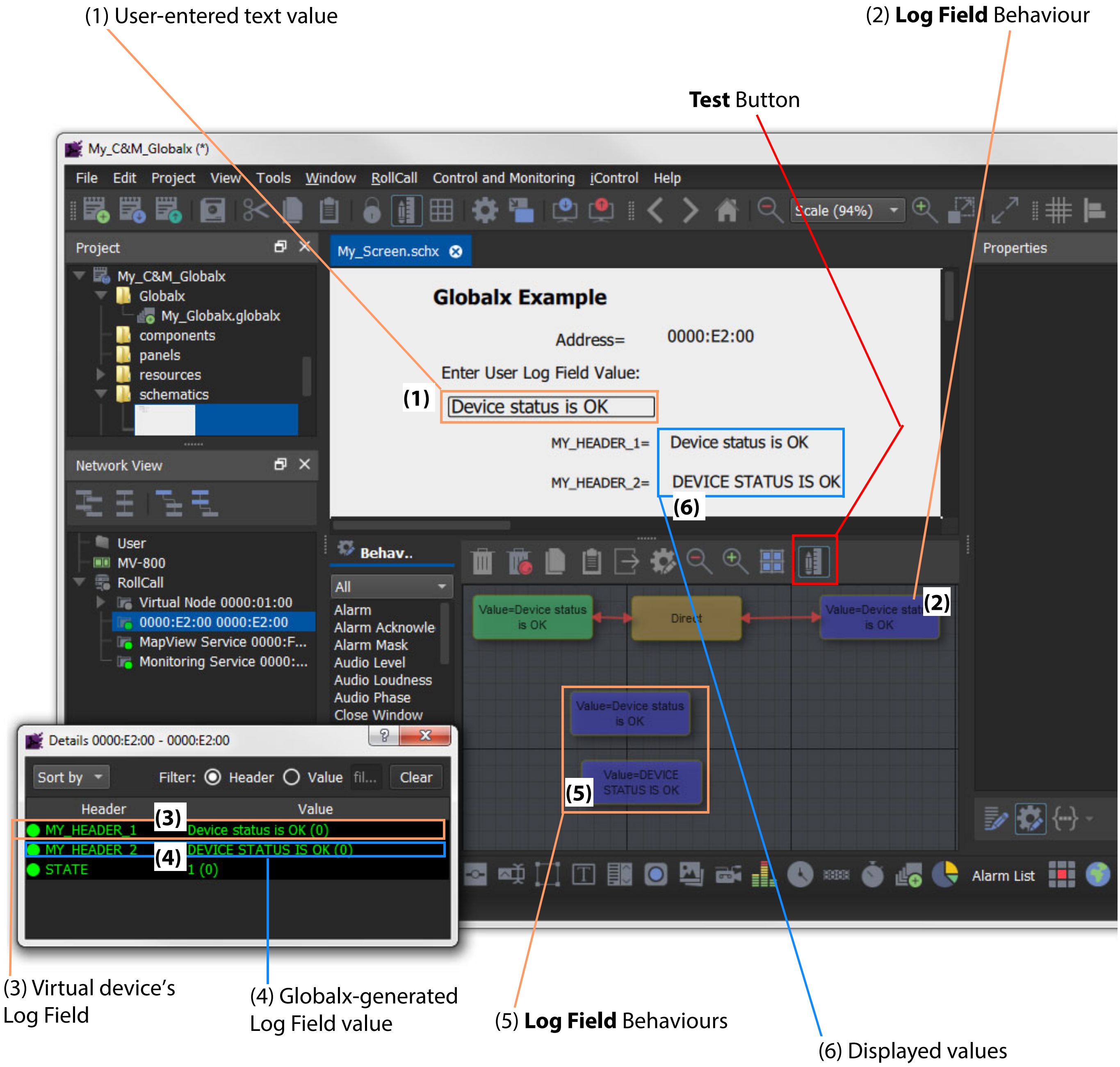
   See Figure 9-4 on page 290.

In Figure 9-4, the following is being exercised:

- The user-entered text value (1) is passed to the **Log Field** Behaviour (2).
- And it is then written to the Log Field header MY_HEADER_1 on the virtual device and shown in its 'Details' window (3).
- The globalx logic generates an UPPER CASE version of MY_HEADER_1's value and writes this to MY_HEADER_2 on the virtual device (4).
- **Log Field** Behaviours on the screen read the values of both headers (5)
  and these are shown on labels on the Orbit MapView screen (6).

Finally:

4. Click the **Test** Button again to stop exercising the screen.


(1) User-entered text value
(2) **Log Field** Behaviour
Test Button
(3) Virtual device's Log Field
(4) Globalx-generated Log Field value
(5) **Log Field** Behaviours
(6) Displayed values
My_C&M_Globalx (*)
File Edit Project View Tools Window RollCall Control and Monitoring iControl Help
Scale (94%)
Project
My_Screen.schx
Properties
Network View
Behav..
Alarm List

Globalx Example

Address= 0000:E2:00

Enter User Log Field Value:

Device status is OK

MY_HEADER_1= Device status is OK

MY_HEADER_2= DEVICE STATUS IS OK

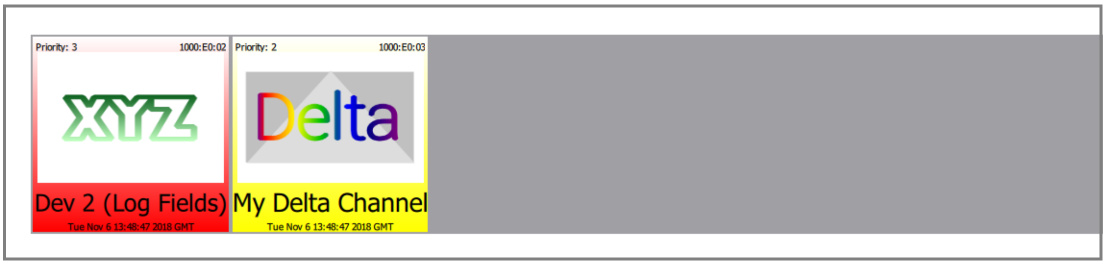
Details 0000:E2:00 - 0000:E2:00

Filter: ● Header ○ Value

Header	Value
MY_HEADER_1	Device status is OK (0)
MY_HEADER_2	DEVICE STATUS IS OK (0)
STATE	1 (0)

Fig. 9-4: Exercising the Global File Example

Example - Monitoring by Exception



This example creates a simple Orbit MapView project that demonstrates monitoring by exception. It uses:

- the **Monitor by Exception** widget;
- the Orbit services;
- **Label** widgets and the Angle Bracket < > syntax.
- Variables and screen re-use.
- Image file project resources.

The example assumes a system to be monitored has:

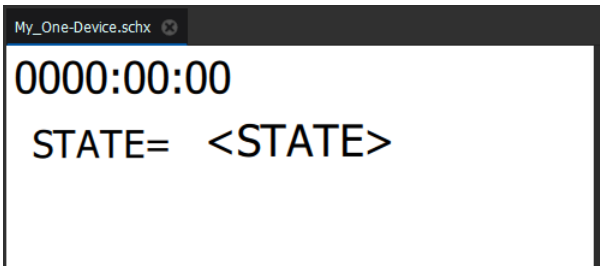
- Orbit services running.
- Several devices exist using RollCall addresses 1000:D0:01 to 1000:D0:08.
- Some unused RollCall addresses 1000:E0:01 to 1000:E0:08.
- Orbit Monitoring service processing Log Headers:
  MY_LOG_HEADER, MY_LOG_ANOTHER, and MY_LOG_YETANOTHER.

Build the Lower-Level Screens

First, create a new Orbit MapView project.

Build Screen to Monitor One Device

1. Create a new screen schematic, 'My_One-Device.schx'.
2. Add a **Label** widget and set:
   - Caption property to {Address}
3. Add a second **Label** Widget and set the caption property to
   - Caption property to STATE=
4. Add a third **Label** Widget and set the caption property to
   - Caption property to <STATE>
5. Click **Save**.



0000:00:00

STATE= <STATE>

Fig. 9-5: My_One-device.schx

Screen 'My_One-Device.schx' will monitor the 'state' at one RollCall address.

Build Screen to Monitor many Log Fields on One Device

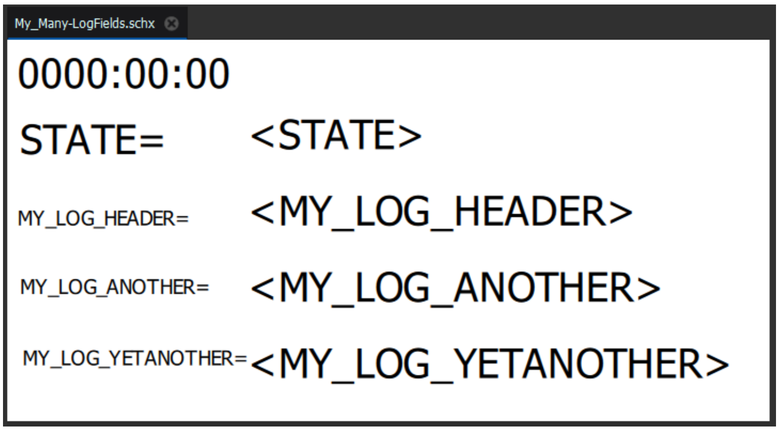
1 Create a new screen schematic, 'My_Many-LogFields.schx'.

2 Add a **Label** widget and set:

- Caption property to {Address}

3 Add more **Label** widgets, and configure them as shown in Figure 9-6.

4 Click **Save**.



0000:00:00

STATE= <STATE>

MY_LOG_HEADER= <MY_LOG_HEADER>

MY_LOG_ANOTHER= <MY_LOG_ANOTHER>

MY_LOG_YETANOTHER= <MY_LOG_YETANOTHER>

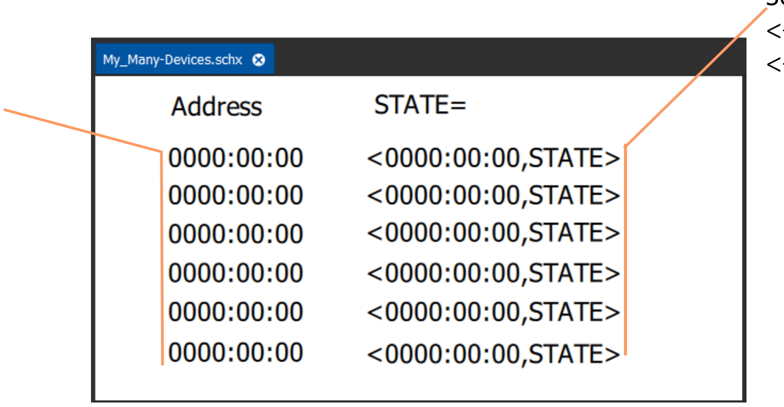
Fig. 9-6: My_Many-LogFields.schx

Screen 'My_Many-LogFields.schx' will monitor several Log Field at one RollCall address.

Build Screen to Monitor many Devices

1 Create a new screen schematic, 'My_Many-Devices.schx'.

2 Add **Label** widgets and configure as shown in Figure 9-7.



Address	STATE=
0000:00:00	<0000:00:00,STATE>
0000:00:00	<0000:00:00,STATE>
0000:00:00	<0000:00:00,STATE>
0000:00:00	<0000:00:00,STATE>
0000:00:00	<0000:00:00,STATE>
0000:00:00	<0000:00:00,STATE>

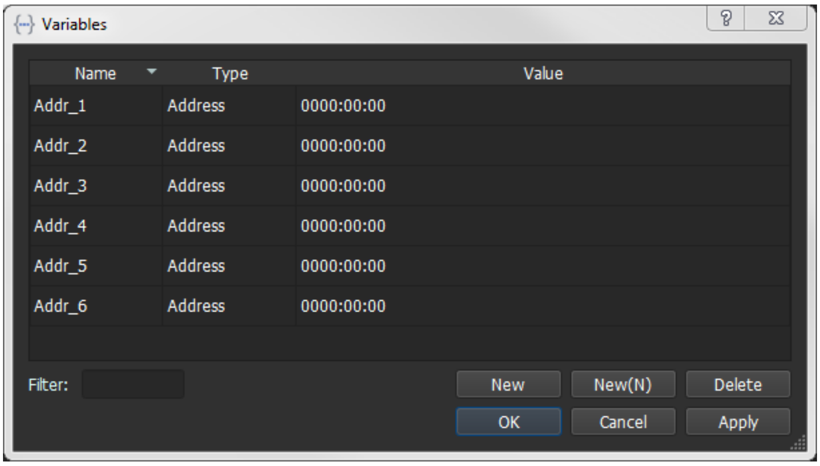
Set label caption properties to: {Addr_1}, {Addr_2},... etc.

Set label caption properties to: <{Addr_1},STATE>, <{Addr_2},STATE>... etc.

Fig. 9-7: My_Many-Devices.schx

3 Right-click on the screen background and select 'Variables...'
The **Variables** dialog is shown for declaring screen variables.

4 Add screen variables Addr_1 to Addr_6, as shown in Figure 9-8.
Leave the values set to their default 0000:00:00 values. (The actual values will be specified in a **Monitor by Exception** widget configuration.)



Name	Type	Value
Addr_1	Address	0000:00:00
Addr_2	Address	0000:00:00
Addr_3	Address	0000:00:00
Addr_4	Address	0000:00:00
Addr_5	Address	0000:00:00
Addr_6	Address	0000:00:00

Fig. 9-8: Screen Variables Declared

5 Click **OK**.

Note: The screen variables here are declared only. Their values will be set when configuring a **Monitor by Exception** widget.

6 Click **Save** to save the screen.

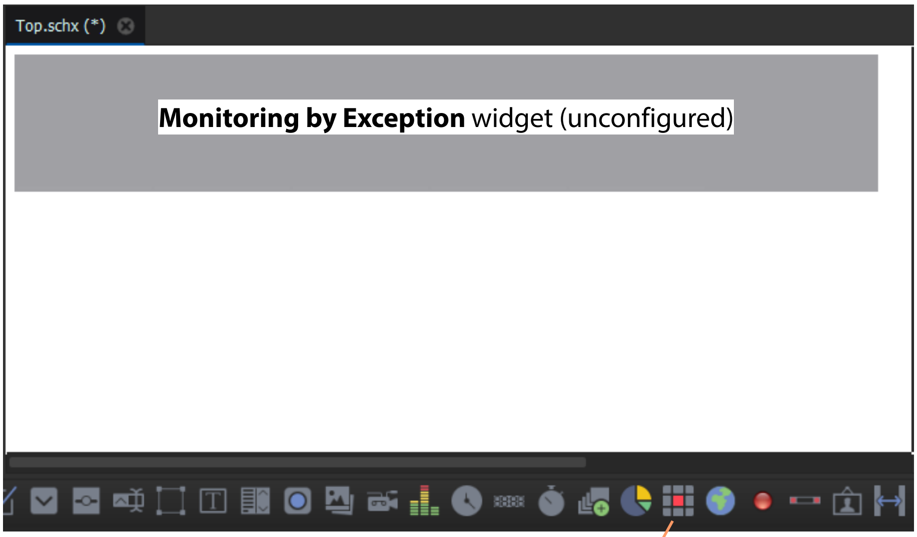
Screen 'My_Many-Devices.schx' will monitor the state at six RollCall address.

Top-Level Screen

Add a Monitor by Exception Widget:

1 Create a new screen schematic, 'Top.schx'.

2 Add a **Monitor by Exception** widget by clicking on the **Exception Monitoring** icon. See Figure 9-9.

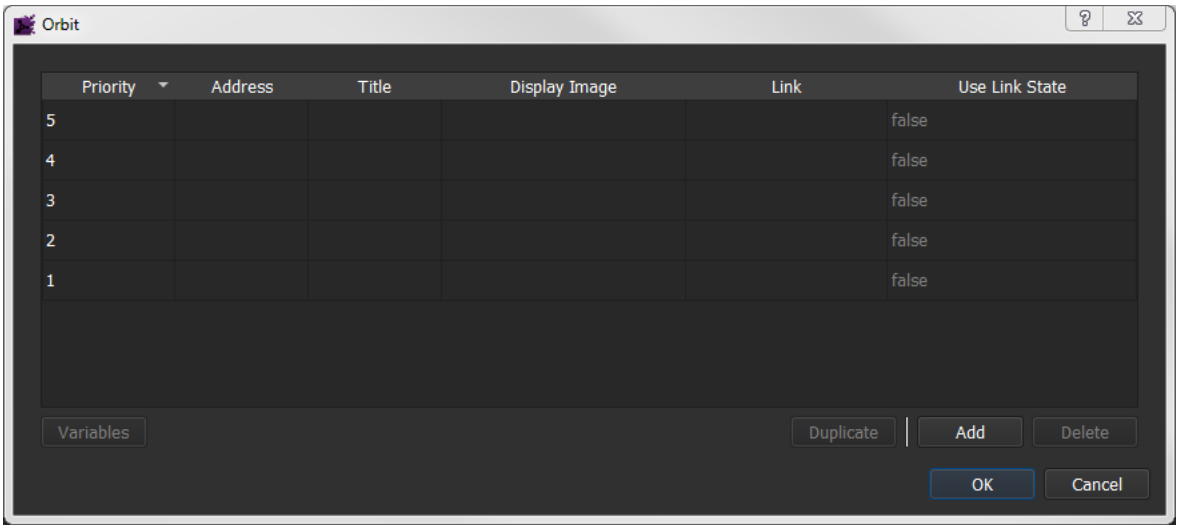


Exception Monitoring icon

Fig. 9-9: Exception Monitoring Widget Added to Screen

3 Select the widget
and, in the Properties box for the 'Item Configuration' property, click on the ⚙ icon.

The **Monitor by Exception** widget configuration dialog is shown. See Figure 9-10.



Priority	Address	Title	Display Image	Link	Use Link State
5					false
4					false
3					false
2					false
1					false

Fig. 9-10: Monitoring by Exception Widget Configuration Dialog

Configure the Monitor by Exception Widget

1 Click **Add** five times.
Five line items appear. See Figure 9-10.

Modify the first line item:

1 Enter an 'Address' of 1000:D0:01
(Note: This is an address of an existing device.)

2 Enter a 'Title' text.

Modify the second line item:

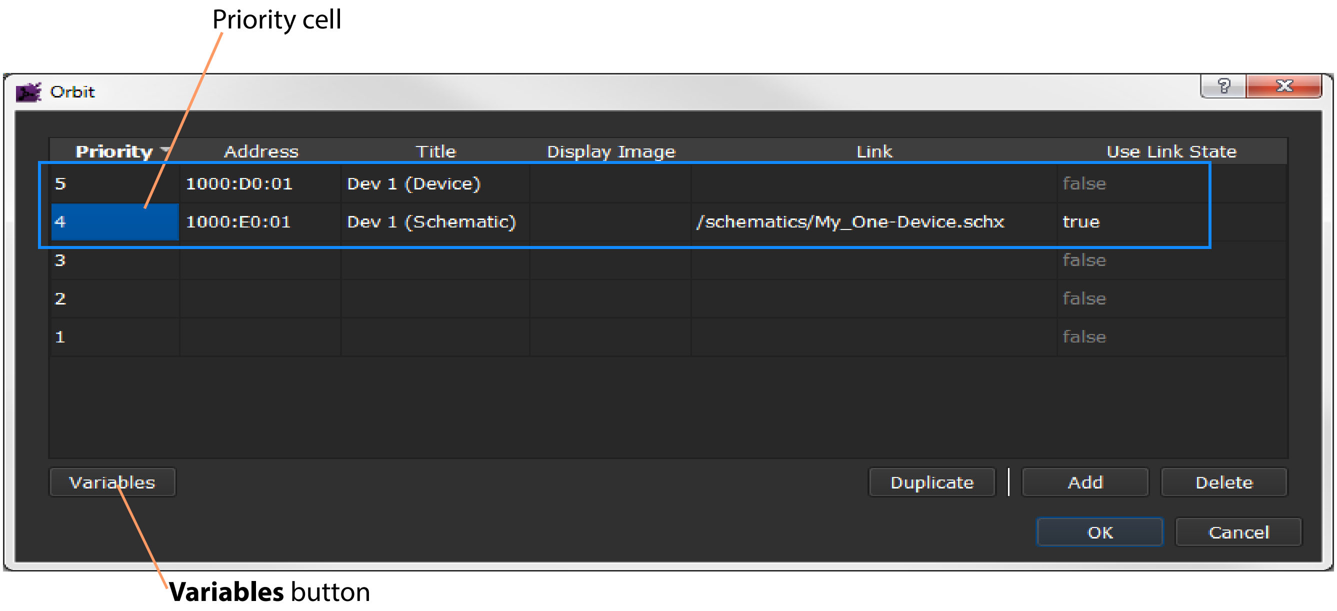
1 Enter an 'Address' of 1000:E0:01
(Note: This is an unused address.)

2 Enter a 'Title' text.

3 In the 'Link' column, add the path to the 'My_One-Device.schx' file (created earlier).

4 Set 'Use Link State' to 'True'.

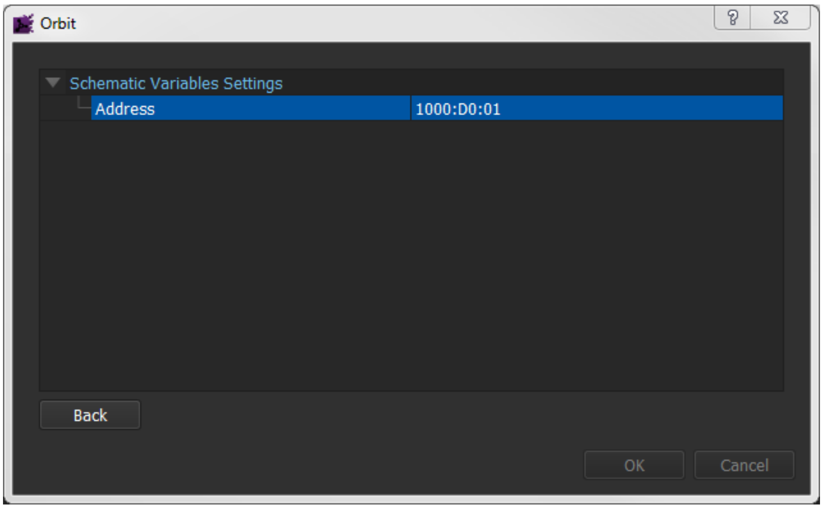
The first two lines have been configured, see Figure 9-11.



Priority	Address	Title	Display Image	Link	Use Link State
5	1000:D0:01	Dev 1 (Device)			false
4	1000:E0:01	Dev 1 (Schematic)		/schematics/My_One-Device.schx	true
3					false
2					false
1					false

Fig. 9-11: Two Line Configured

5 Select the ‘Priority’ cell in the second line item, and click on the **Variables** button. The screen variables setting dialog is shown, see Figure 9-12.



Schematic Variables Settings

Address	1000:D0:01

Fig. 9-12: Setting Screen Variables

The screen variables of the ‘Link’ screen are shown in a table.

6 Set the ‘Address’ variable to 1000:D0:01
(Note: This is the address of an existing device.)

7 Click **Back**.

The first two line items are now configured to monitor the same device (address 1000:D0:01) but by different Monitoring by Exception modes (schematic overview and device overview).

Modify the third line item:

1 Set the ‘Address’ variable to 1000:E0:02
(Note: This is an unused address.)

2 Enter ‘Title’ text.

3 Set 'Display Image' to an image file, one which has been previously created and imported into the project.

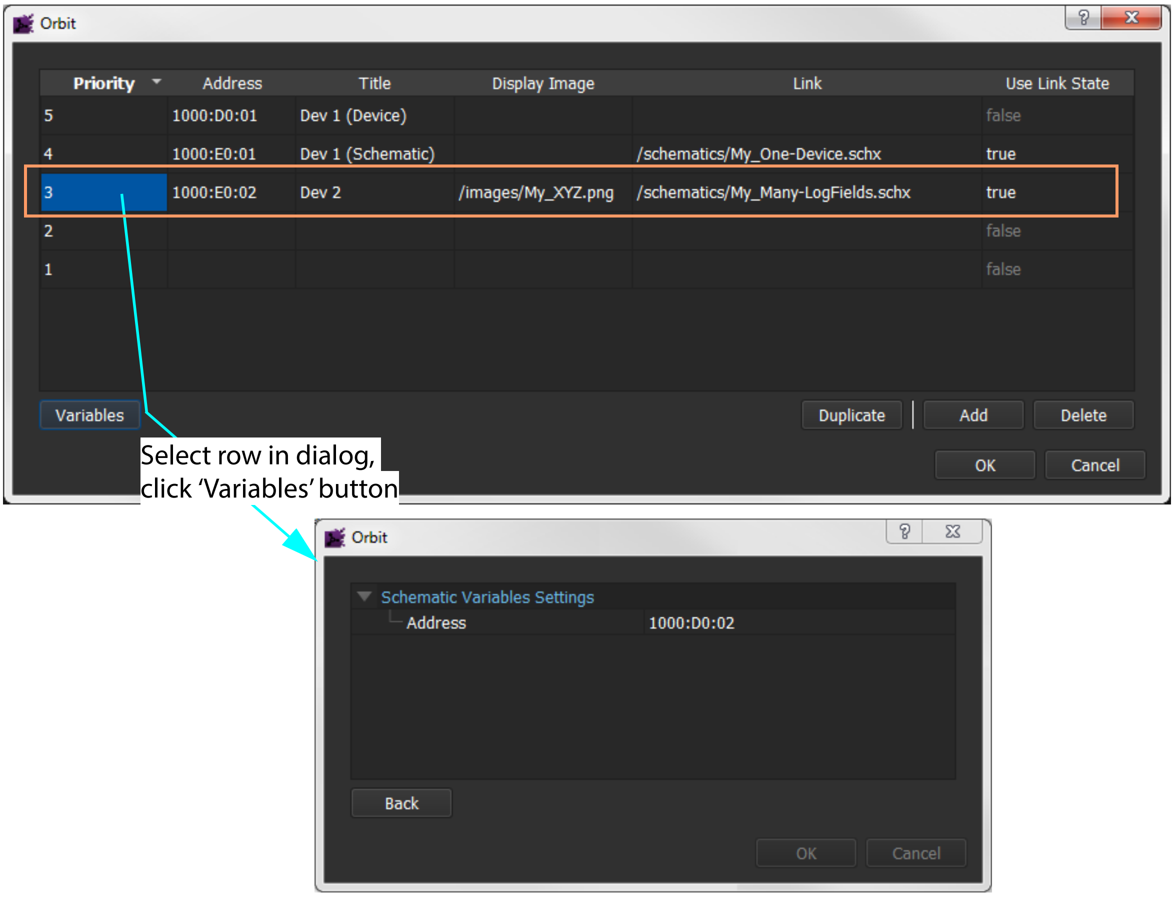
4 In the 'Link' column, add the path to the 'My_Many-LogFields.schx' file (created earlier).

5 Set 'Use Link State' to 'True'.

6 Select the 'Priority' cell in the third line item, and click on the **Variables** button. The screen variables setting dialog is shown.

7 Set the 'Address' variable to 1000:D0:02
(Note: This is the address of an existing device.)

This has configured the third line item to monitor several Log Fields on one device. See Figure 9-13.



Priority	Address	Title	Display Image	Link	Use Link State
5	1000:D0:01	Dev 1 (Device)			false
4	1000:E0:01	Dev 1 (Schematic)		/schematics/My_One-Device.schx	true
3	1000:E0:02	Dev 2	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2					false
1					false

Select row in dialog,
click 'Variables' button

Schematic Variables Settings
Address 1000:D0:02

Fig. 9-13: Third Line Item Configured

Modify the fourth line item:

1 Set the 'Address' variable to 1000:E0:03 (Note: This is an unused address.)

2 Enter 'Title' text.

3 Set 'Display Image' to an image file previously imported into the project.

4 In the 'Link' column, add the path to file 'My_Many-Devices.schx' (created earlier).

5 Set 'Use Link State' to 'True'.

6 Select the 'Priority' cell in the third line item, and click on the **Variables** button. The screen variables setting dialog is shown.

7 Set the listed variables to:

- Addr_1 to 1000:D0:03
- Addr_2 to 1000:D0:04
- Addr_3 to 1000:D0:05

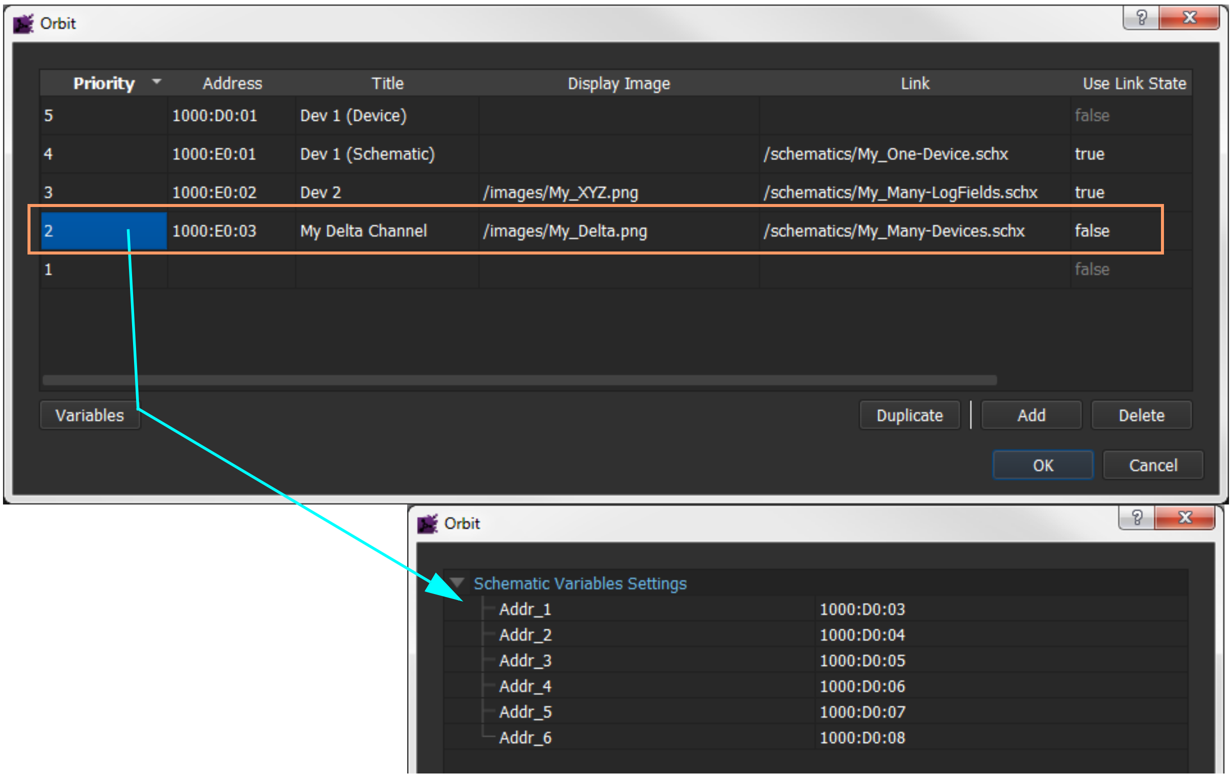
- Addr_4 to 1000:D0:06
- Addr_5 to 1000:D0:07
- Addr_6 to 1000:D0:08

(Note: These are addresses of existing devices.)

8 Click **Back**.

This has configured the fourth line item to monitor several devices. See Figure 9-14.

9 Click **OK**.



Priority	Address	Title	Display Image	Link	Use Link State
5	1000:D0:01	Dev 1 (Device)			false
4	1000:E0:01	Dev 1 (Schematic)		/schematics/My_One-Device.schx	true
3	1000:E0:02	Dev 2	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2	1000:E0:03	My Delta Channel	/images/My_Delta.png	/schematics/My_Many-Devices.schx	false
1					false

Schematic Variables Settings

Addr_1	1000:D0:03
Addr_2	1000:D0:04
Addr_3	1000:D0:05
Addr_4	1000:D0:06
Addr_5	1000:D0:07
Addr_6	1000:D0:08

Fig. 9-14: Fourth Line Item Configured

Modify the fifth line item:

1 Set the 'Address' variable to 1000:D0:09 (Note: This is an address of an existing device.)

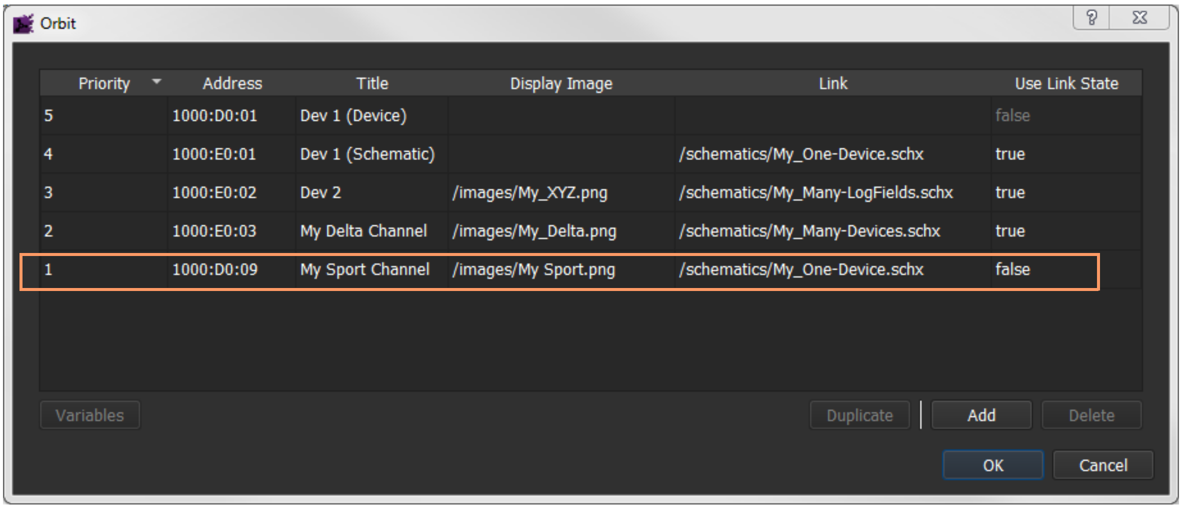
2 Enter 'Title' text.

3 Set 'Display Image' to an image file previously imported into the project.

4 In the 'Link' column, add the path to project file 'My_One-Device.schx' (created earlier).

5 Set 'Use Link State' to 'False'.

This has configured the fifth line item to monitor one device directly with a linked screen which does not contribute to the state of the monitoring, but is for information only. See Figure 9-15.



Priority	Address	Title	Display Image	Link	Use Link State
5	1000:D0:01	Dev 1 (Device)			false
4	1000:E0:01	Dev 1 (Schematic)		/schematics/My_One-Device.schx	true
3	1000:E0:02	Dev 2	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2	1000:E0:03	My Delta Channel	/images/My_Delta.png	/schematics/My_Many-Devices.schx	true
1	1000:D0:09	My Sport Channel	/images/My Sport.png	/schematics/My_One-Device.schx	false

Fig. 9-15: Fifth Line Item Configured

6 Click **OK**.

Further widget properties:

1 With the **Monitor by Exception** widget selected, in the **Properties** box, set:

- Property 'Extended Style > Sort/Filter Options > Hide Below State' to 0.

This is useful for debugging and will ensure Monitor by Exception icons are not hidden when their state is OK.

The designed top level screen is shown in Figure 9-16.

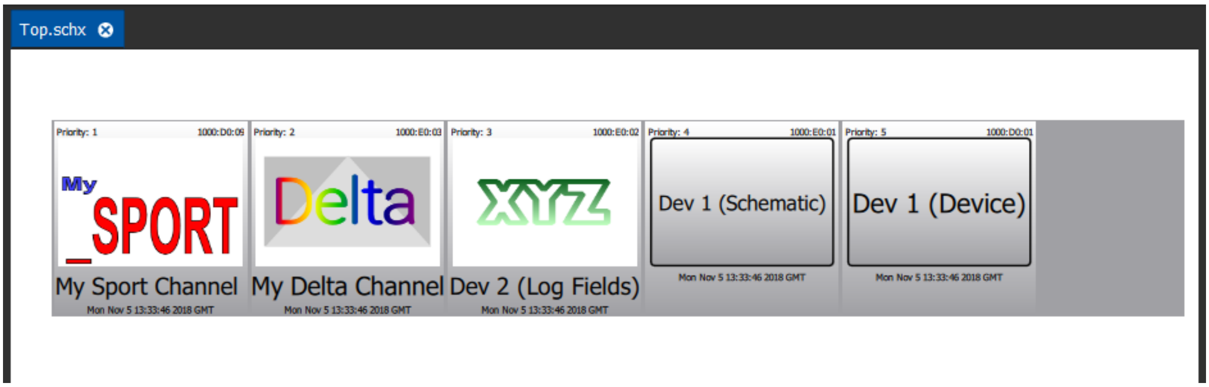


Fig. 9-16: Top Screen with Monitor by Exception Widget

2 Click **Save** to save the screen.

3 Click **Save Project** to save the whole project.

Exercise the Example

The Orbit Monitoring service should be running. Then:

1. Point the Orbit MapView service at the saved project and save the service settings.
2. Re-start the Orbit MapView service.
3. Open the top level screen and enter **Run Mode**.
   The **Monitor by Exception** widget now shows the status of the monitored items.

The on-screen appearance of the project is shown below for various **Monitor by Exception** widget settings etc.

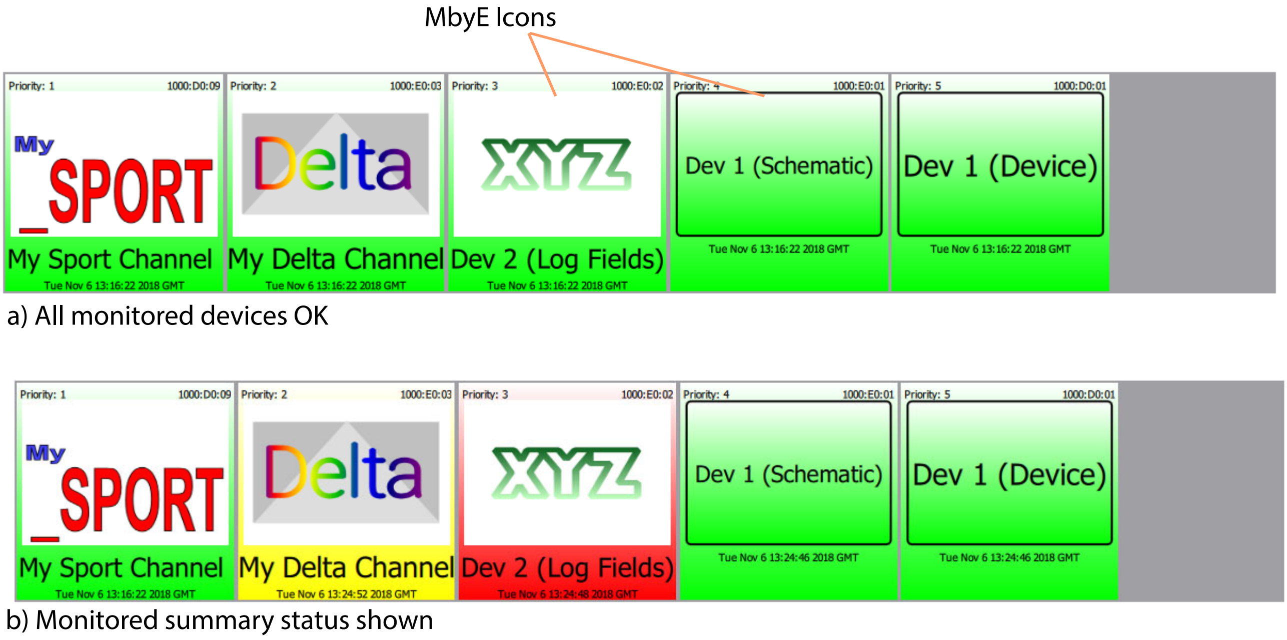
All MbyE Icons Shown

All **MbyE** icons are shown on the widget by setting:

- Widget property ‘Extended Style > Sort/Filter Options > Hide Below State’ = 0

Icon arrangement determined by:

- Primary Sort Order = Priority
- Secondary Sort Order = Date (Newest First)



a) All monitored devices OK

b) Monitored summary status shown

Fig. 9-17: Monitor by Exception Widget Showing all MbyE Icons

Hiding Some MbyE Icons

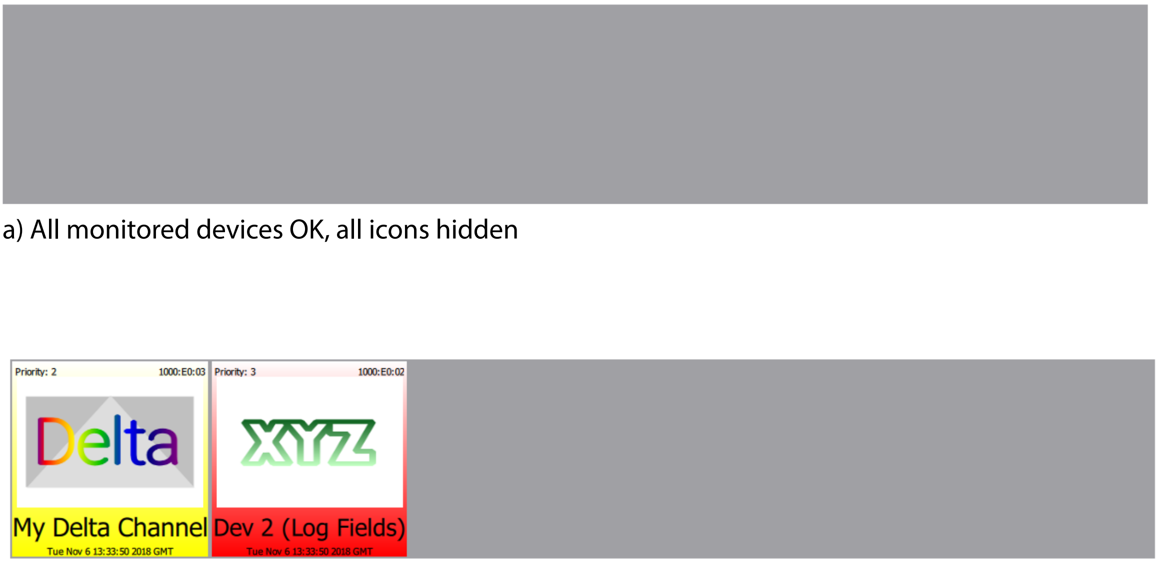
All **MbyE** icons are shown on the widget by setting:

- Widget property ‘Extended Style > Sort/Filter Options > Hide Below State’ = 10
  This hides icons with overview state of below 10. This includes ‘OK’ and ‘Masked’ state values.

Icon arrangement determined by:

- Primary Sort Order = Priority
- Secondary Sort Order = Date (Newest First)

See Figure 9-18.



a) All monitored devices OK, all icons hidden

b) Monitored summary status shown

Fig. 9-18: Monitor by Exception Widget Hiding Some MbyE Icons

All **MbyE** icons may be shown on the widget by setting:

- Widget property 'Extended Style > Sort/Filter Options > Hide Below State' = 80

This hides icons with overview state below 80 (includes 'Warning', 'OK' and 'Masked' state values).

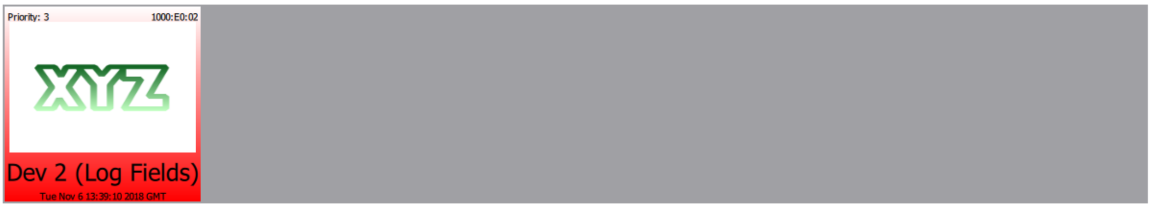


Fig. 9-19: Monitor by Exception Widget Hiding all but MbyE Icons with Fail State

MbyE Icons Shown in State Order

All **MbyE** icons are shown on the widget by setting:

- Widget property 'Extended Style > Sort/Filter Options > Hide Below State' = 0

Icon arrangement determined by:

- Primary Sort Order = State
- Secondary Sort Order = Priority

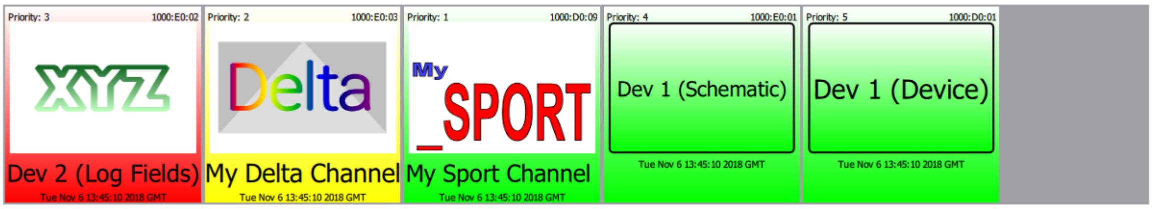
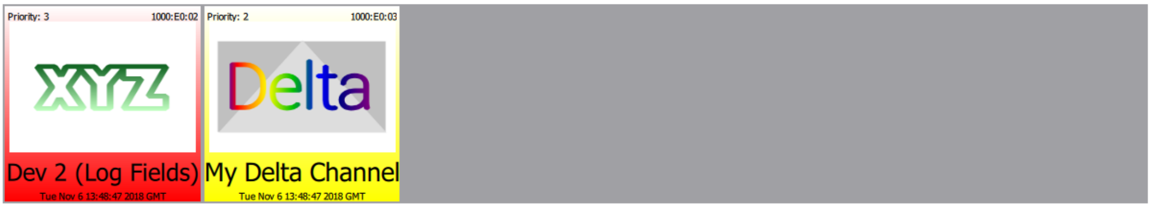


Fig. 9-20: Monitor by Exception Widget - MbyE Icons in Alarm State Order

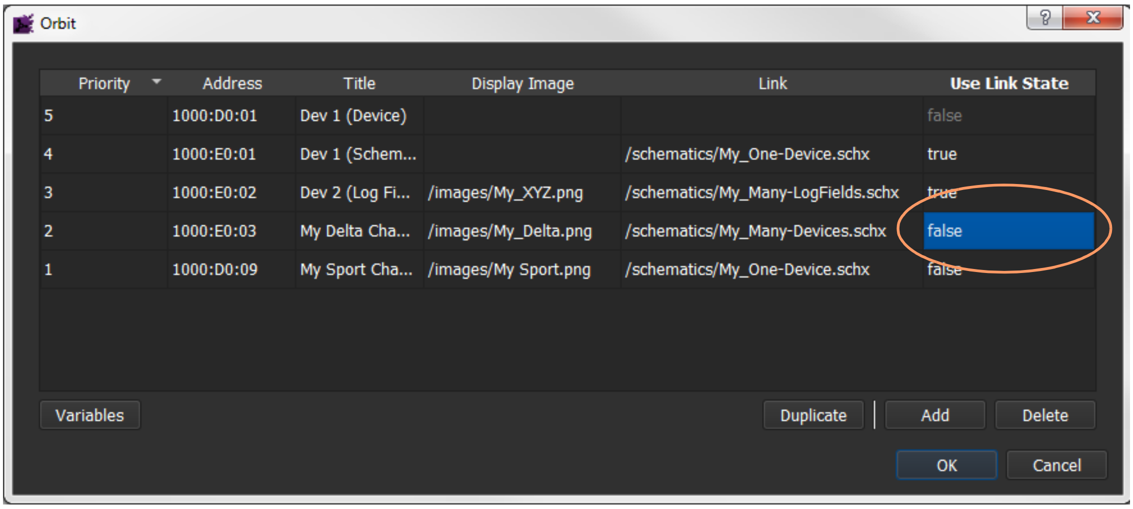


Hiding ‘OK’ states (‘Extended Style > Sort/Filter Options > Hide Below State’ = 10)

Fig. 9-21: Monitor by Exception Widget - MbyE Icons in State Order and Hiding Below 10

Exclude MbyE Icons from Screen Link State

If required, an **MbyE** icon item may be excluded from contributing to the overall ‘Link State’ of the screen by setting its corresponding ‘Use Link State’ setting to ‘False’. See Figure 9-22.



Priority	Address	Title	Display Image	Link	Use Link State
5	1000:D0:01	Dev 1 (Device)			false
4	1000:E0:01	Dev 1 (Schem...		/schematics/My_One-Device.schx	true
3	1000:E0:02	Dev 2 (Log Fi...	/images/My_XYZ.png	/schematics/My_Many-LogFields.schx	true
2	1000:E0:03	My Delta Cha...	/images/My_Delta.png	/schematics/My_Many-Devices.schx	false
1	1000:D0:09	My Sport Cha...	/images/My Sport.png	/schematics/My_One-Device.schx	false

Fig. 9-22: Excluding RollCall Address 1000:E0:03 from the Overall Screen Link State

Show MbyE Icons Depending on Their State

Icons can be shown according to their ‘State’ value by using the ‘Hide Below State’ **MbyE** widget property.
(I.e. in the **Properties** box for the **MbyE** widget, the property ‘Extended Style > Sort/Filter Options > Hide Below State’. **Note:** State value is in the range 0 to 100, see State Value 0 to 100, on page 17.)

For example:

- Set ‘Hide Below State’ = 48

  Icons with ‘Warning’ or ‘Fail’ status are shown.
- Set ‘Hide Below State’ = 98

  Icons with ‘Fail’ status are shown.

All **MbyE** icons may be shown on the **MbyE** widget by setting ‘Hide Below State’ = 0

Example - Monitoring the Rate of Change of a Value

Rate of Change alarm: 100

One Orbit Service is the Orbit Monitoring service. It can be configured to generate alarm messages from Log Field values in device log messages; the rate of change of a value is monitored. This is useful in cases where a small increase in a value is to be allowed, yet a large increase over a period of time requires a warning or an error to be generated.

For example, a CRC error count on a SDI video input: Normal connecting and disconnecting of a video input connector may result in some CRC messages. The user may wish to ignore these. But a large increase in a CRC error count over a given time *would* require some alarm to be generated.

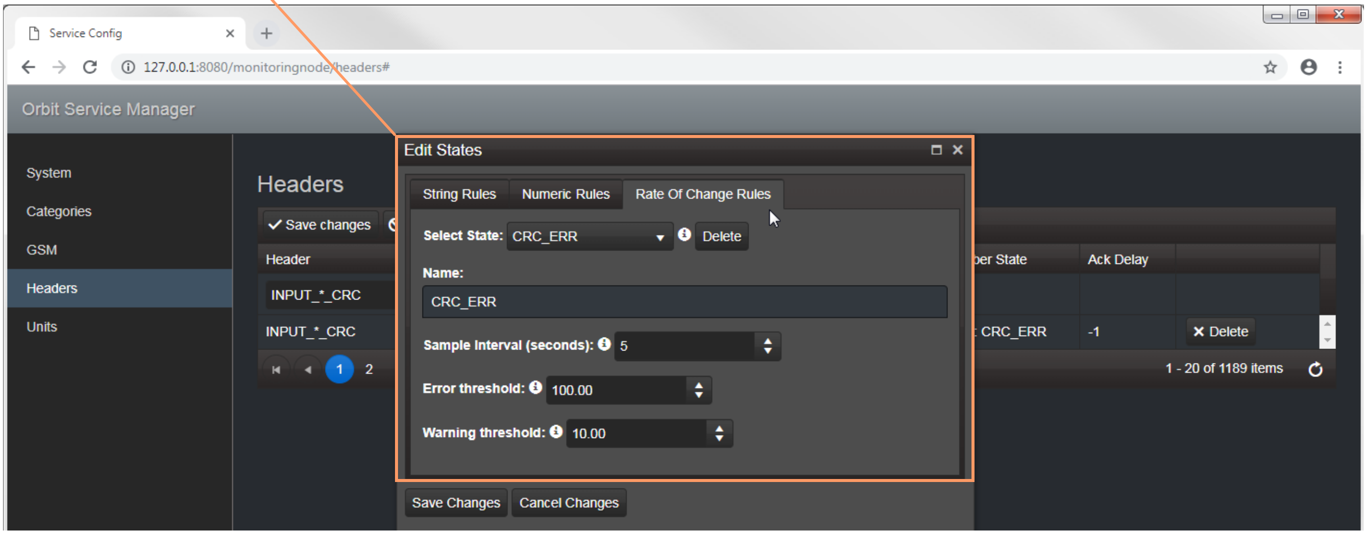
Note: The 'Rate of Change' (RoC) function monitors the *overall change* in a value over a period (sample interval). The overall change is compared against thresholds to determine the resulting state (OK, Warning, or Error).

Configuring the Orbit Monitoring Service

1. Open the Orbit Services manager and go to the 'Monitoring Service' page.
2. Open the 'Headers' page.
3. Click **Edit State Rules**,
   and click on the **Rate of Change Rules** tab.
4. Set up a rule for the Orbit Monitoring system. (See Figure 9-23 on page 303.)
5. Click **Save Changes**.

Edit states:

- **Select State –** Selects the rule to edit, or creates a new rule.
- **Delete –** Deletes the selected rule group (on 'Save Changes').
- **Name –** The name of the rule to be edited.
- **Sample Interval –** Defines the period over which the overall change in a value is calculated.
- **Error threshold –** Overall change threshold for error condition.
  If the overall change in a log field value (over the duration of the **Sample Interval**) is greater than **Error Threshold**, an error alarm state results.
- **Warning threshold –** Overall change threshold for warning condition.
  If the overall change in a log field value (over the duration of the **Sample Interval**) is greater than **Warning Threshold**, a warning alarm state results.
- (Note: Value *changes* below this threshold will be treated as 'OK'.)



Select State: CRC_ERR

Name: CRC_ERR

Sample Interval (seconds): 5

Error threshold: 100.00

Warning threshold: 10.00

Fig. 9-23: Edit a Rate of Change Rule

This has set up a 'Rate of Change' rule.

To apply this rule to a Log Field:

6 Select the Log Field Header.

7 Select the defined 'Rate of Change' rule in the 'Number State' column. See Figure 9-24 on page 304.

Note: The **Rate of Change** rule is only used for numeric Log Field values. When a Rate of Change rule (RoC) is selected, any string rules are ignored.

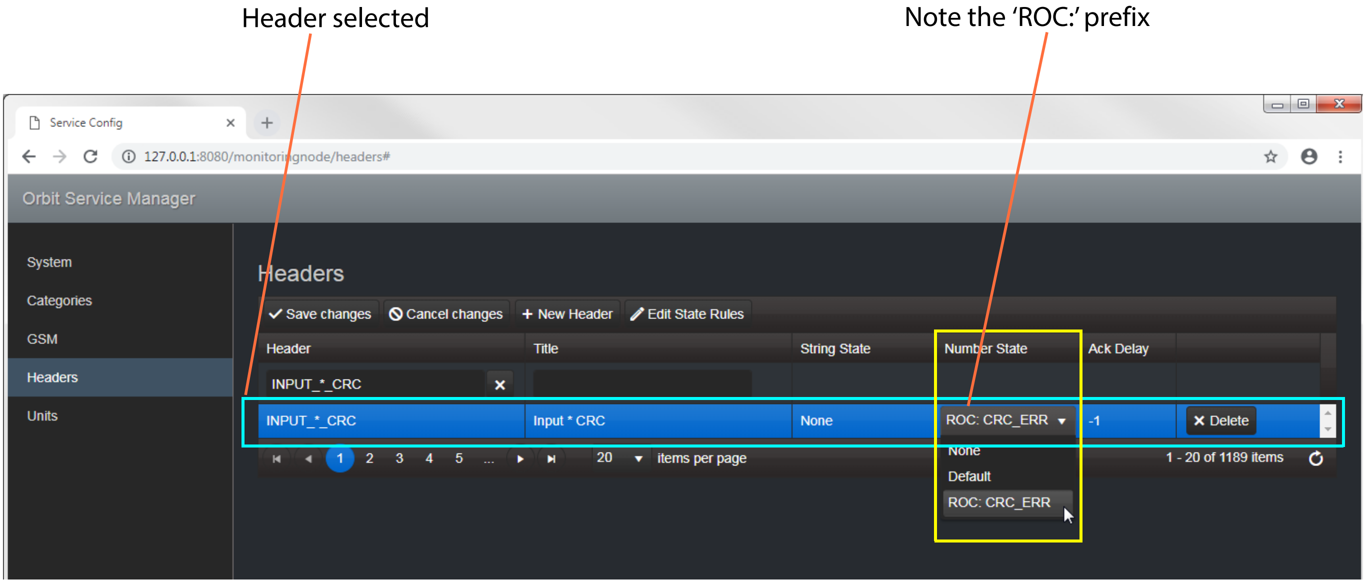


Fig. 9-24: Select the Rate of Change (ROC) Rules

8 Click **Save Changes**.

9 Stop and then restart the Monitoring service.

The Monitoring service will now monitor the defined Log Header (in this example, INPUT_*_CRC) over time to see if it any value changes exceed the 'Warning Threshold' or 'Error Threshold'.

The 'OK', 'Warning' or 'Error' state condition generated actively changes to reflect the latest Log Field value change. The alarm is not latched.

Monitoring Rate of Change with MapView

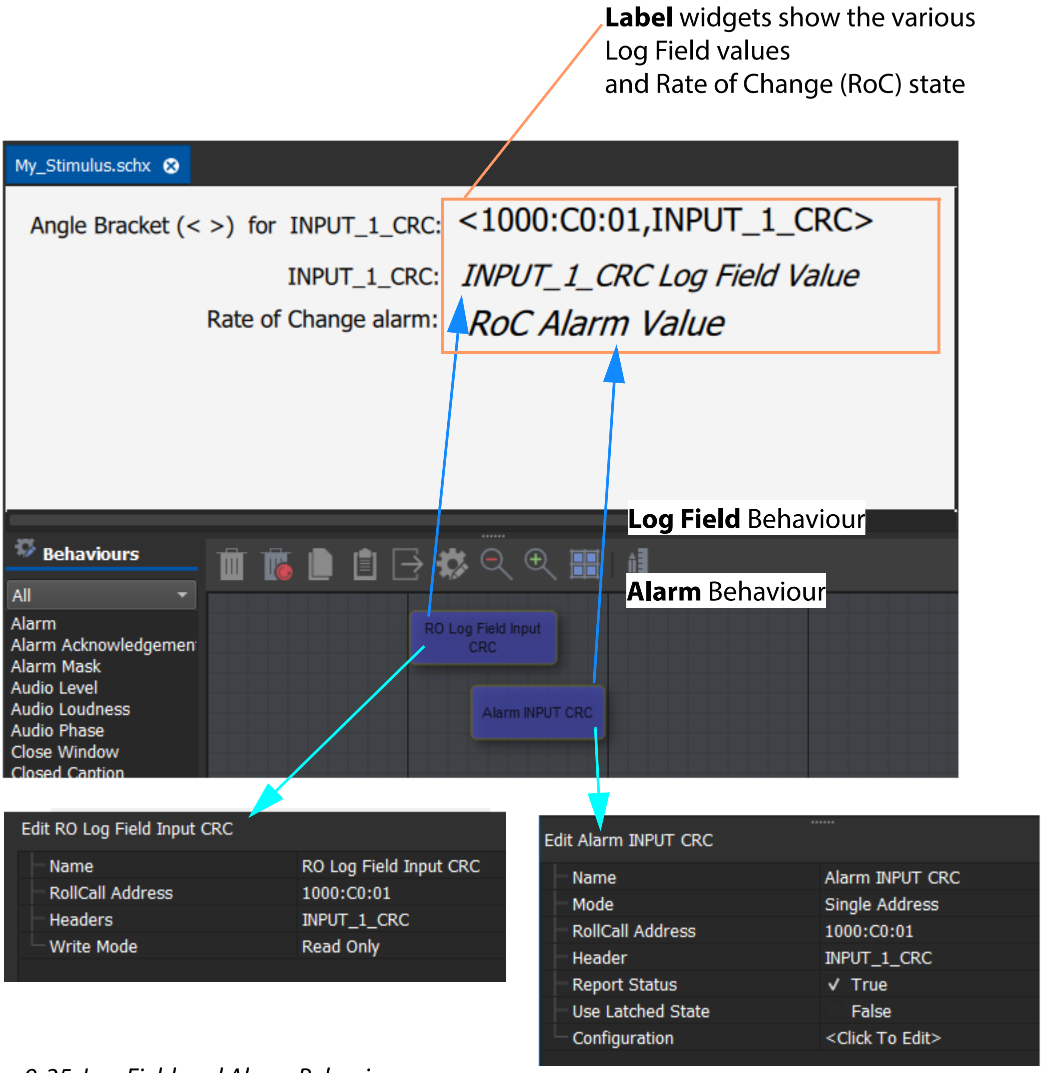
To use this ‘Rate of Change’ state information, from the services, in an Orbit MapView screen:

- Use an **Alarm** Behaviour to monitor a Log Field’s ‘Rate of Change’ alarm state. (The angle bracket syntax also works, < > .)

Build the Screen

In the MapView example below:

- a device at RollCall address 1000:C0:01 is generating INPUT_1_CRC Log messages with a changing value; and
- the Orbit Monitoring Service is monitoring the ‘Rate of Change’ the monitored value.



Label widgets show the various Log Field values and Rate of Change (RoC) state

Log Field Behaviour

Alarm Behaviour

Angle Bracket (< >) for INPUT_1_CRC: <1000:C0:01,INPUT_1_CRC>

INPUT_1_CRC: *INPUT_1_CRC Log Field Value*

Rate of Change alarm: *RoC Alarm Value*

Edit RO Log Field Input CRC

Name	RO Log Field Input CRC
RollCall Address	1000:C0:01
Headers	INPUT_1_CRC
Write Mode	Read Only

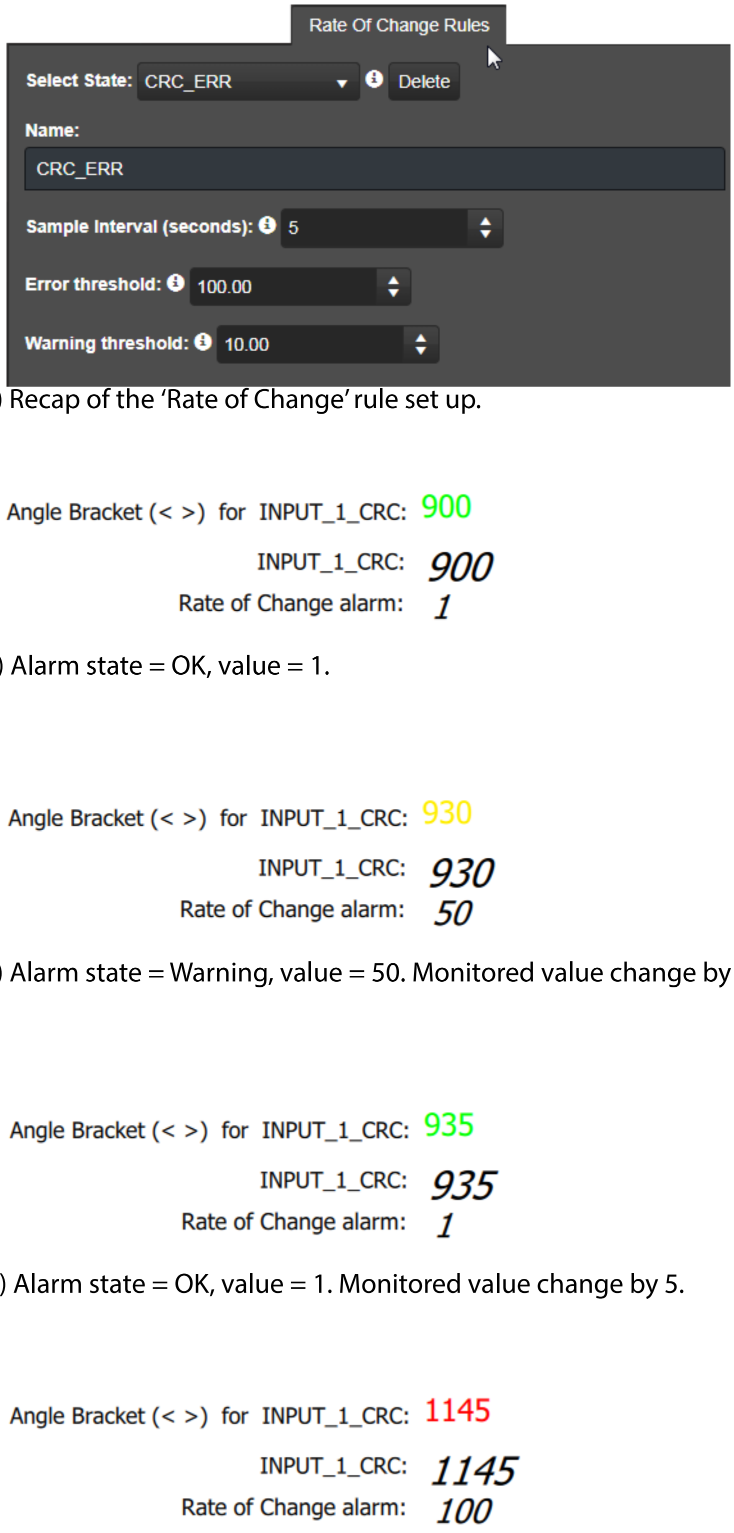
Edit Alarm INPUT CRC

Name	Alarm INPUT CRC
Mode	Single Address
RollCall Address	1000:C0:01
Header	INPUT_1_CRC
Report Status	✓ True
Use Latched State	False
Configuration	<Click To Edit>

Fig. 9-25: Log Field and Alarm Behaviours

Exercising the Screen

Running this screen while the INPUT_1_CRC Log Field values are changing produces the behavior shown in Figure 9-26.



Select State: CRC_ERR

Name: CRC_ERR

Sample Interval (seconds): 5

Error threshold: 100.00

Warning threshold: 10.00

a) Recap of the ‘Rate of Change’ rule set up.

Angle Bracket (< >) for INPUT_1_CRC: 900

INPUT_1_CRC: 900

Rate of Change alarm: 1

b) Alarm state = OK, value = 1.

Angle Bracket (< >) for INPUT_1_CRC: 930

INPUT_1_CRC: 930

Rate of Change alarm: 50

c) Alarm state = Warning, value = 50. Monitored value change by 30.

Angle Bracket (< >) for INPUT_1_CRC: 935

INPUT_1_CRC: 935

Rate of Change alarm: 1

d) Alarm state = OK, value = 1. Monitored value change by 5.

Angle Bracket (< >) for INPUT_1_CRC: 1145

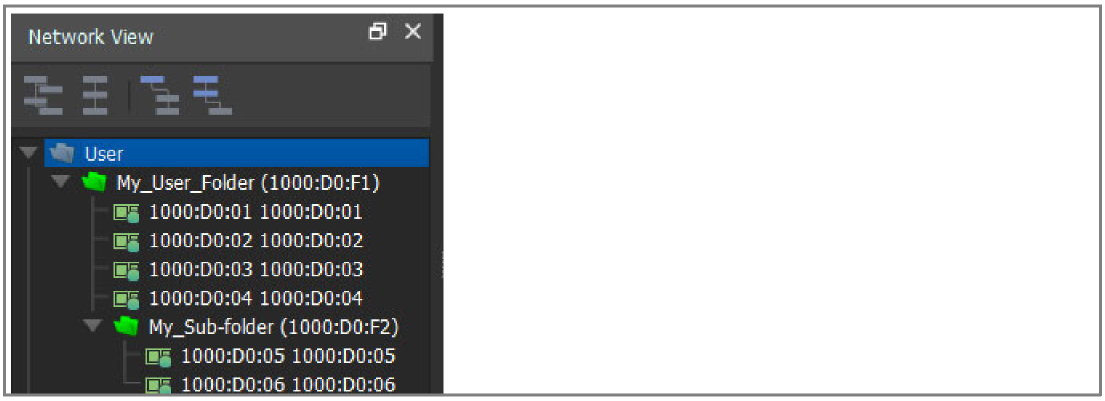
INPUT_1_CRC: 1145

Rate of Change alarm: 100

e) Alarm state = Error, value = 100. Monitored value change by 210.

Fig. 9-26: Monitored Log Field Value and Rate of Change Alarm state value

Example - 'Network View' User Folder and Virtual Alarms



User Folders in the **Network View** pane may also be used to generate an alarm for everything contained in the folder; this is a virtual alarm. A **User Folder** may contain a mixture of Grass Valley devices with differing alarm types (Densité/IQ). The virtual alarm is a hybrid alarm containing the aggregate state (worst case) of all the devices.

The virtual alarm is generated server-side by the Orbit MapView service. The virtual alarm is defined and configured in Orbit MapView at the **Network View** pane. When the alarm has been defined, the Orbit MapView project must be saved and be accessible by the Orbit MapView service.

The virtual alarm is associated with a (RollCall) address and is thus available to a running Orbit MapView project.

The action of virtual alarms is shown below with an example. The example has the Orbit client application connected to some devices.

Prepare a User Folder

In the Network View pane:

1 Right-click on the **User** item and select 'Create Folder'.

This creates a **User Folder**.

2 Right-click on the user folder and select 'Assign Address'.

3 Enter an unused RollCall address for this folder to use.

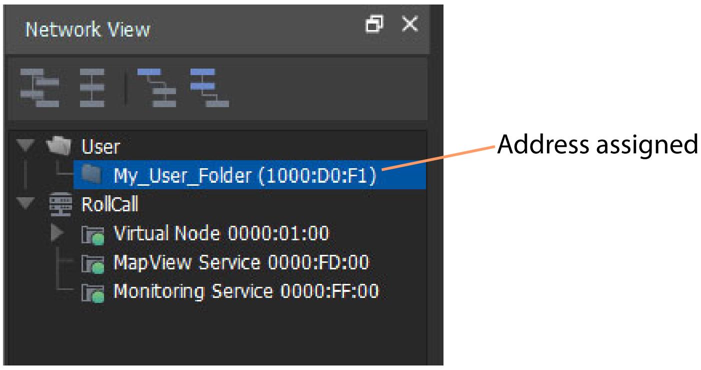
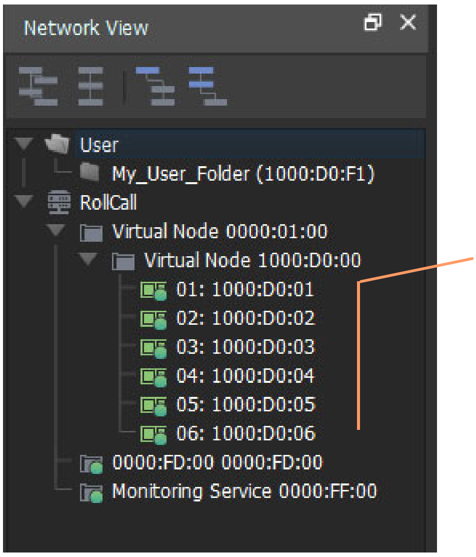

Network View
User
My_User_Folder (1000:D0:F1)
RollCall
Virtual Node 0000:01:00
MapView Service 0000:FD:00
Monitoring Service 0000:FF:00
Address assigned

Fig. 9-27: User Folder Assigned a RollCall Address

Note: The Orbit MapView service only calculates 'state' for user folders with an address assigned.

4 Expand the **Network View** pane to show the connected devices.



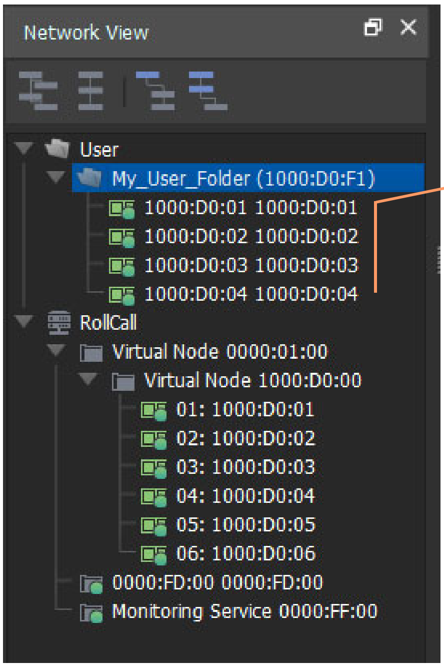
Network View

User
My_User_Folder (1000:D0:F1)
RollCall
Virtual Node 0000:01:00
Virtual Node 1000:D0:00
01: 1000:D0:01
02: 1000:D0:02
03: 1000:D0:03
04: 1000:D0:04
05: 1000:D0:05
06: 1000:D0:06
0000:FD:00 0000:FD:00
Monitoring Service 0000:FF:00

Connected devices

Fig. 9-28: Expanded Network View

5 To add devices,
drag and drop devices onto the newly-created user folder.



Network View

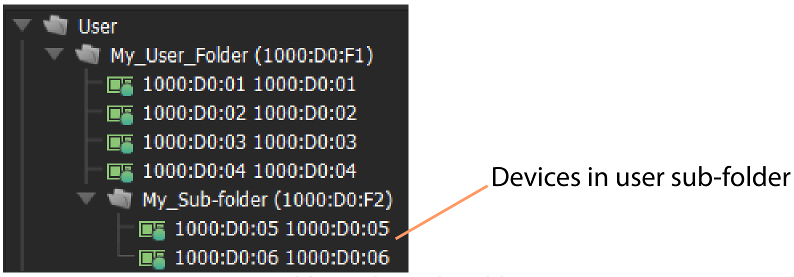
User
My_User_Folder (1000:D0:F1)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03
1000:D0:04 1000:D0:04
RollCall
Virtual Node 0000:01:00
Virtual Node 1000:D0:00
01: 1000:D0:01
02: 1000:D0:02
03: 1000:D0:03
04: 1000:D0:04
05: 1000:D0:05
06: 1000:D0:06
0000:FD:00 0000:FD:00
Monitoring Service 0000:FF:00

Devices in user folder

Fig. 9-29: Devices in User Folder

Note: User folders may contain sub-folders which contain devices, or further sub-folders.

Note: For common alarms which are used in more than one user folder, create a user sub-folder and drag/drop that into user folders where it is used. (For sub-folders, Orbit MapView service uses the calculated state of the sub-folder rather than re-calculating state from the sub-folder contents.)



User
My_User_Folder (1000:D0:F1)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03
1000:D0:04 1000:D0:04
My_Sub-folder (1000:D0:F2)
1000:D0:05 1000:D0:05
1000:D0:06 1000:D0:06

Devices in user sub-folder

Fig. 9-30: Devices in User Folder with a Sub-Folder

6 Save the project.

Exercise the User Folder

Using the Orbit Service Manager:

1 Set the Orbit MapView service to point at the Orbit MapView project.

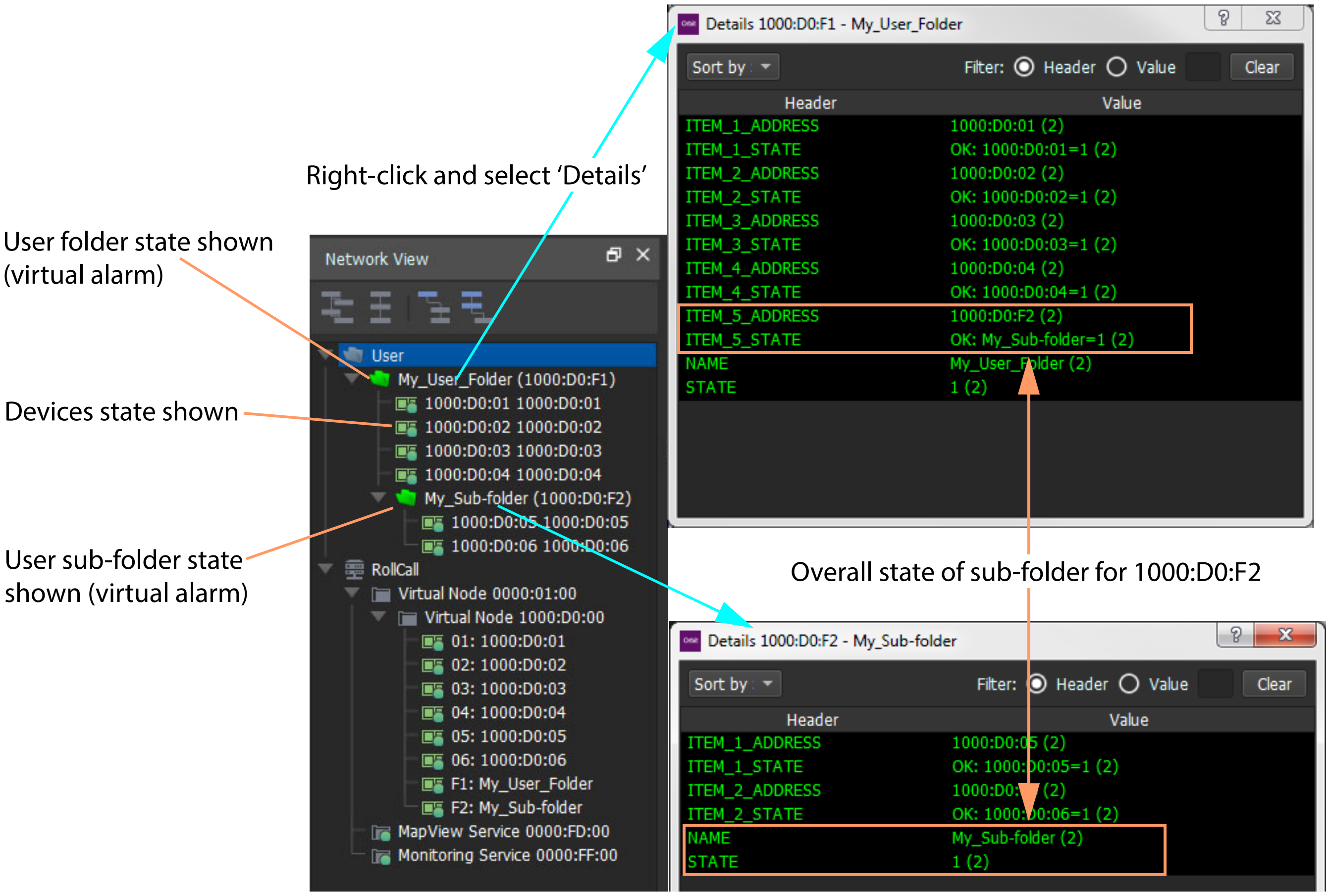
2 (Re-)Start the Orbit MapView service.

In the Orbit client window:

3 Run the Orbit MapView project. (Click **Run Mode** in the main tool bar.)

The Network View shows the state of connected devices and reflects the state of each user folder (virtual alarm) calculated by the Orbit MapView service. See Figure 9-31.

The log fields published by the Orbit MapView service for user folders can be seen in the 'Details' windows for each folder.



Right-click and select 'Details'

User folder state shown (virtual alarm)

Devices state shown

User sub-folder state shown (virtual alarm)

Overall state of sub-folder for 1000:D0:F2

Network View

User
My_User_Folder (1000:D0:F1)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03
1000:D0:04 1000:D0:04
My_Sub-folder (1000:D0:F2)
1000:D0:05 1000:D0:05
1000:D0:06 1000:D0:06
RollCall
Virtual Node 0000:01:00
Virtual Node 1000:D0:00
01: 1000:D0:01
02: 1000:D0:02
03: 1000:D0:03
04: 1000:D0:04
05: 1000:D0:05
06: 1000:D0:06
F1: My_User_Folder
F2: My_Sub-folder
MapView Service 0000:FD:00
Monitoring Service 0000:FF:00

Details 1000:D0:F1 - My_User_Folder

Sort by Filter: ● Header ○ Value Clear

Header	Value
ITEM_1_ADDRESS	1000:D0:01 (2)
ITEM_1_STATE	OK: 1000:D0:01=1 (2)
ITEM_2_ADDRESS	1000:D0:02 (2)
ITEM_2_STATE	OK: 1000:D0:02=1 (2)
ITEM_3_ADDRESS	1000:D0:03 (2)
ITEM_3_STATE	OK: 1000:D0:03=1 (2)
ITEM_4_ADDRESS	1000:D0:04 (2)
ITEM_4_STATE	OK: 1000:D0:04=1 (2)
ITEM_5_ADDRESS	1000:D0:F2 (2)
ITEM_5_STATE	OK: My_Sub-folder=1 (2)
NAME	My_User_Folder (2)
STATE	1 (2)

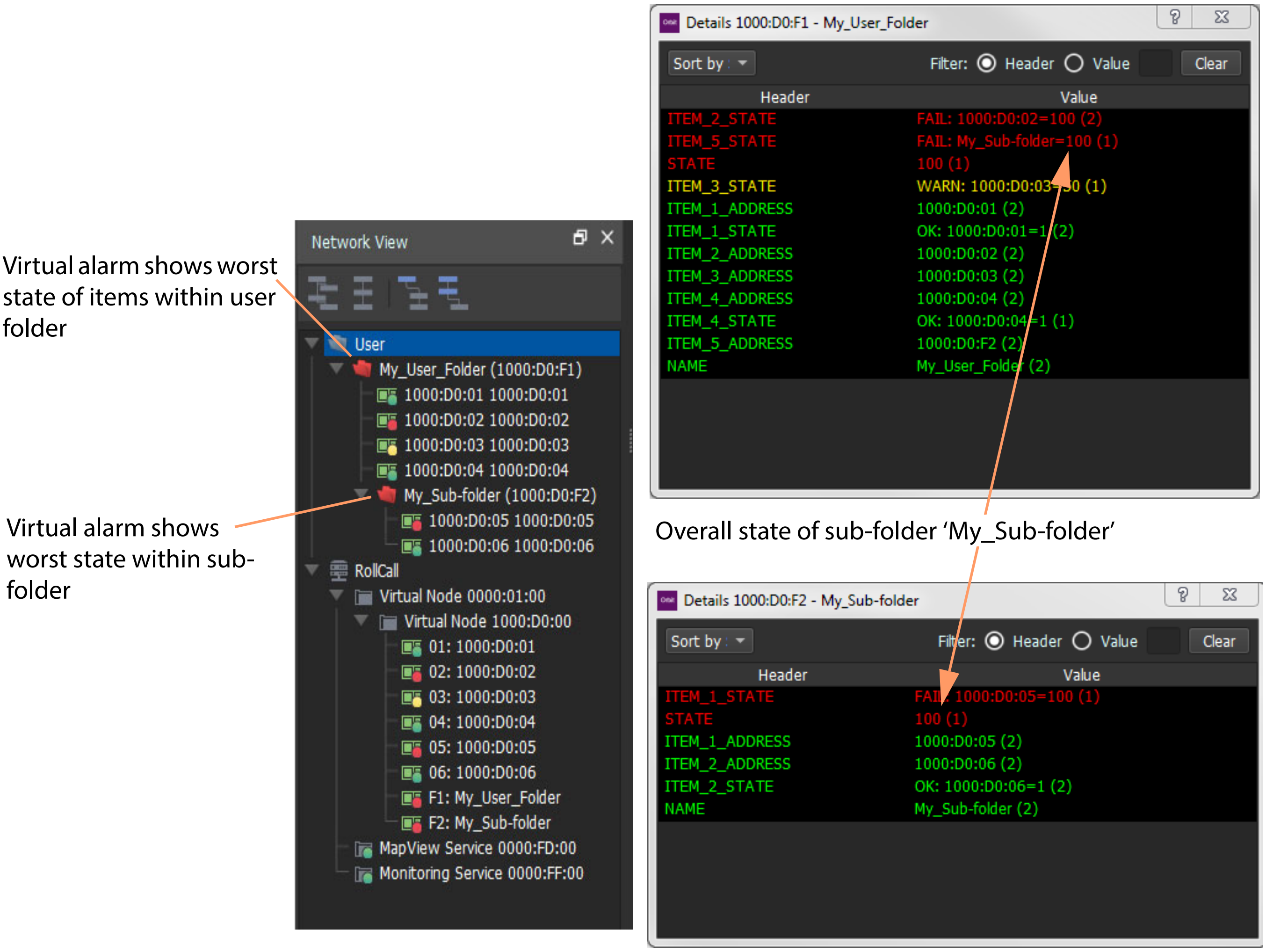
Details 1000:D0:F2 - My_Sub-folder

Sort by Filter: ● Header ○ Value Clear

Header	Value
ITEM_1_ADDRESS	1000:D0:05 (2)
ITEM_1_STATE	OK: 1000:D0:05=1 (2)
ITEM_2_ADDRESS	1000:D0:06 (2)
ITEM_2_STATE	OK: 1000:D0:06=1 (2)
NAME	My_Sub-folder (2)
STATE	1 (2)

Fig. 9-31: Virtual Alarm Shown in Network View

4 When devices send out ‘warning’ or ‘fail’ state log messages, the device state is correspondingly shown in the user folder.
Sub-folder states are reflected in their parent folder’s state. (Alarms ‘ripple upwards’.)
See Figure 9-32.



Virtual alarm shows worst state of items within user folder

Virtual alarm shows worst state within sub-folder

Overall state of sub-folder ‘My_Sub-folder’

Details 1000:D0:F1 - My_User_Folder

Header	Value
ITEM_2_STATE	FAIL: 1000:D0:02=100 (2)
ITEM_5_STATE	FAIL: My_Sub-folder=100 (1)
STATE	100 (1)
ITEM_3_STATE	WARN: 1000:D0:03=50 (1)
ITEM_1_ADDRESS	1000:D0:01 (2)
ITEM_1_STATE	OK: 1000:D0:01=1 (2)
ITEM_2_ADDRESS	1000:D0:02 (2)
ITEM_3_ADDRESS	1000:D0:03 (2)
ITEM_4_ADDRESS	1000:D0:04 (2)
ITEM_4_STATE	OK: 1000:D0:04=1 (1)
ITEM_5_ADDRESS	1000:D0:F2 (2)
NAME	My_User_Folder (2)

Details 1000:D0:F2 - My_Sub-folder

Header	Value
ITEM_1_STATE	FAIL: 1000:D0:05=100 (1)
STATE	100 (1)
ITEM_1_ADDRESS	1000:D0:05 (2)
ITEM_2_ADDRESS	1000:D0:06 (2)
ITEM_2_STATE	OK: 1000:D0:06=1 (2)
NAME	My_Sub-folder (2)

Fig. 9-32: Virtual Alarm with Device Warning and Failure States

Masking Alarms from the Network View Pane

Masking the alarms we can see the state of the **User Folder** is updated.

To mask an alarm:

- Right-click on an item in a **User Folder** and select 'Mask'.
  The following masking options are available:
  - **Mask Unit**.
  - **Mask Until Green**.
  - **Mask Until Time**.

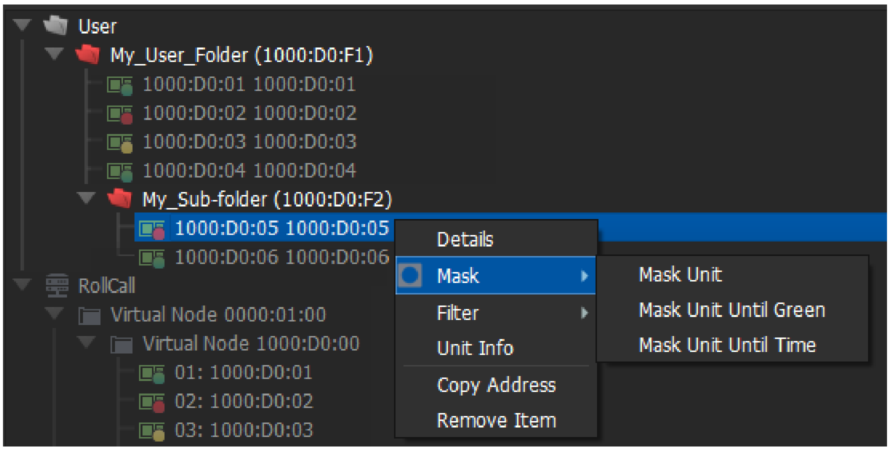
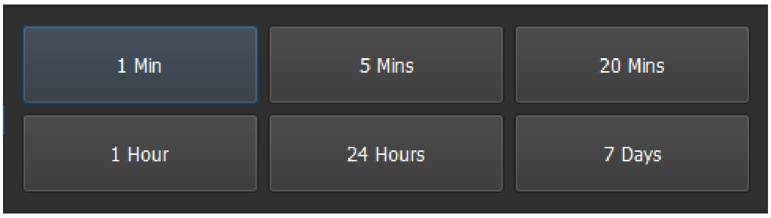
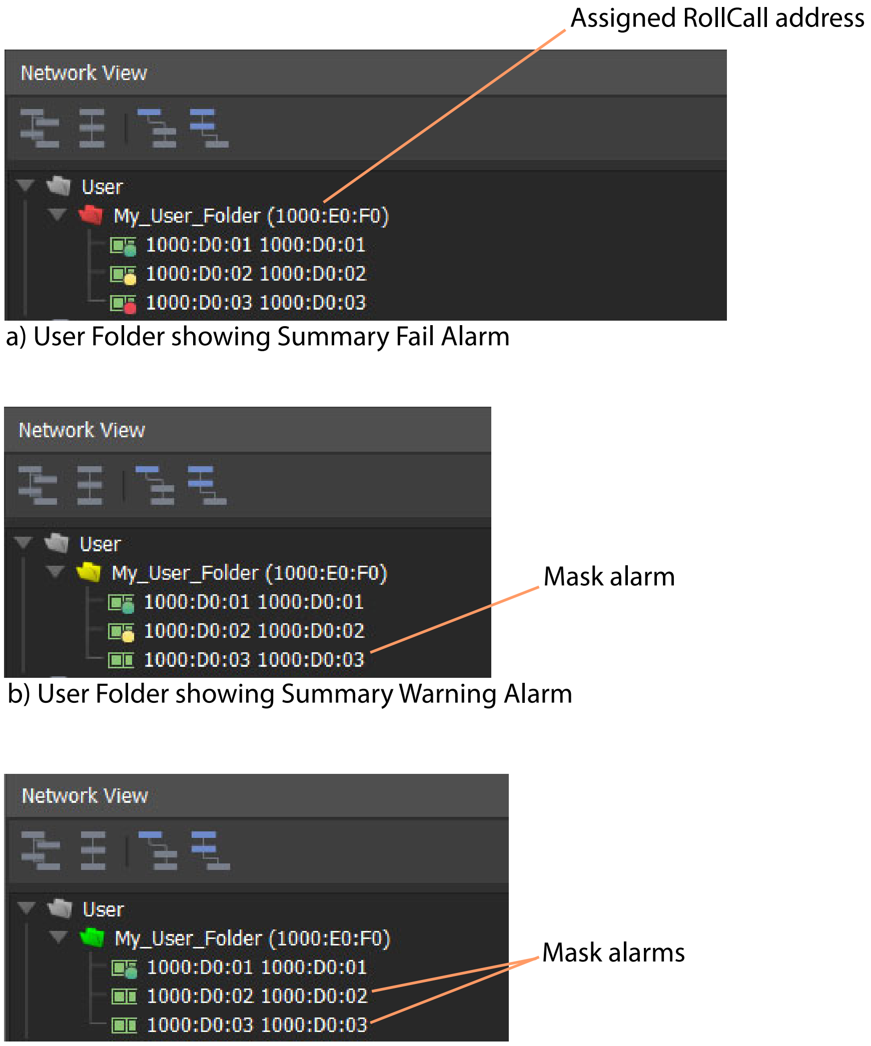


Fig. 9-33: Mask Right-click Menu Item

Table 9-1: Masking Right-click menu Items

Property	Description
Mask Unit / **Unmask Unit**	Toggle control. Select to exclude (mask) or include (unmask) the device from the user folder Virtual Alarm calculation.
Mask Unit Until Green	Select to exclude the device from the user folder Virtual Alarm calculation until the unit has an OK state. Once a device OK state is achieved, the device is included into the calculation.
Mask Unit Until Time	Select to exclude the device from the user folder Virtual Alarm calculation for a period of time. Select the required pre-set period from the panel shown:  Once the time period has elapsed, the device alarms are monitored once more by the virtual alarm mechanism. **Note:** The masking time periods are set up in Orbit via 'Tools > Options > Monitoring'. in the 'Masking' tab



Assigned RollCall address

Network View

User
My_User_Folder (1000:E0:F0)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03

a) User Folder showing Summary Fail Alarm

Network View

User
My_User_Folder (1000:E0:F0)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03

Mask alarm

b) User Folder showing Summary Warning Alarm

Network View

User
My_User_Folder (1000:E0:F0)
1000:D0:01 1000:D0:01
1000:D0:02 1000:D0:02
1000:D0:03 1000:D0:03

Mask alarms

c) User Folder showing Summary OK Alarm

Fig. 9-34: Masking Alarms in User Folder:
a) A Summary Fail Virtual Alarm.
b) A Summary Warning Virtual Alarm.
c) A Summary OK state.

Inverting Alarms

Inverting sets 'Warning' and 'Failure' alarm states temporarily to an 'OK' state. The state of the **User Folder** is updated.

Filtering Alarms

Log Field headers may be included or excluded in the Virtual Alarm calculation that the Orbit MapView service does. This is done with **Filters** on items within a **User Folder**.

Use **Filters** to:

- just use a list of Log Headers (include); or
- exclude a list of Log Headers.

To filter items in a User Folder:

- Right-click on a User Folder item and select 'Filter':

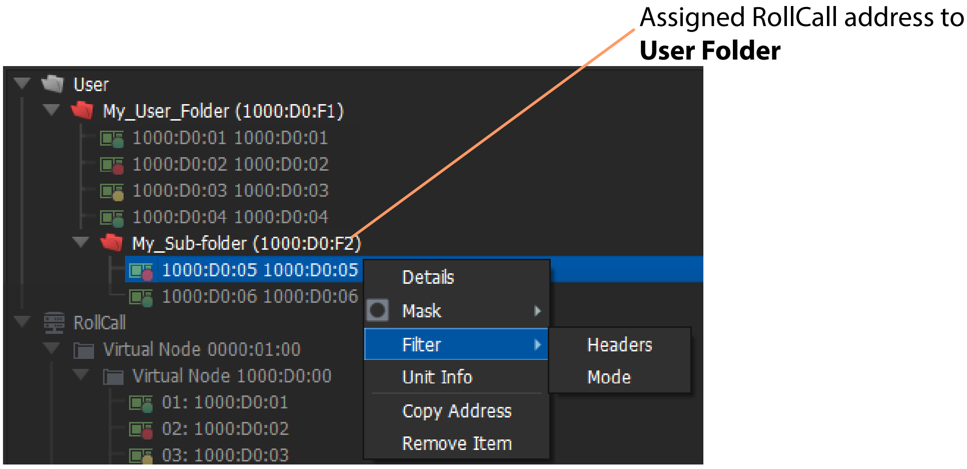


Fig. 9-35: Filter Items within a User Folder - Right-click Menu Item

Table 9-2: Masking Right-click menu Items

Property	Description
Headers	Select this to build a list of LOG_HEADERS for the user folder item. To add to the list: • Select the header item and click '**>>**'. The header appears in the right-hand side list. To remove an item from the list: • Select it in the list and click '**Delete Selected**'.

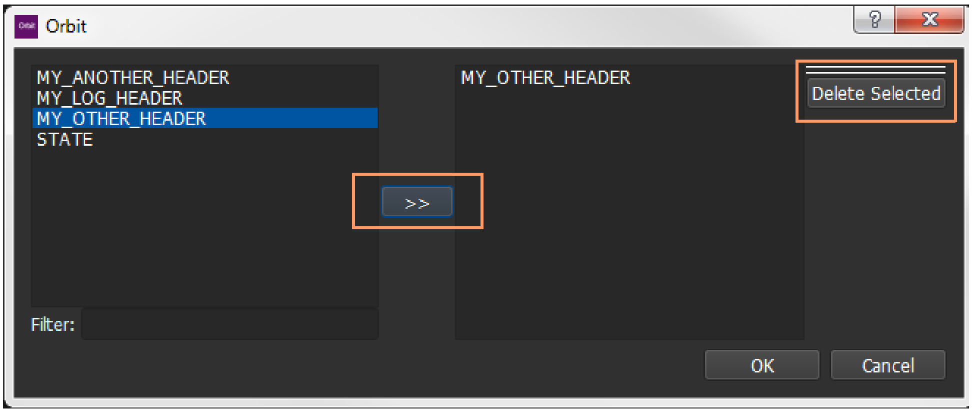
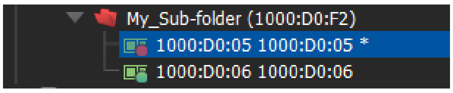
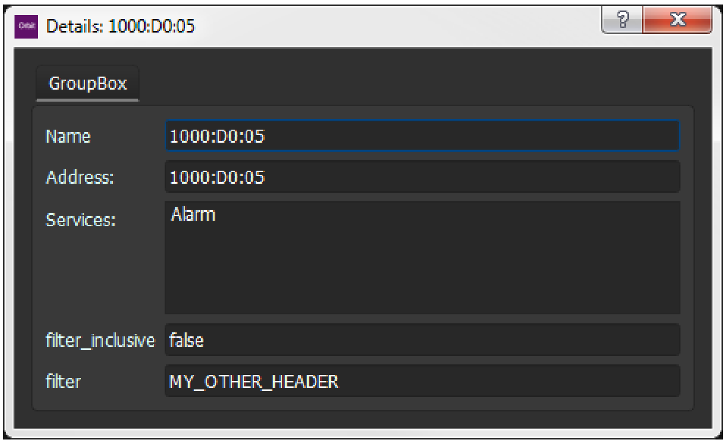
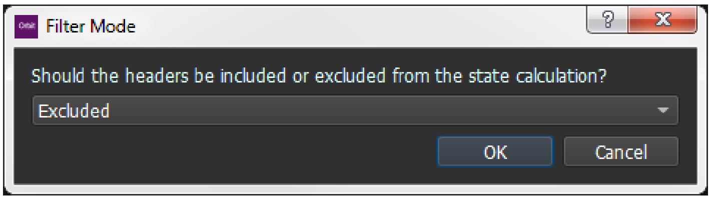


Table 9-2: Masking Right-click menu Items (continued)

Property	Description
	Filtering that is applied to a **User Folder** item is indicated in the **Network View.** Items are shown with an asterisk (***)** appended to their name when a filter is applied.  My_Sub-folder (1000:D0:F2) 1000:D0:05 1000:D0:05 * 1000:D0:06 1000:D0:06 Right-click on a User Folder item and select ‘Unit Info’ to show a ‘Details’ dialog showing some information:  Name 1000:D0:05 Address: 1000:D0:05 Services: Alarm filter_inclusive false filter MY_OTHER_HEADER
Mode	Select this to exclude or include the listed LOG_HEADERS. Options: • **Excluded** - Log headers in the list (for the user folder item) are excluded from the Virtual Alarm calculation. • **Included** - Log headers in the list are the only ones (from the user folder item) used in Virtual Alarm calculation.  Should the headers be included or excluded from the state calculation? Excluded The filtering mode applied to a user folder item is indicated: • Right-click on the user folder item and select ‘Unit Info’: • With **filter_inclusive** = ‘false’, then mode is ‘**Excluded**’. • With **filter_inclusive** = ‘true’, then mode is ‘**Included**’.

Filtering at Folder Level

Filtering can be applied at device-level or at user (sub-)folder-level.

When filtering is used at folder level, then only alarms of devices within that folder will be available for selection for filtering.

10 Custom Logic

Summary

Custom Logic

Introduction to Custom Logic

Custom Logic allows the functionality, or logic, that is defined with Behaviours/Bindings to be captured in an Orbit 'logic file' for re-use. **Custom Logic** may be thought of as a function or a method. One or more Behaviours and/or Bindings may be used in 'logic files'. And there can optionally be:

- zero, one or more input arguments; and
- zero or one output values.

User-defined functionality can be captured once and re-used many times in screens or in Orbit global files. Figure 10-1 shows example functionality that can be encapsulated into a 'logic file'.

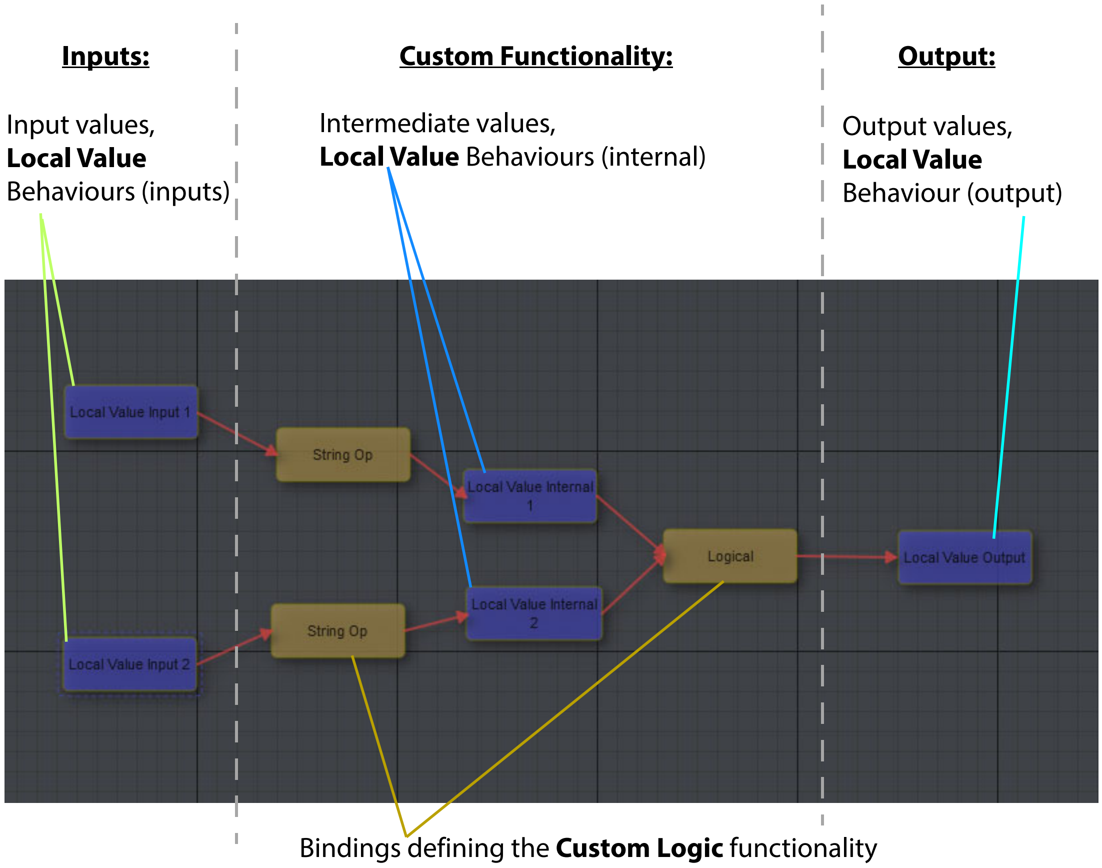

Inputs:
Input values, **Local Value** Behaviours (inputs)
Custom Functionality:
Intermediate values, **Local Value** Behaviours (internal)
Output:
Output values, **Local Value** Behaviour (output)
Local Value Input 1
String Op
Local Value Internal 1
Logical
Local Value Output
Local Value Internal 2
String Op
Local Value Input 2
Bindings defining the **Custom Logic** functionality

Fig. 10-1: Functionality for Encapsulated into Custom Logic

Advantages of using **Custom Logic**:

- Wraps up a complex set of Behaviours/Bindings into a single file which can be re-used.
- Keeps all maintenance in one place.
- Avoids duplication across and inside screens and other files.
- Can be configured using Orbit variables.
  (Variables may be defined inside the 'logic file' and overridden when the **Custom Logic** is used.)

For example, a text string may be extracted from a Log Field and then made lower case; this requires the use of two **String** Bindings - which would be arduous to repeat numerous times. With **Custom Logic**, this string processing is defined once and then configured for use with multiple Log Field value inputs.

Custom Logic File

Create Logic File

To create a new ‘logic file’, in the **Project View** of the Orbit MapView project:

1 Right-click on the project name and select ‘New Folder’. Name the folder ‘logic’.

2 Right-click on the project name and select ‘New File’.

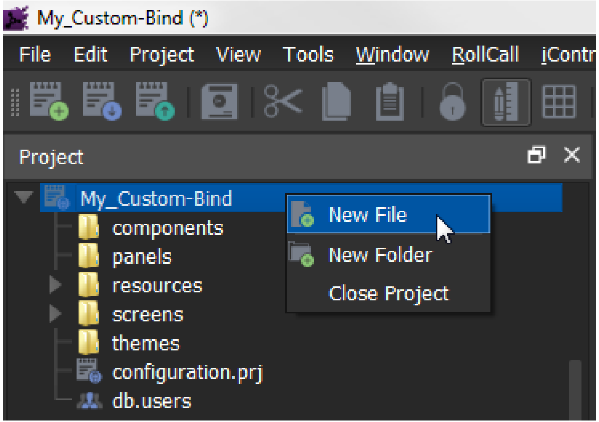
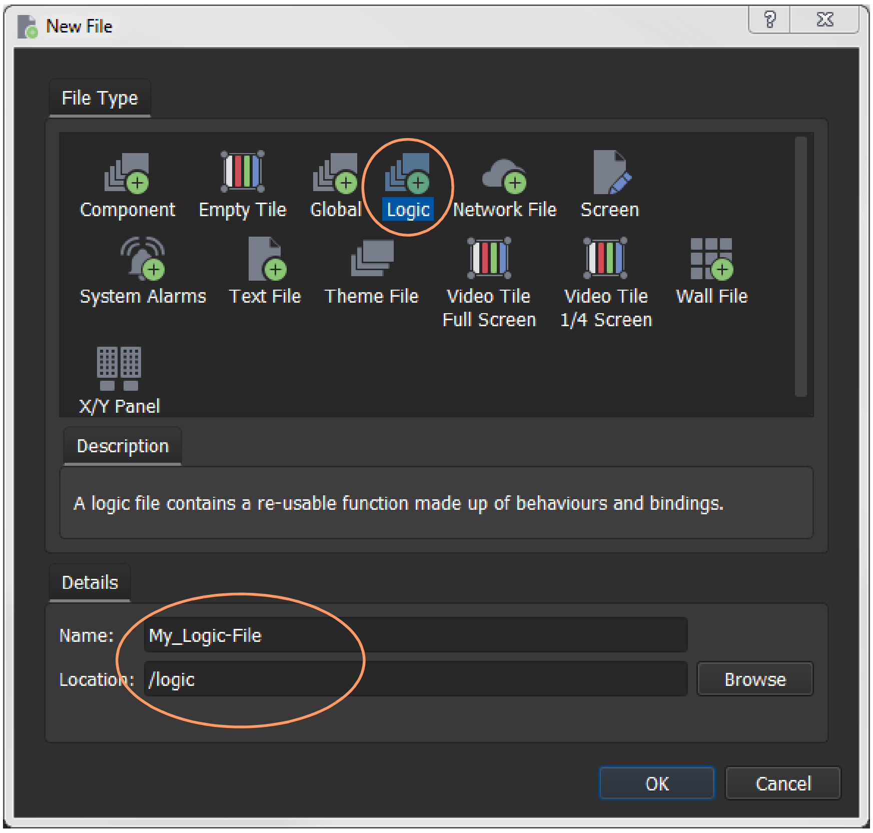


Fig. 10-2: Right-click and Select ‘New File’

3 Browse to the new ‘logic’ folder and enter a name for the new logic file (‘My_Logic-File’ in this example). See Figure 10-3.



A logic file contains a re-usable function made up of behaviours and bindings.

Name: My_Logic-File

Location: /logic

Fig. 10-3: Creating a New Custom Logic File

4 Click **OK**.

The new file is opened in a Behaviour/Bindings graphic editor. See Figure 10-4.

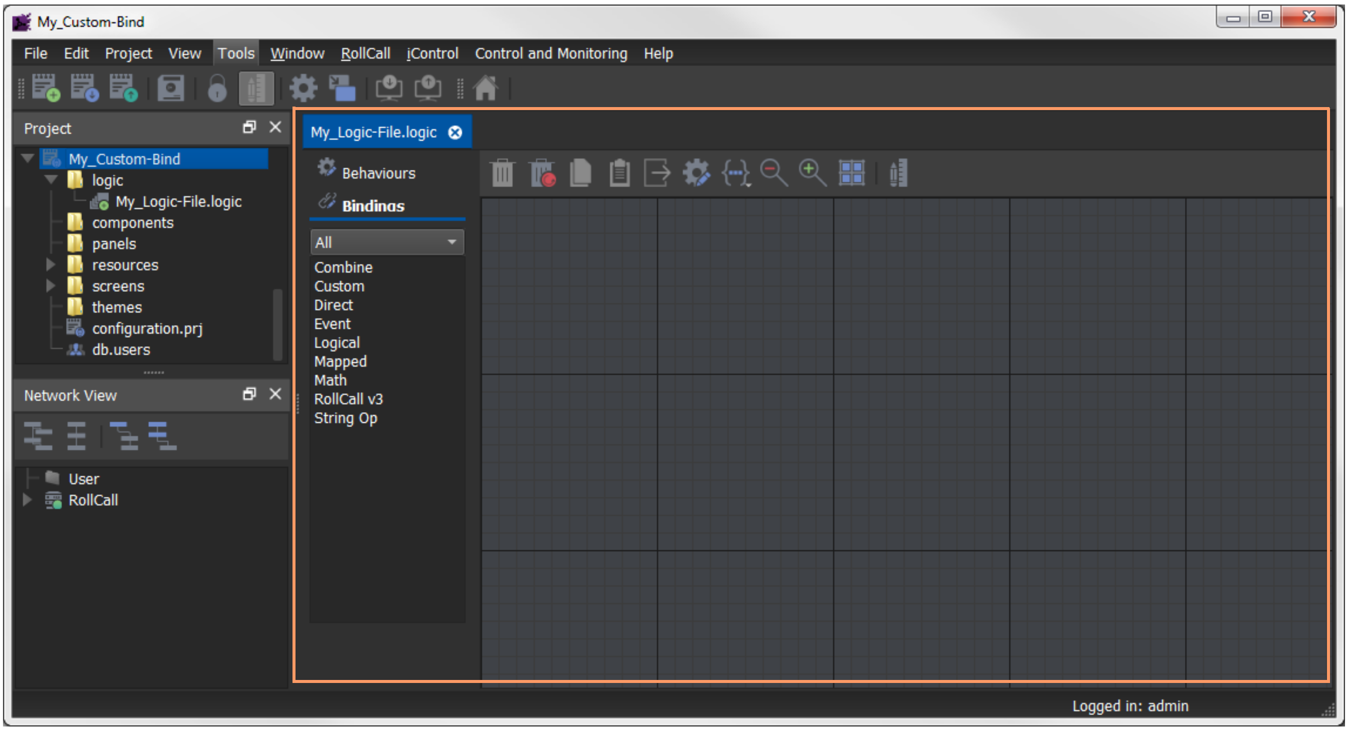


Fig. 10-4: Behaviour/Binding Graphic Editor

An (empty) **Custom Logic** file has been created.

Configure Custom Logic File Input(s) and an Output

Custom Logic files may have zero or more input values, and zero or one output value. An input or output is defined using a **Local Value** Behaviour. A **Local Value** Behaviour can also act as an internal node of the **Custom Logic**, holding an interim value.

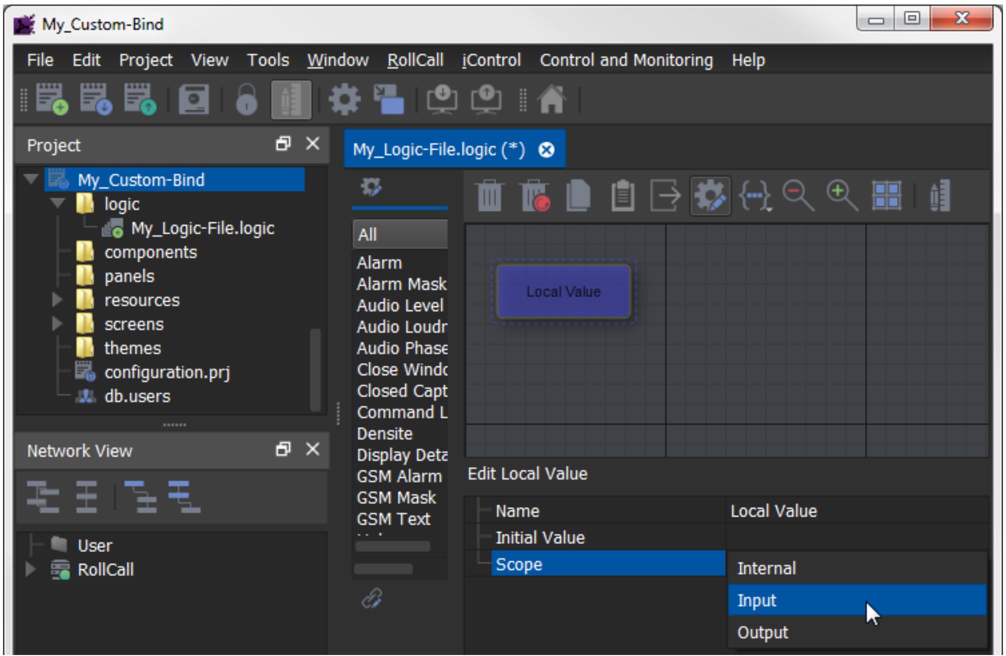


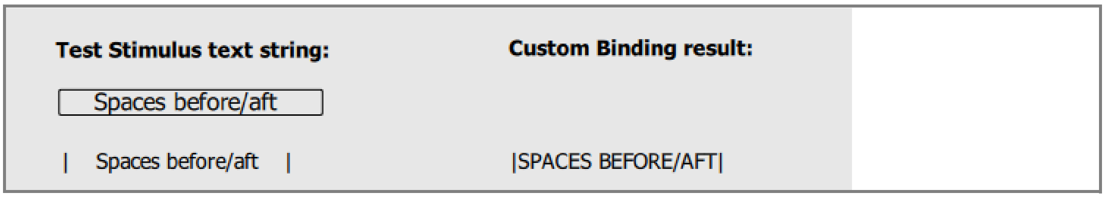
Fig. 10-5: Local Value Behaviour Edit Properties

After a **Local Value** Behaviour is added, its properties include a 'Scope' property. The **Local Value** Behaviour holds a value whose scope in the **Custom Logic** is set by the 'Scope' property:

- **Internal** - holds a value which is internal.
- **Input** - holds a value which is an input.
- **Output** - holds a value which the output.

Note: Do not delete and re-create inputs or outputs.
Orbit identifies each input and output using an internal ID. Therefore, the user is free to change the 'Name' property.
However, if these are deleted and re-created, then the ID would change.

Example - Custom Text Processing (Trim Text, Convert to UPPERCASE)



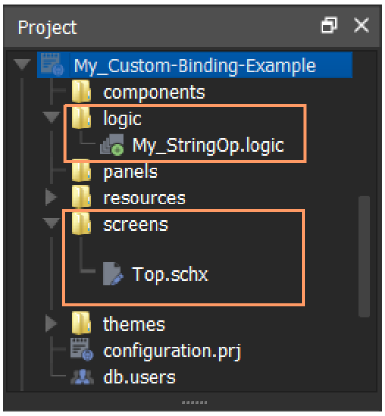
Test Stimulus text string: **Custom Binding result:**

Spaces before/aft

| Spaces before/aft | |SPACES BEFORE/AFT|

Preliminaries

1 Create a Orbit MapView project.
2 Create a new (empty) screen (.schx file).
3 Create a new (empty) **Custom Logic** file (.logic).



Project

My_Custom-Binding-Example
components
logic
My_StringOp.logic
panels
resources
screens
Top.schx
themes
configuration.prj
db.users

Fig. 10-6: Example Project View with New Screen and Custom Logic Files

Build the Custom Logic File

1 Double-click on the logic file item in the **Project View**.
The **Custom Logic** file is opened in the editor.

2 Add a **Local Value** Behaviour.
and set 'Name' = 'My Input' and set 'Scope' = 'Input'.

3 Add a **Local Value** Behaviour.
and set 'Name' = 'My Output' and set 'Scope' = 'Output'.

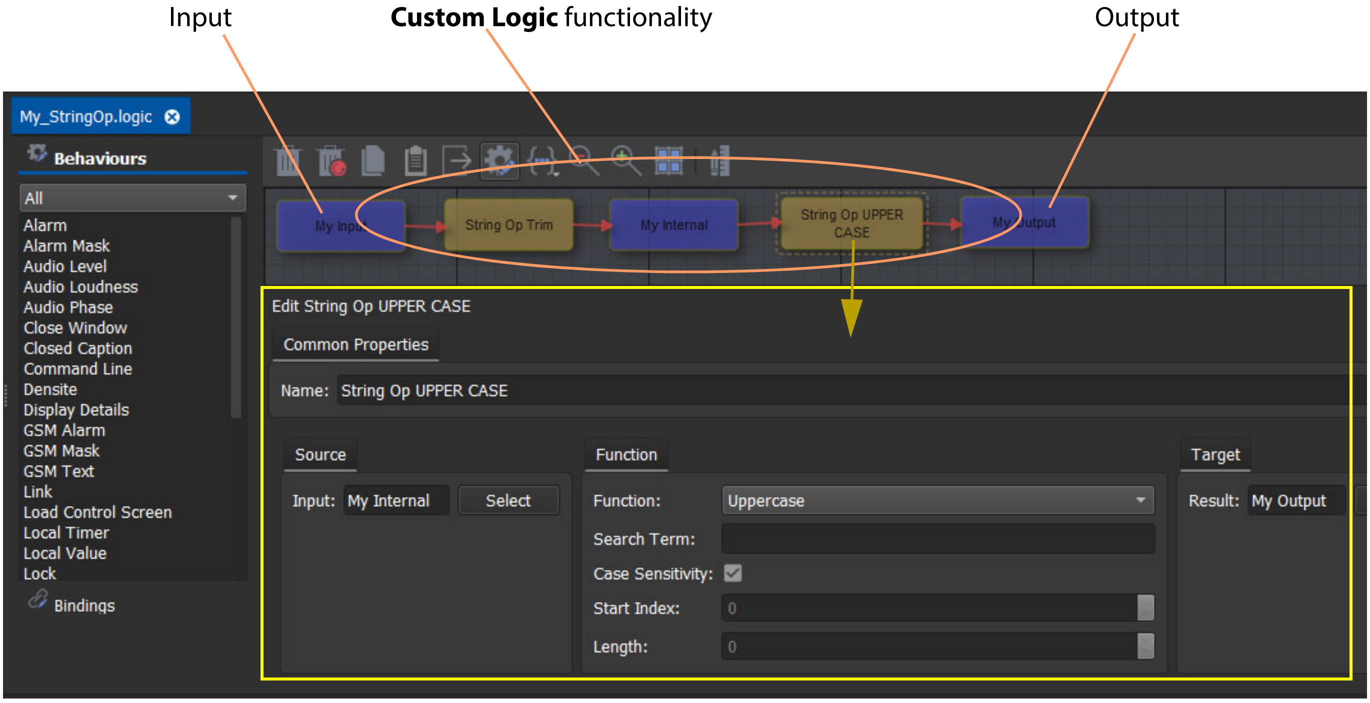
4 Add a **Local Value** Behaviour.
and set 'Name' = 'My Internal' and set 'Scope' = 'Internal'.

5 Add a **String Op** Binding
and set:
- 'Source' = 'My Input' **Local Value** Behaviour;
- 'Function' = 'Trim';
- 'Target' = 'My Internal' **Local Value** Behaviour;

6 Add a **String Op** Binding
and set:
- 'Source' = 'My Internal' **Local Value** Behaviour;
- 'Function' = 'Uppercase';
- 'Target' = 'My Output' **Local Value** Behaviour;

7 Click **Save**.

A functional **Custom Logic** file has been created. See Figure 10-7.



Edit String Op UPPER CASE

Common Properties

Name: String Op UPPER CASE

Input: My Internal

Function: Uppercase

Search Term:

Case Sensitivity: ☑

Start Index: 0

Length: 0

Result: My Output

Fig. 10-7: Created Custom Logic File

The **Custom Logic** file can be used like a Binding in a screen schematic or in an Orbit global file, including being used within another logic file.

Using Custom Logic on a Screen

The **Custom Logic** file created above is used on an Orbit MapView screen. This example creates a test screen to demonstrate the use of a **Custom Logic** file:

1 Create a screen schematic.

2 Add a **Text Edit** widget, select it and click the **Edit Behaviours** button to show the 'Behaviour/Bindings' graphic editor.

3 Add a **Local Value** Behaviour.

Orbit connects a **Local Value** Behaviour to the **Text Edit** widget's value via a **Direct** Binding. The Behaviour will hold the value of the **Text Edit** widget.

4 Select the screen background.

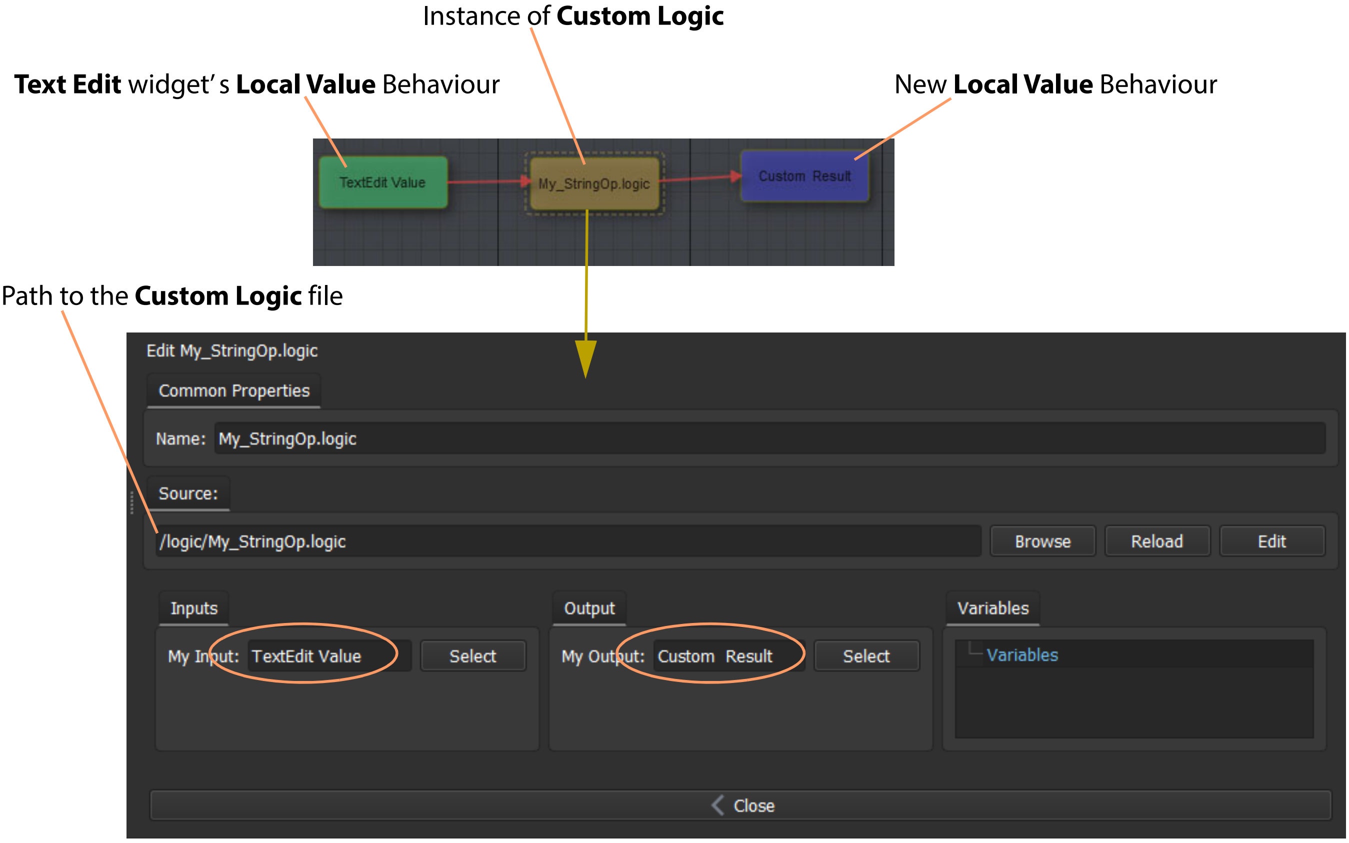
5 Drag and drop the **Custom Logic** file onto the 'Behaviour/Bindings' graphic editor.

6 Double-click on the **Custom Logic** in the graphic editor to edit its properties:

- Set 'Input' to be the **Text Edit** widget's **Local Value** Behaviour.
- Set 'Output' to be a new **Local Value** Behaviour.

See Figure 10-8.

7 Click **Close**.



Instance of **Custom Logic**

Text Edit widget' s **Local Value** Behaviour

New **Local Value** Behaviour

Path to the **Custom Logic** file

Edit My_StringOp.logic

Common Properties

Name: My_StringOp.logic

Source:

/logic/My_StringOp.logic

Inputs

My Input: TextEdit Value

Output

My Output: Custom Result

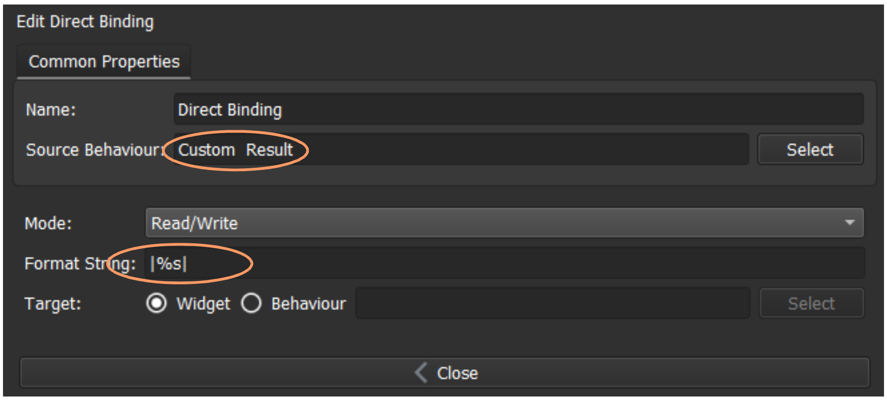
Fig. 10-8: Custom Logic Settings

8 Add a **Label** widget to the screen, select it and add a **Direct** Binding.

9 Double-click on the **Direct** Binding to edit its properties.

- Set 'Source Behaviour' = **Local Value** Behaviour holding the **Custom** Binding result.
- Set 'Format String' = |%s|
  (This will print a string surrounded by vertical bar characters, | |.)

See Figure 10-9.



Edit Direct Binding

Common Properties

Name: Direct Binding

Source Behaviour: Custom Result

Mode: Read/Write

Format String: |%s|

Target: ● Widget ○ Behaviour

Fig. 10-9: Direct Binding Edit Properties

10 Click **Close**.

This has created a **Label** to show the resulting text string from the **Custom** Binding. The text string will be shown within the vertical bar characters, | |, specified in the format string.

11 Add a further **Label** widget, in a similar way, to show the test stimulus text (i.e. the value of the **Text Edit** widget).

12 Click **Close**.

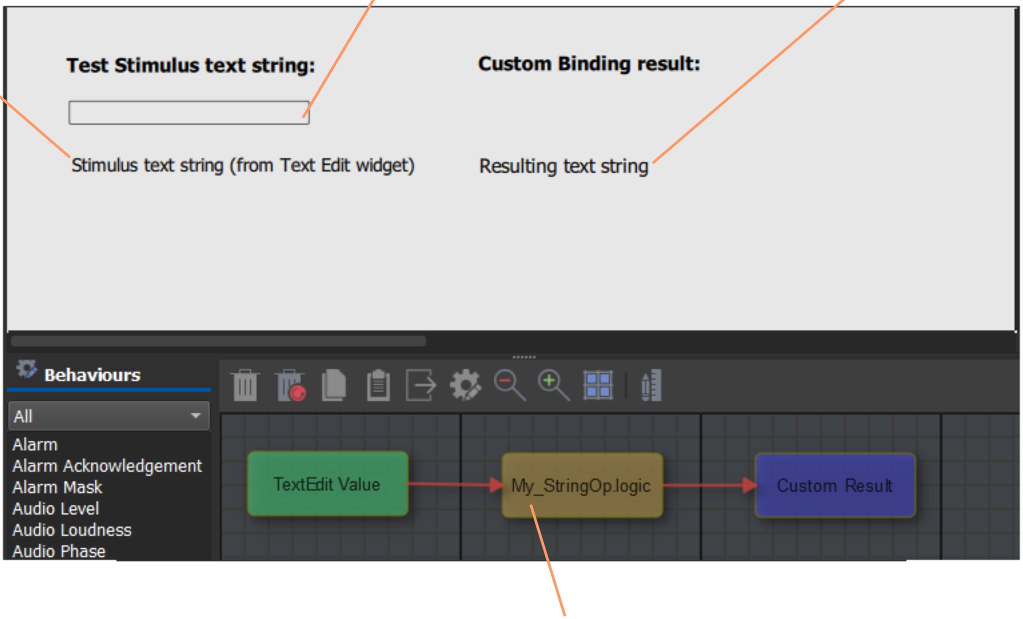
13 Click **Save**. Click **Save Project**.

The resulting screen (see Figure 10-10) can be used to exercise the **Custom Logic**.

Enter text as a test stimulus into the **Text Edit** widget

Label widget showing value of the **Text Edit** widget

Label widget showing the **Custom Logic** result



Custom Logic under test

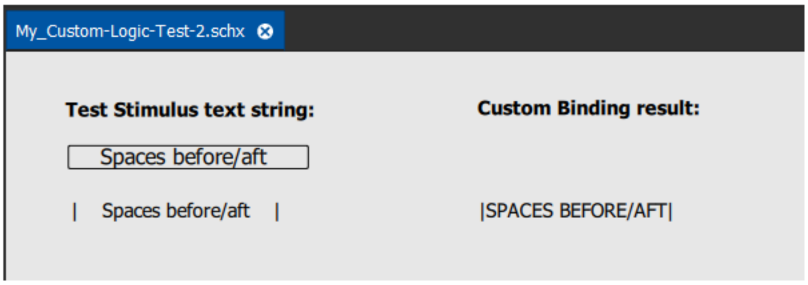
Fig. 10-10: Test Screen for Custom Logic

Exercising the Custom Logic

1 Set the screen into **Run Mode**.

2 Enter a string value into the **Text Edit** widget and view the resulting text.

The resulting text will be converted to upper case and have any initial or final space characters trimmed.



Test Stimulus text string:

Spaces before/aft

| Spaces before/aft |

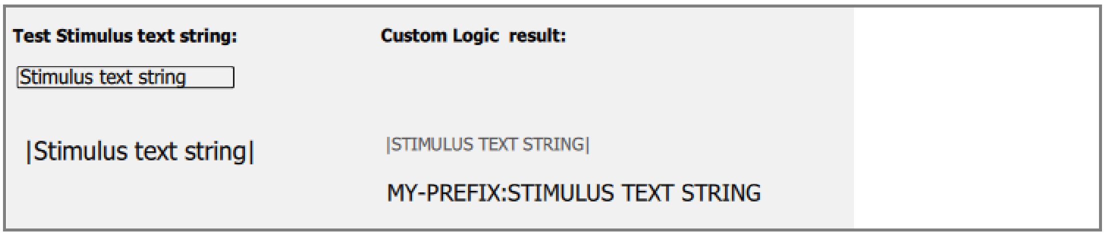
Custom Binding result:

|SPACES BEFORE/AFT|

Fig. 10-11: Custom Logic Exercising

This example shows **Custom Logic** being passed a value via a **Local Value** Behaviour.

Example - Pass Variables to Custom Logic (Prefix a Text String)



Test Stimulus text string:

Stimulus text string

|Stimulus text string|

Custom Logic result:

|STIMULUS TEXT STRING|

MY-PREFIX:STIMULUS TEXT STRING

The above example (Example - Custom Text Processing (Trim Text, Convert to UPPERCASE), on page 320) passes an argument to **Custom Logic** by using a **Local Value** Behaviour, and shows how value can be passed into or out of **Custom Logic**.

Custom Logic may also be configured by using variables which can be set when a **Custom Logic** file is used. This is described in the example below:

Prepare a Custom Logic File with Variables

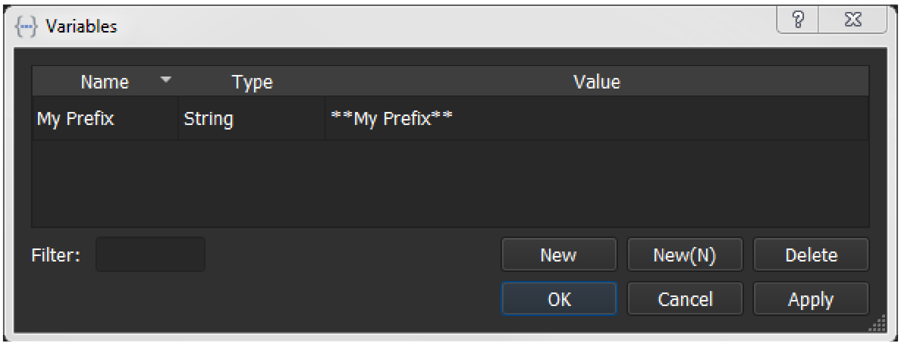
This example prefixes a text string with some text. A 'prefixing' **Custom Logic** file is generated where the prefix text can be set. This adds an additional step to the above example, Example - Custom Text Processing (Trim Text, Convert to UPPERCASE), on page 320.

1 Create a **Custom Logic** file, 'My_Prefixing'.

2 Right-click in the background of the **Custom Logic** graphic editor
and select '{...} Variables'.

3 In the **Variables** dialog,
create a new variable called 'My Prefix'
and set its value to '**My Prefix**'. See Figure 10-12.

4 Click **OK**.



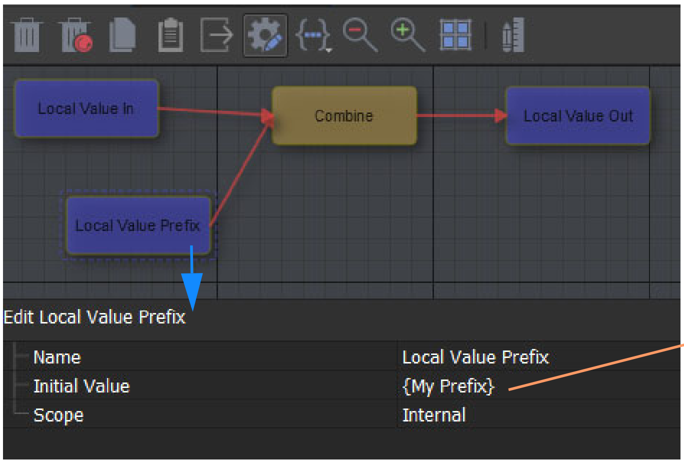
Name	Type	Value
My Prefix	String	**My Prefix**

Fig. 10-12: Variables Dialog

5 Add a **Local Value** Behaviour, 'Local Value In', with 'Scope' = 'Input'.

6 Add a **Local Value** Behaviour, 'Local Value Out', with 'Scope' = 'Output'.

7 Add a **Local Value** Behaviour, 'Local Value Prefix', with 'Scope' = 'Internal.
And set its 'Initial Value' = '{My Prefix}'.
(Using the curly braces, { }, specifies the *value of* the variable 'My Prefix'),
See Figure 10-13.



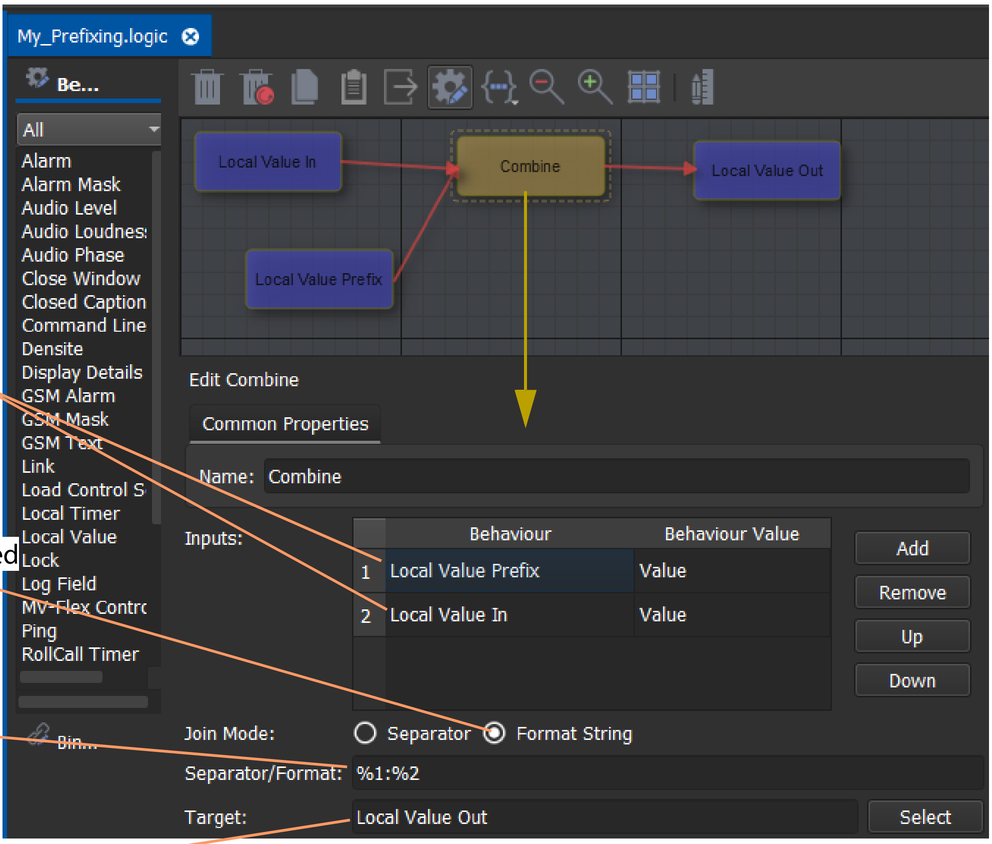
Edit Local Value Prefix

Name	Local Value Prefix
Initial Value	{My Prefix}
Scope	Internal

Initial Value = {My Prefix}

Fig. 10-13: Prefixing Custom Logic

8 Add a **Combine** Binding and configure it to concatenate two text strings with a ‘colon’ character. See Figure 10-14.



Edit Combine

Common Properties

Name: Combine

Inputs:

	Behaviour	Behaviour Value
1	Local Value Prefix	Value
2	Local Value In	Value

Join Mode: ○ Separator ● Format String

Separator/Format: %1:%2

Target: Local Value Out

Input text strings

‘Format String’ selected

Output string format specified:

%1:%2

Output the text string to a new **Local Value** Behaviour

Fig. 10-14: Custom Logic

9 Click **Save**.

The logic file is complete and the prefixing **Custom Logic** is now ready for use.

Using a Custom Logic File with Variables

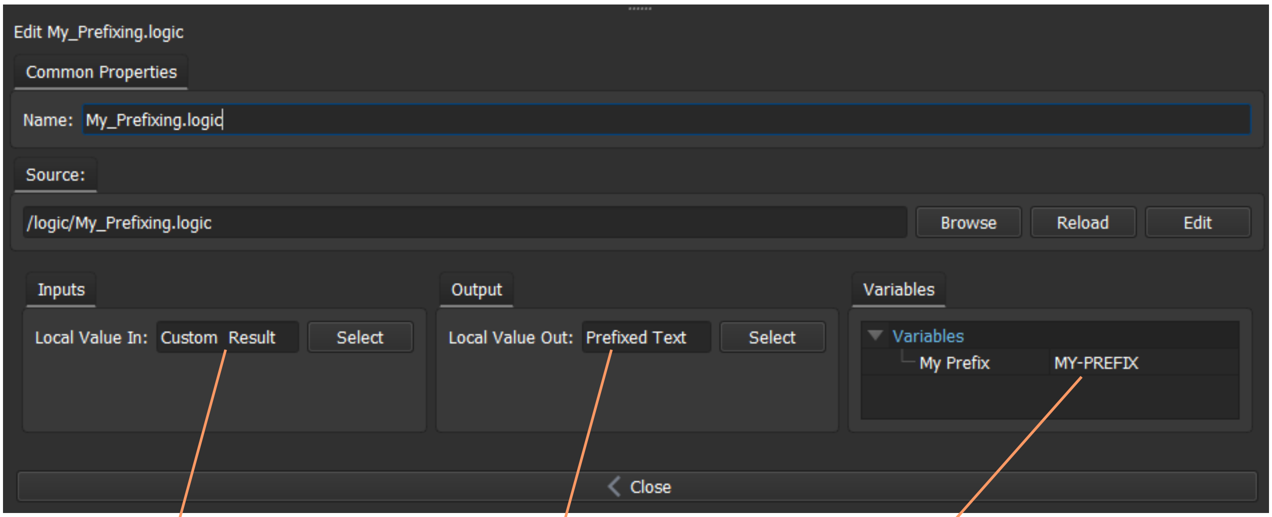
This example extends the example of Example - Custom Text Processing (Trim Text, Convert to UPPERCASE), on page 320 above:

1 Open the screen of Example - Custom Text Processing (Trim Text, Convert to UPPERCASE), on page 320 above,
or make a copy of the screen for use here.
(To make a copy:
   - select the file in the **Project View**; right-click on it;
   - select 'Copy'; and
   - enter a new file name.)

2 Select the background of the screen
and open the 'Behaviour/Bindings' graphic editor.

3 Drag and drop the new 'prefixing **Custom Logic** file' onto the graphic editor.
And double-click on it to edit its properties.
Set:
   - 'Input' to be the **Local Value** Behaviour holding the *result* of the earlier example.
   - Set the 'Output' to be a new **Local Value** Behaviour, call it 'Prefixed Text'.
   - Set the variable listed in the 'Variables' section to be 'MY-PREFIX' .

The new **Custom Logic** instance has been configured, see Figure 10-15.



Edit My_Prefixing.logic

Common Properties

Name: My_Prefixing.logic

Source:

/logic/My_Prefixing.logic

Inputs

Local Value In: Custom Result

Output

Local Value Out: Prefixed Text

Variables

Variables
My Prefix MY-PREFIX

Input = result of previous **Custom Logic**

Variable 'My prefix' = 'MY-PREFIX'

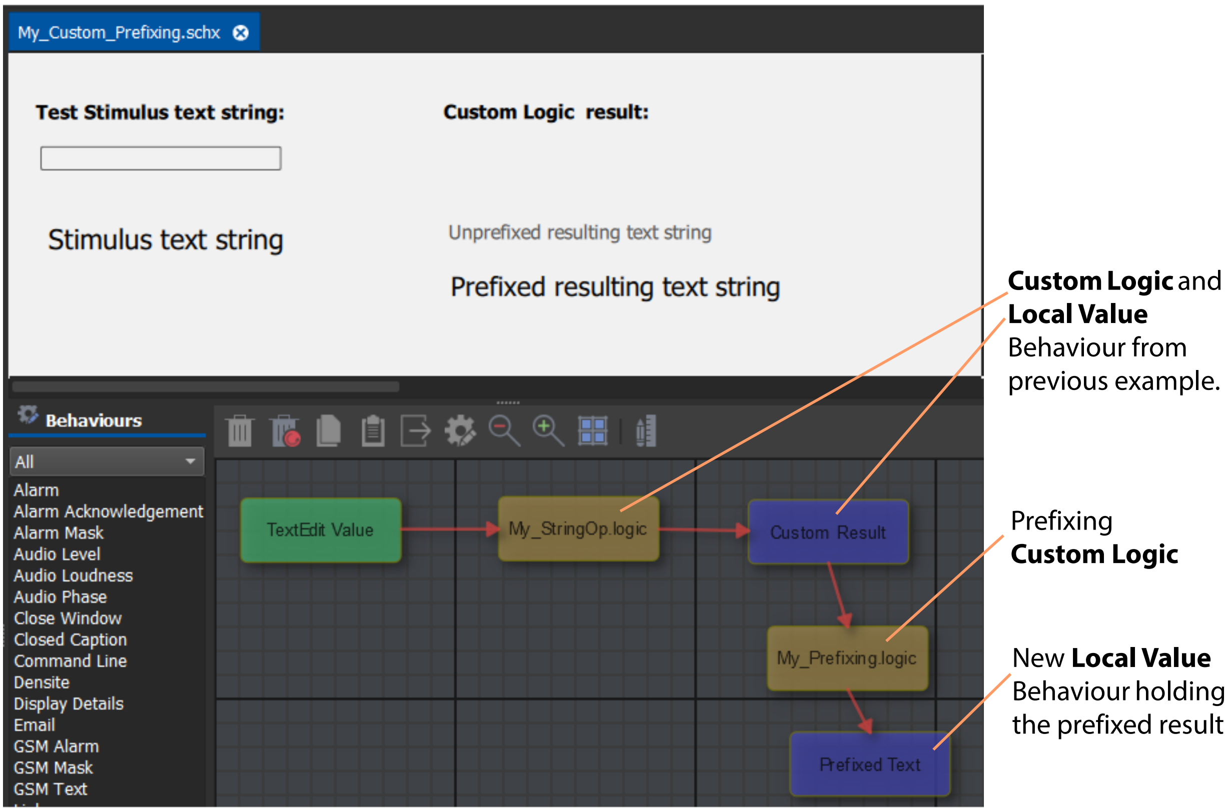
Output = new **Local Value** Behaviour

Fig. 10-15: Configuring New Custom Logic File with Variables

4 Click **Close**.

5 Click **Save**.

6 The 'Behaviour/Bindings' graphic editor now shows the previous example connected to this prefixing **Custom Logic,** and the result is connected to a **Local Value**. See Figure 10-16.



My_Custom_Prefixing.schx

Test Stimulus text string:

Custom Logic result:

Stimulus text string

Unprefixed resulting text string

Prefixed resulting text string

Custom Logic and **Local Value** Behaviour from previous example.

Prefixing **Custom Logic**

New **Local Value** Behaviour holding the prefixed result

Fig. 10-16: Prefixing Custom Logic Instantiated

7 Add a **Label** widget to the screen and connect it to the 'prefixed text' **Local Value** Behaviour. This will show the final, prefixed result.

8 Click **Save**. Click **Save Project**.

The screen is ready for running and testing.

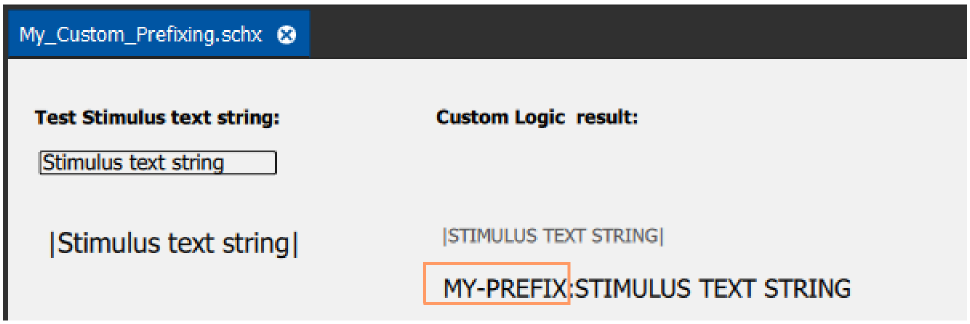
Exercise the Custom Logic File with Variables

1 Enter **Run Mode**.

2 Exercise the prefixing **Custom Logic**:

A string value entered into the original **Text Edit** box is converted to upper case and trimmed, as before, and is then prefixed with some text.

See Figure 10-17.



My_Custom_Prefixing.schx

Test Stimulus text string:

Stimulus text string

Custom Logic result:

|Stimulus text string|

|STIMULUS TEXT STRING|

MY-PREFIX:STIMULUS TEXT STRING

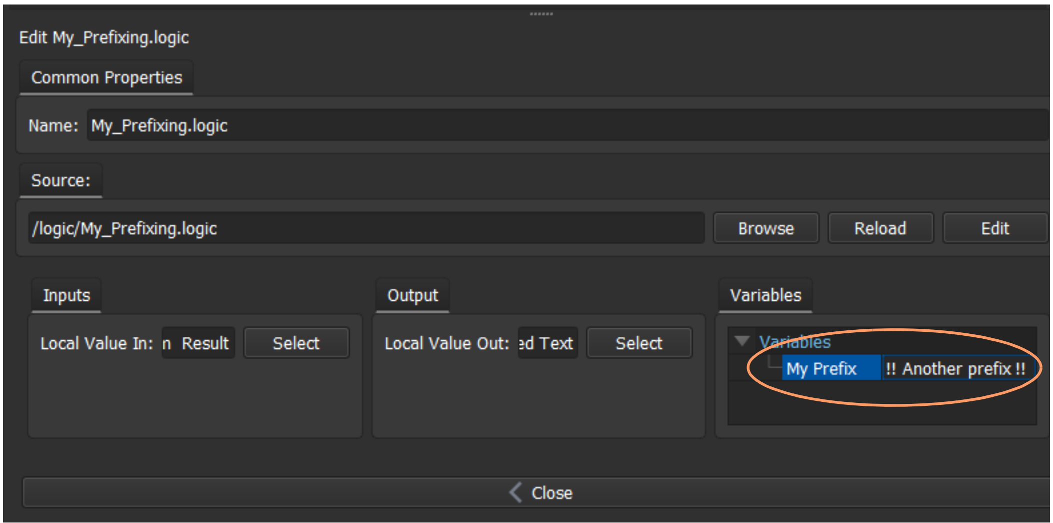
Fig. 10-17: Exercising the Prefixing

The preset text is currently set to 'MY-PREFIX', but this can be changed for the specific instance if the **Custom Logic**:

3 Enter **Design Mode**.

4 Select the background of the screen and show the 'Behavours and Bindings' graphical editor.

5 Double-click on the prefixing Custom Logic to change it settings. See Figure 10-18.



Edit My_Prefixing.logic

Common Properties

Name: My_Prefixing.logic

Source:

/logic/My_Prefixing.logic

Local Value In: n Result

Local Value Out: ed Text

Variables

My Prefix !! Another prefix !!

Fig. 10-18: Change Prefix Variable for an Instance of Custom Logic

6 Click **Close**.

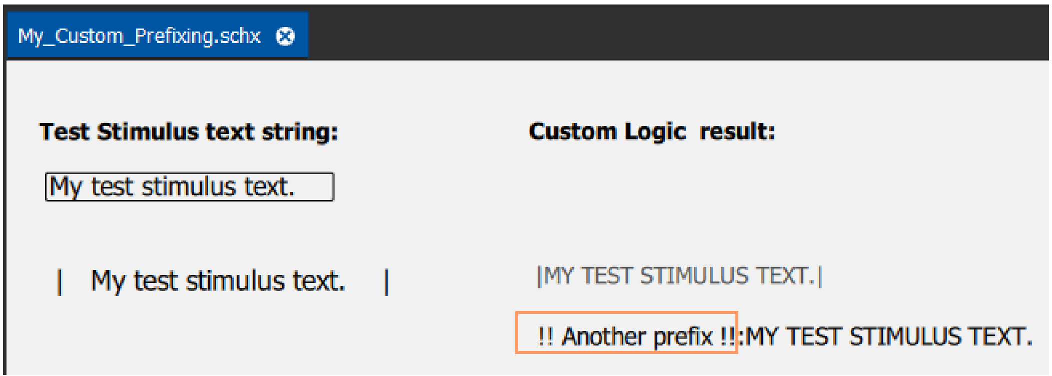
7 Click **Save**.

8 Enter **Run Mode**.

9 Exercise the prefixing **Custom Logic**:

A string value entered into the original **Text Edit** box is converted to upper case and trimmed, as before, and is then prefixed with some text.

See Figure 10-19.



My_Custom_Prefixing.schx

Test Stimulus text string:

My test stimulus text.

| My test stimulus text. |

Custom Logic result:

|MY TEST STIMULUS TEXT.|

!! Another prefix !!:MY TEST STIMULUS TEXT.

Fig. 10-19: Exercising Another Prefix

Other Custom Logic

Custom Logic with No ‘Output’

It is valid to construct **Custom Logic** which has no ‘output’. This may be required, for example, to send a message to an external device. (RollCall+ Command Behaviour could be used.)

Custom Logic with No Input

It is valid to construct Custom Logic which has no inputs. This may be required, for example, where some action is to be driven off of a local timer, or some other triggering mechanism, allowing such **Custom Logic** to operate ‘standalone’.

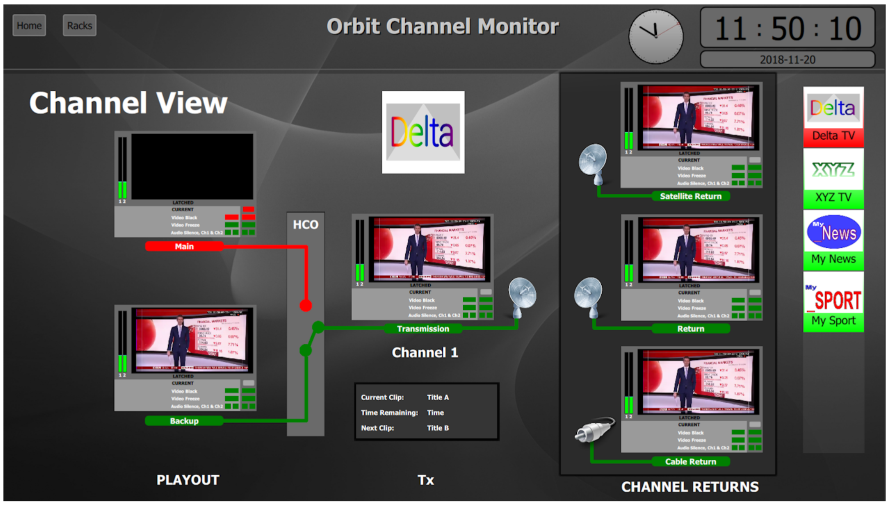
Custom Logic with No Inputs and No Output

It is possible to construct **Custom** Logic which has no inputs and no outputs.

11 Channel Monitoring Example

Summary

Channel Monitoring Example



This chapter presents an example overall Orbit MapView project, which uses some elements of previous examples presented in this manual. An overview of the finished project is described and key features noted.

Introduction to Example MapView Project

An Orbit MapView project will comprise a hierarchy of several screens. Apart from graphical widgets, some screens may use components, Custom Logic, image files. And screen ‘behind the scenes’ functionality is captured in screen schematics and in components using Orbit’s Behaviours and Bindings.

Screens

The presented example project, ‘My_Example’, uses and re-uses the following basic screen designs, shown in Figure 11-1:

a) Top Level screen.

b) Channel Monitor screen.

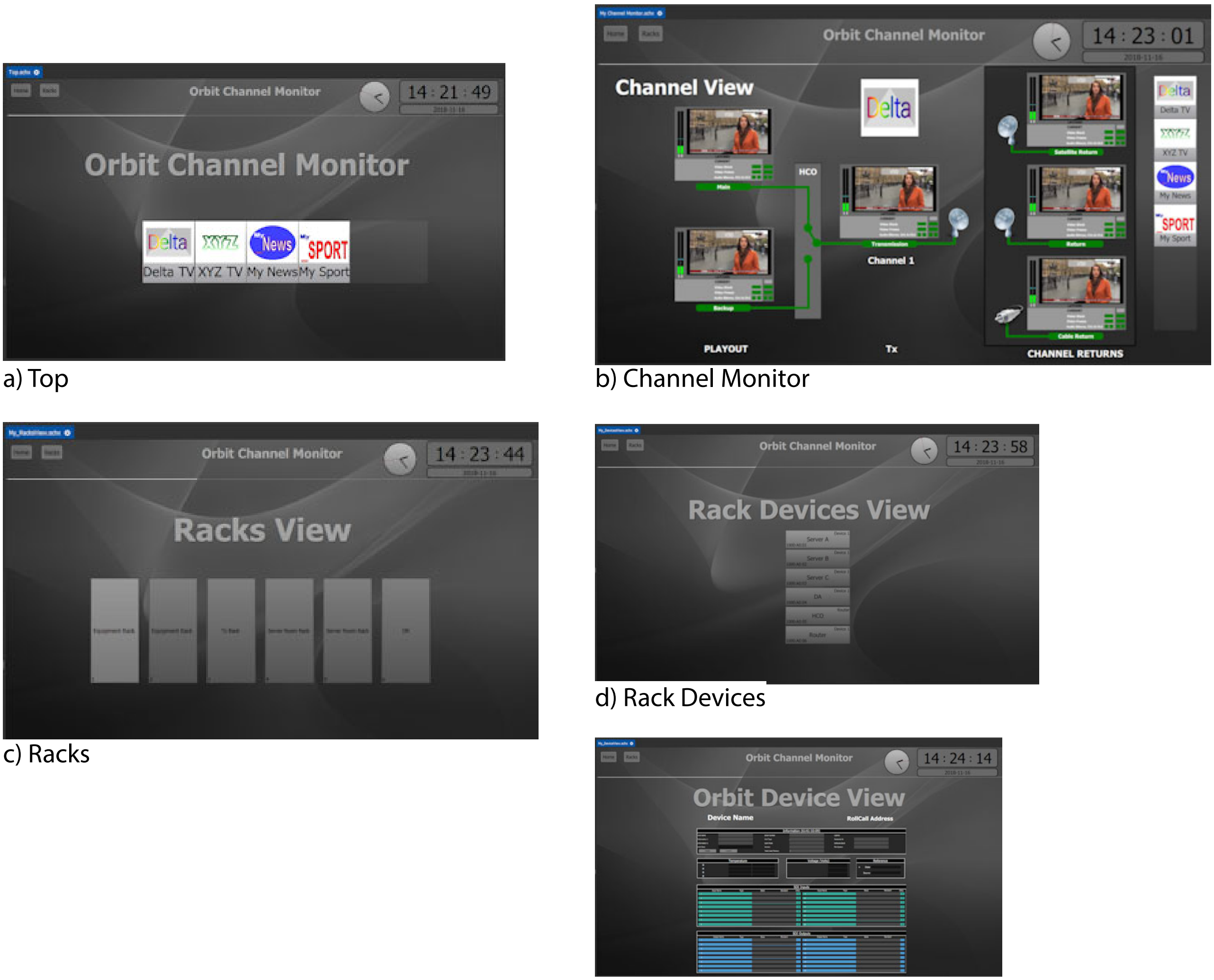
c) Racks screen.

d) Rack Devices screen.

e) Device Information screen.

Screens use components where similar graphical on-screen widgets occur and, for each different component instance, variables are set to differentiate each instance.

Some screens are used more than once and screen instances are differentiated by different screen variables settings.



a) Top

b) Channel Monitor

c) Racks

d) Rack Devices

e) Device Status Information

Fig. 11-1: ‘My_Example’ Project Screens

Screens Hierarchy

The example project’s screen are arranged in a hierarchy: A top level, summary screen shows overall status of several channels and provides time/date information. Access is provided to screens for each channel. Additionally, there is access to engineering-level information for logged-in users with engineering permissions. The screen hierarchy is shown in Figure 11-2.

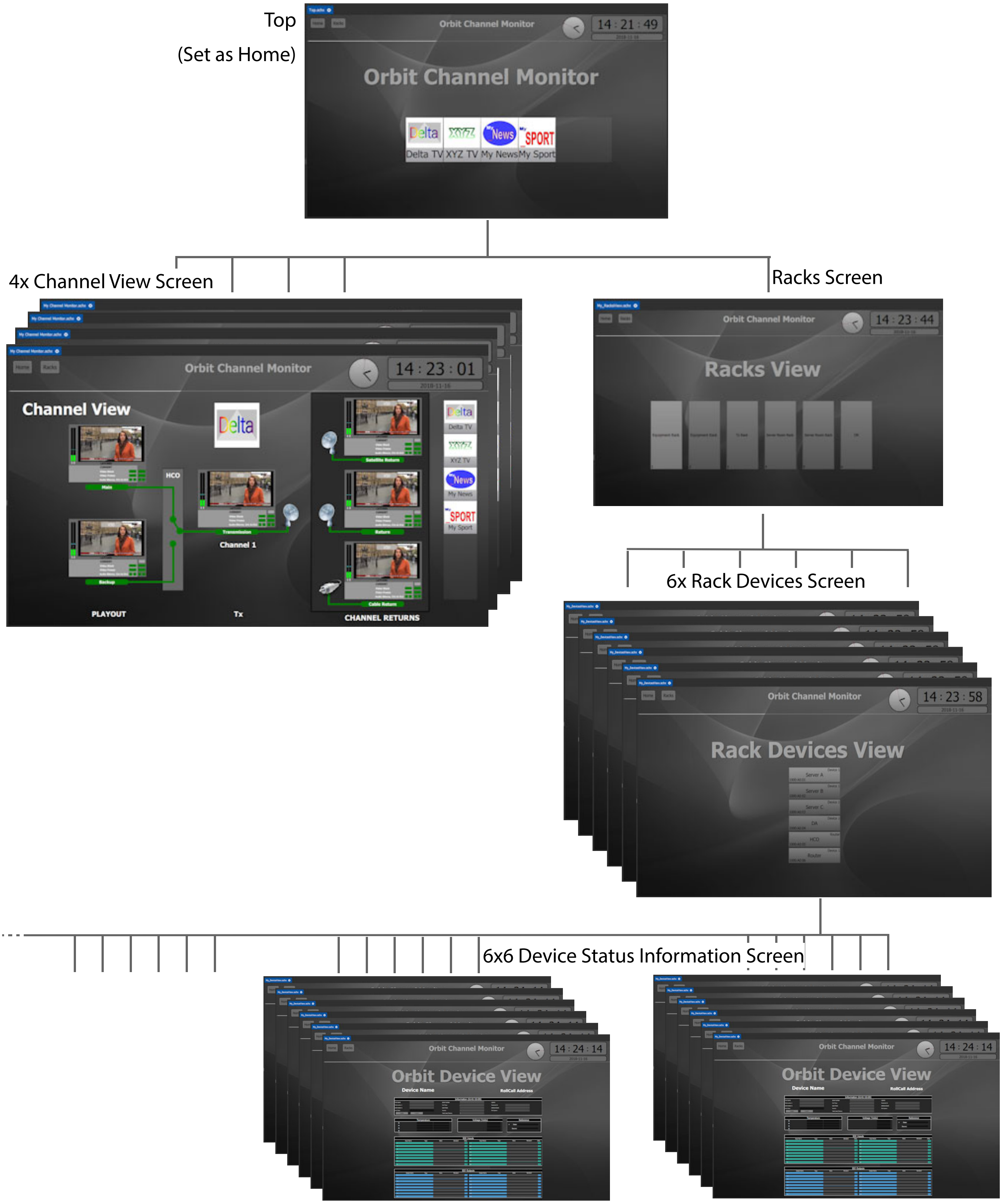


Fig. 11-2: ‘My_Example’ Project Screen Hierarchy

Top Level Screen

The top level screen is shown in Figure 11-3.

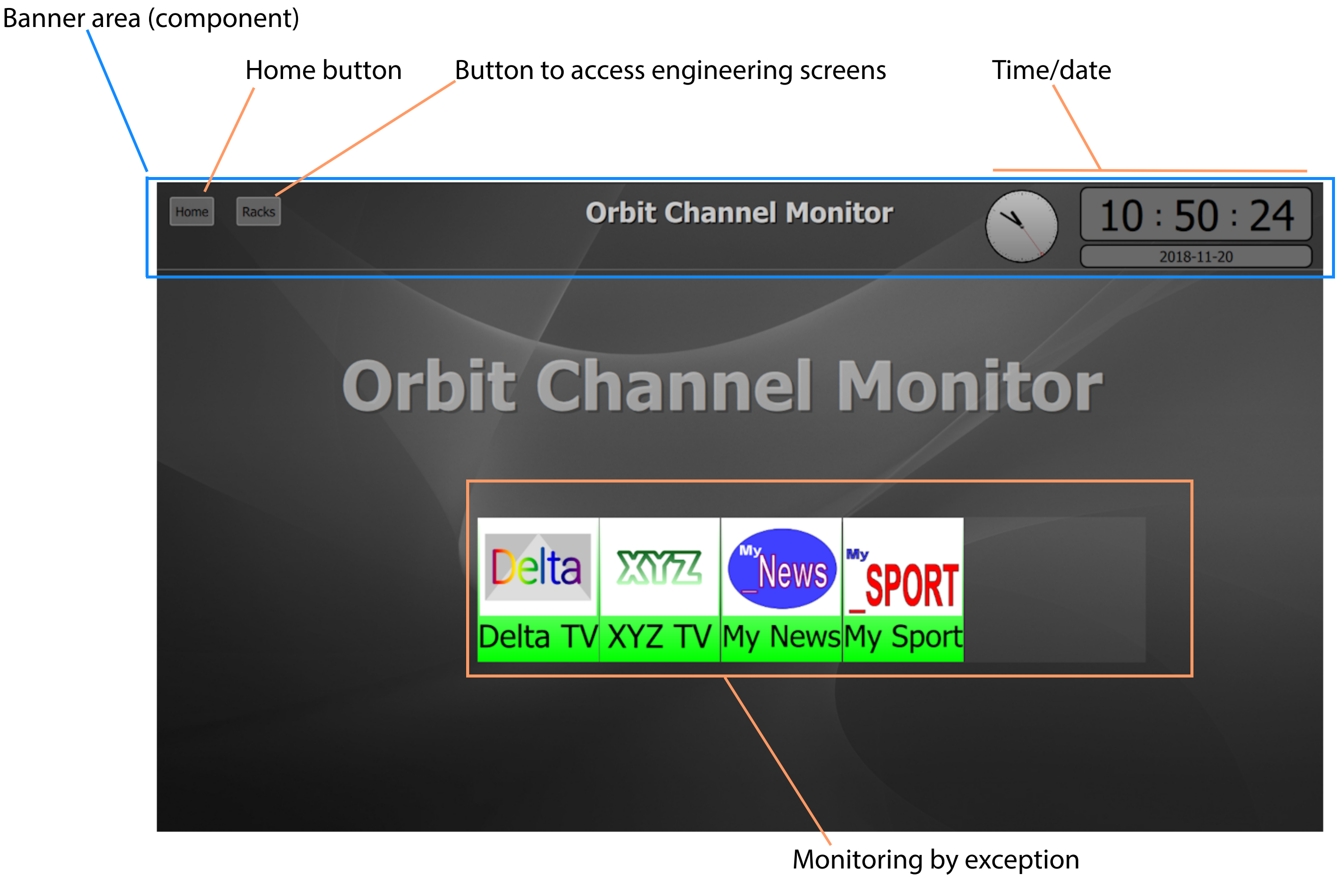


Fig. 11-3: Top Level Screen

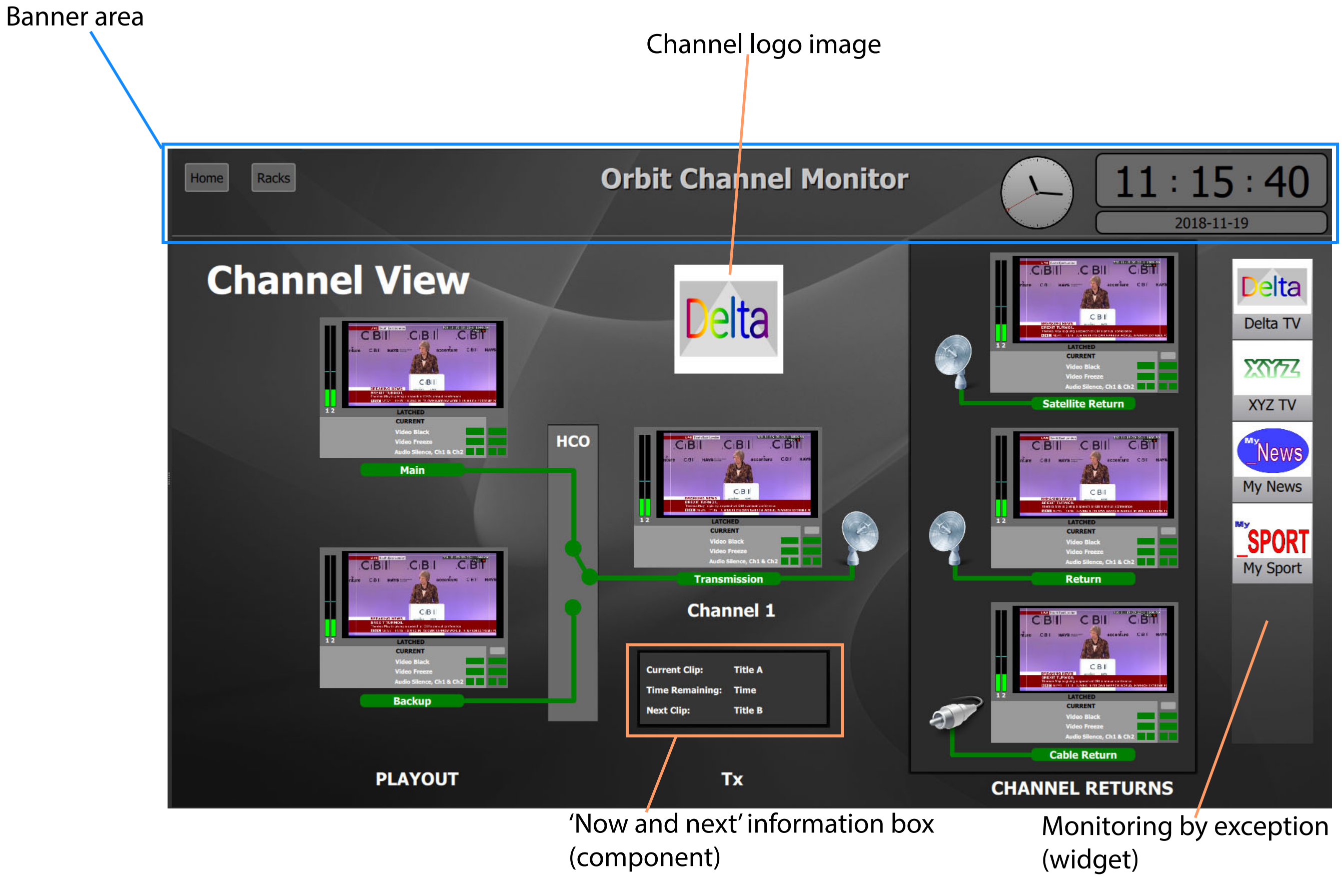
The top level screen comprises:

- A top banner area - which is formed as a component and appears on each other screen. The banner:
  - contains a 'home' button to return to the top level screen from any other;
  - allows access to engineering screens ('Racks' button) for permitted users; and
  - shows the time and date.
- An **Exception Monitor** widget - which shows the overall status of several channels:
  - Each monitored channel is represented by an icon showing a channel logo image.
  - Click on an icon to open the 'channel view' screen for the corresponding channel.

The 'Racks' button links to a 'racks view' screen, from which further device information can be accessed. The button itself uses the 'Permissions' widget property to hide the button when a logged in user does not have engineering permissions.

Channel View Screen

The channel view screen is shown in Figure 11-4.



Current Clip: Title A
Time Remaining: Time
Next Clip: Title B

Fig. 11-4: Channel View Screen

The channel view screen comprises:

- A banner area.
- A channel logo image.
- An animated play-out chain graphic, showing channel information, see Animated Play-Out Chain, on page 336.
- 'Now and Next' information box (formed with a component).
  This reads information from an automation play-out device and shows it on screen. Information includes:
  - the name of the current clip being played;
  - the time remaining; and
  - the name of the next clip.
- An **Exception Monitor** widget.

Animated Play-Out Chain

A representation of the channel's play-out chain is shown, which monitors the chain at various points (see Figure 11-5). Each monitor point is annotated with live video, audio bars, and status information and is formed from a component.

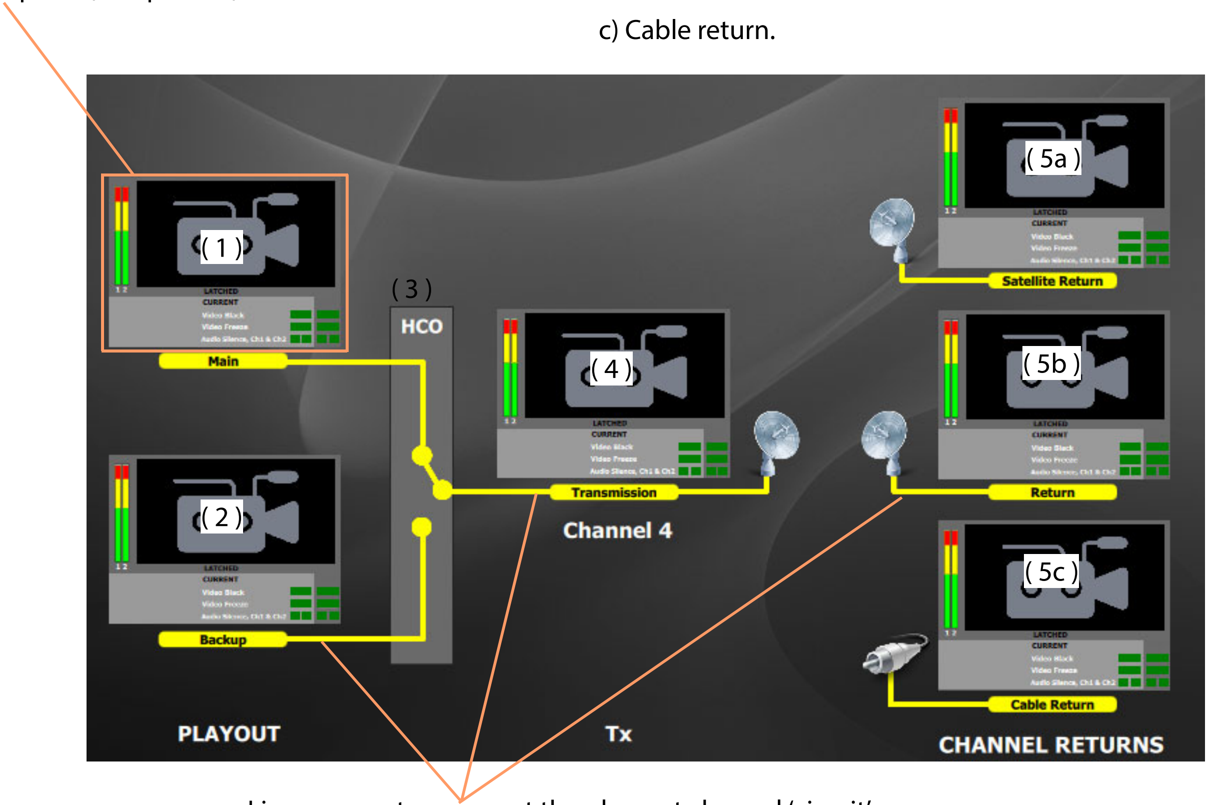
The channel play-out chain graphic depicts:

- 'Main' and 'backup' play-out servers.
- Change-over switch (HCO).
- Transmission signal.
- Channel returns.
- Line segments showing status and interconnection between monitor points.

Play-out chain monitored at:

(1) Output of 'main' play-out server.
(2) Output of 'backup' play-out server.
(3) Selection of 'main' or 'backup' at HCO change-over switch.
(4) Transmission.
(5) Channel reception from:
a) Satellite return.
b) Off-Air return.
c) Cable return.

Monitor point (component)



Line segments represent the play-out channel 'circuit'.

Note: When the project is run, the color of each line segment is linked to an alarm state; the line color then reflects the status of the channel at each point.

Fig. 11-5: Play-Out Chain

Channel Monitor Points

Each channel monitor point is an instance of a ‘channel monitor point’ component, designed for the project and with specific variables set for each component instance on the channel view screen.

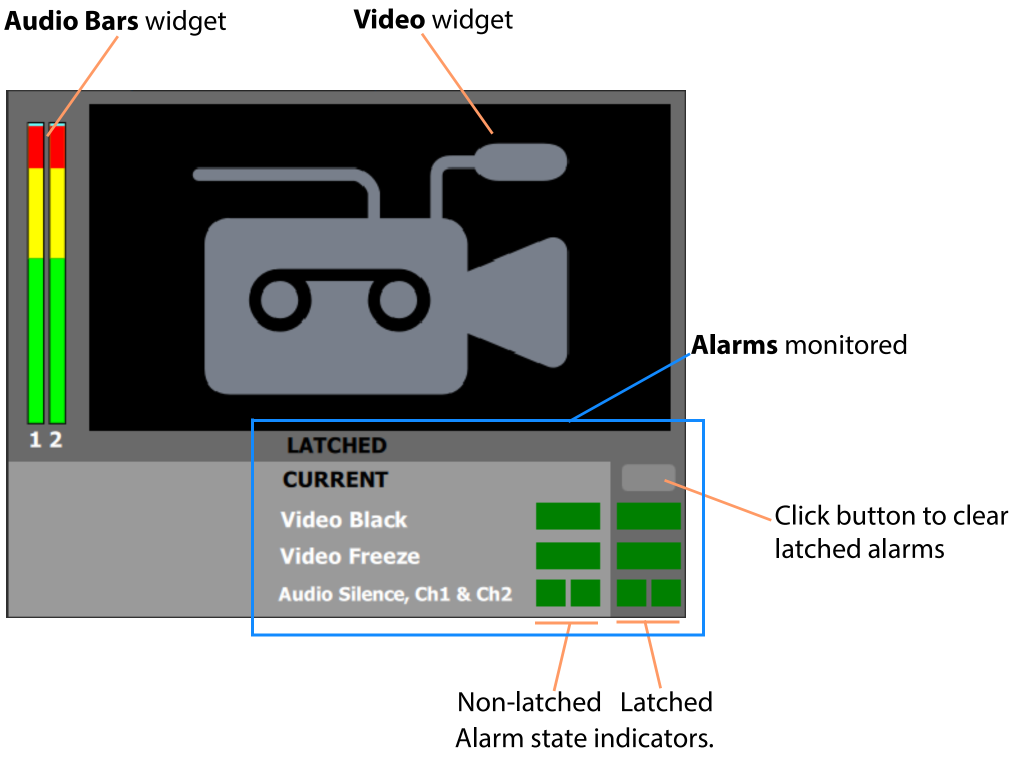


Fig. 11-6: Channel Monitor Point Component

There is an **Alarm** Behaviour for each alarm being monitored (separate Behaviours for latched and non-latched) and the alarm state is reflected with rectangular on-screen indicators.

There is a **Button** widget which clears latched alarms by sending ‘clear’ commands to the Orbit services when it is clicked.

Change-Over Switch

Each play-out server ('Main' and 'Backup') feeds a change-over switch module (HCO). The module monitors the state of its two video input signals; it passes 'input 1' and will automatically switch over to 'input 2' if it detects an error in its received 'input 1', and vice versa.

On the screen schematic, the HCO module is represented by a:

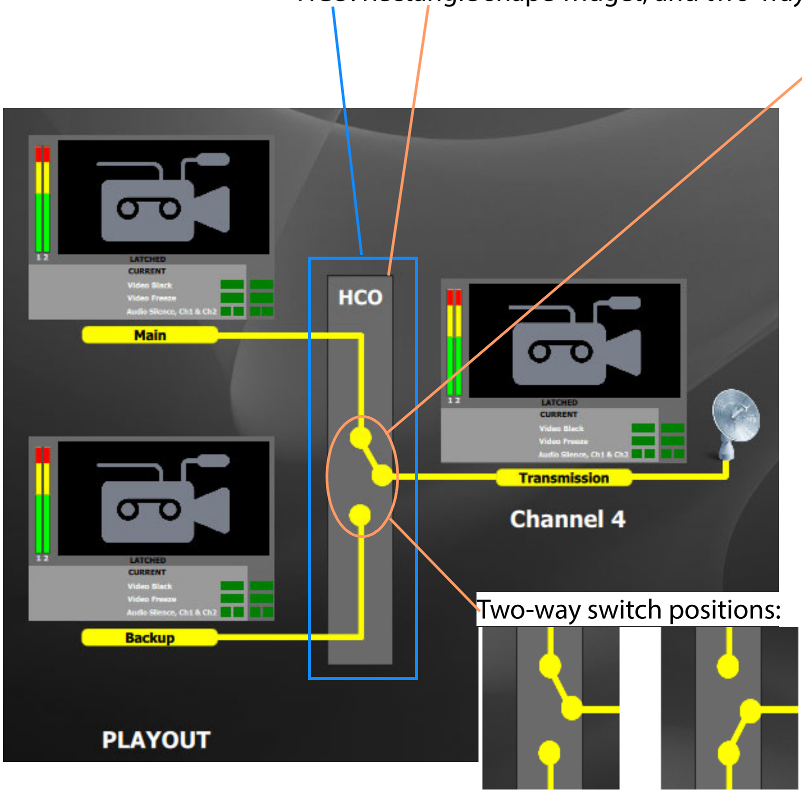
- a rectangle; and
- a graphic representation of a two-way switch.

An **Alarm** Behaviour monitors the state of the HCO module itself.

A **Log Field** Behaviour reads a **Log Field** header from the HCO module to find out which input is selected to be passed. This information is used to control the visibility of one or other line-segments comprising the two-way switch (see Figure 11-7).

.

HCO: Rectangle shape widget; and two-way switch 'line drawing'.



Two-way switch positions:

Fig. 11-7: Change Over Switch

Engineering Information Screens

Racks Screen

The first screen of engineering information summarizes the state of each rack in a system (Figure 11-8) using **Event** Bindings (Event Binding, on page 128) and **Link** Behaviours (Link Behaviour, on page 174).

The hierarchical structure of this part of the example in this section uses variable files (see Example - Screen Link States and Screen Re-Use with Variable Files, on page 243).

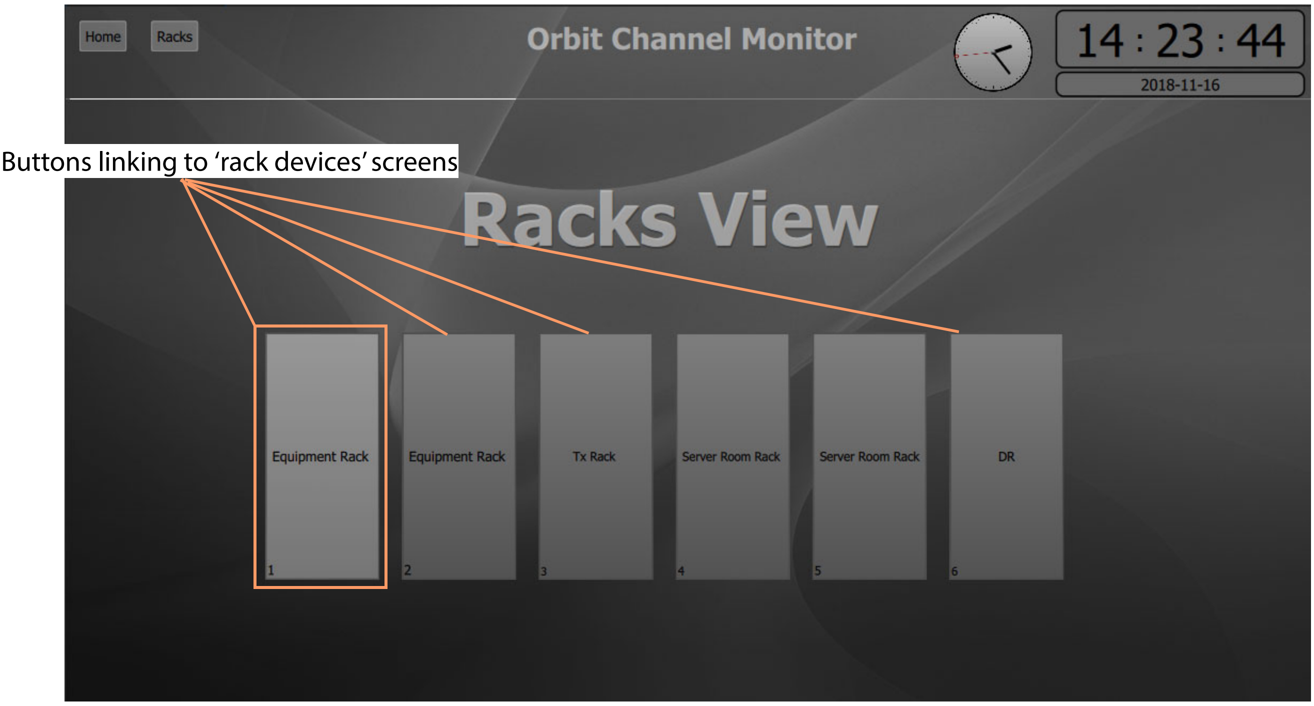


Fig. 11-8: Racks Screen - Access to Rack Devices Screens

Rack Devices Screen

The next level of engineering information summarizes the state of each device in a rack (see Figure 11-9). Similar to the 'racks view' screen, it uses **Event** Bindings and **Link** Behaviours.

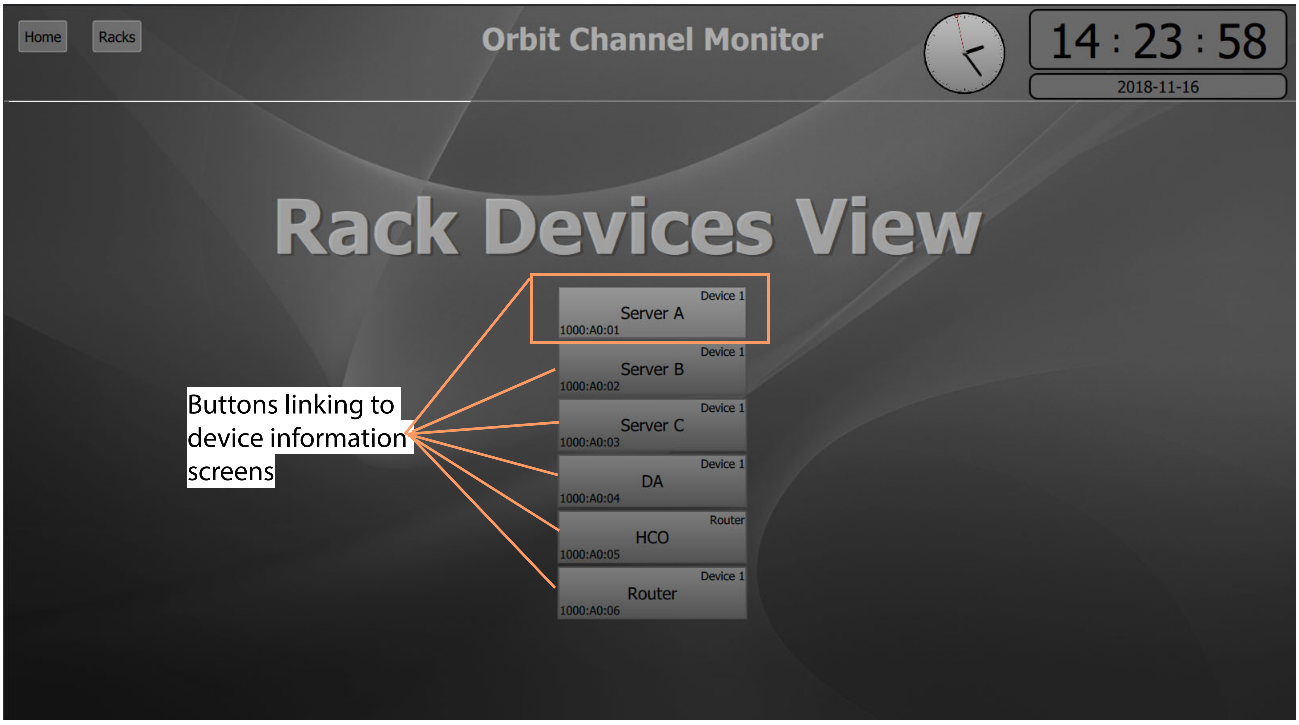


Fig. 11-9: Rack Devices Screen - Access to Device Information

Device Status Information Screen

The lowest level of engineering information summarizes the state of a device (see Figure 11-10). The information is typically laid out in a tabular form. The source of the information is **Alarm** and **Log Field** Behaviours showing various aspects of device status.

- The hash field syntax (#, #Hash Field# Syntax, on page 164) can be used to quickly shows a Log Field value in, for example, a **Label** caption text.
- The angle bracket syntax (< >, <Angle-Bracket> Syntax, on page 166) can be used to flag 'warning' or 'failure' states in, for example, formatted **Label** captions etc. (see Build a Device Information Screen (Low-Level), on page 247).

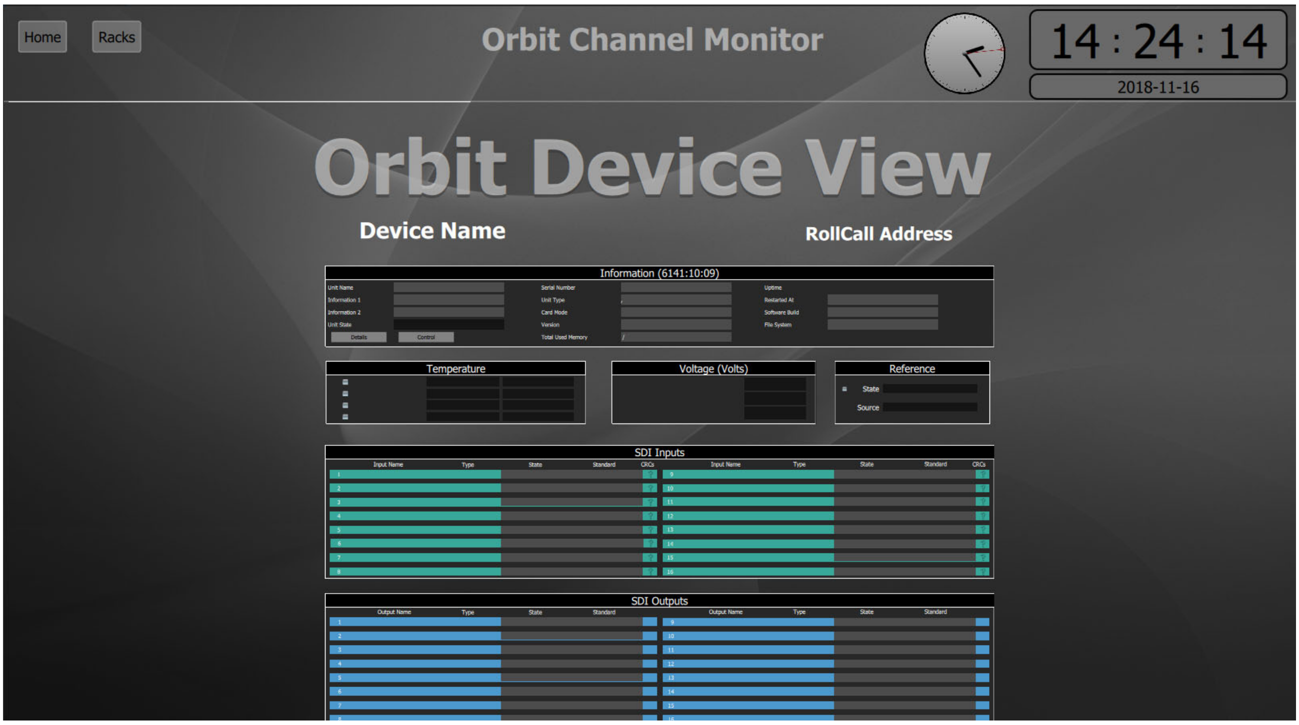
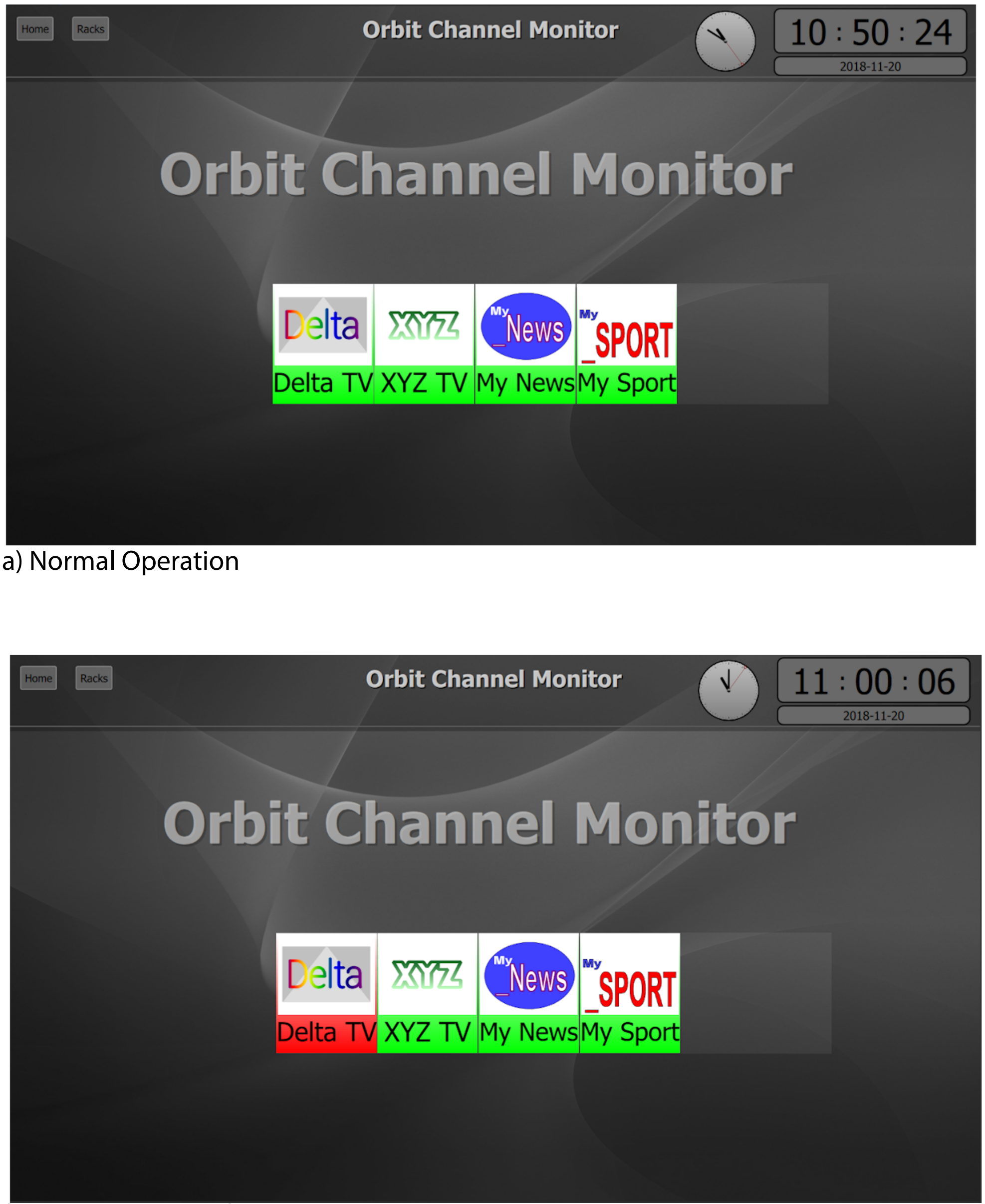


Fig. 11-10: Device Information Screen - Showing All Status Information for a Device

Operation

Top Level

To run the project, open the Top level screen and enter 'Run Mode'.



a) Normal Operation

b) Failure on One Channel

Fig. 11-11: Top Level: a) Normal Operation; b) Failure on One Channel.

Channel View

Normal Operation, All OK

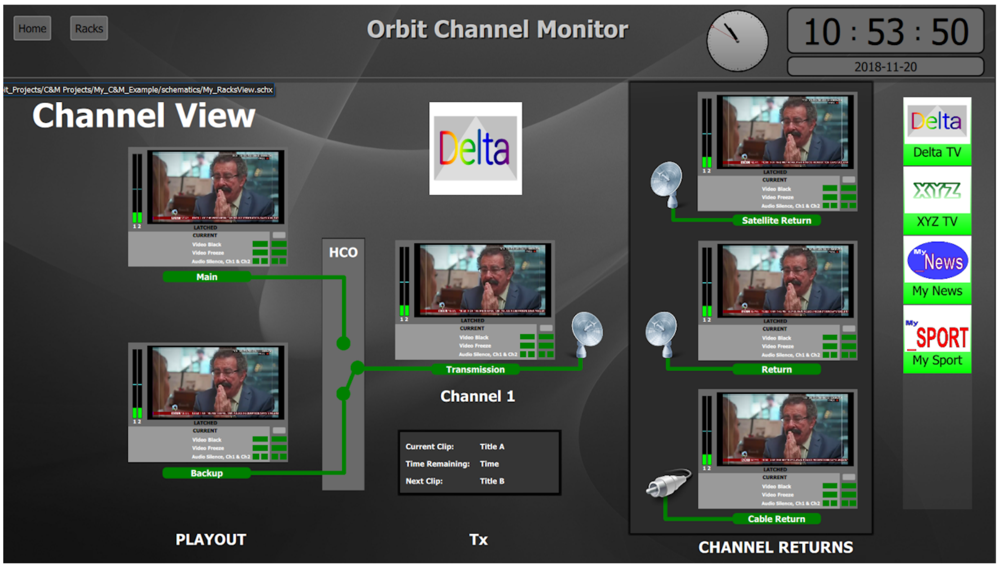


Fig. 11-12: Channel View

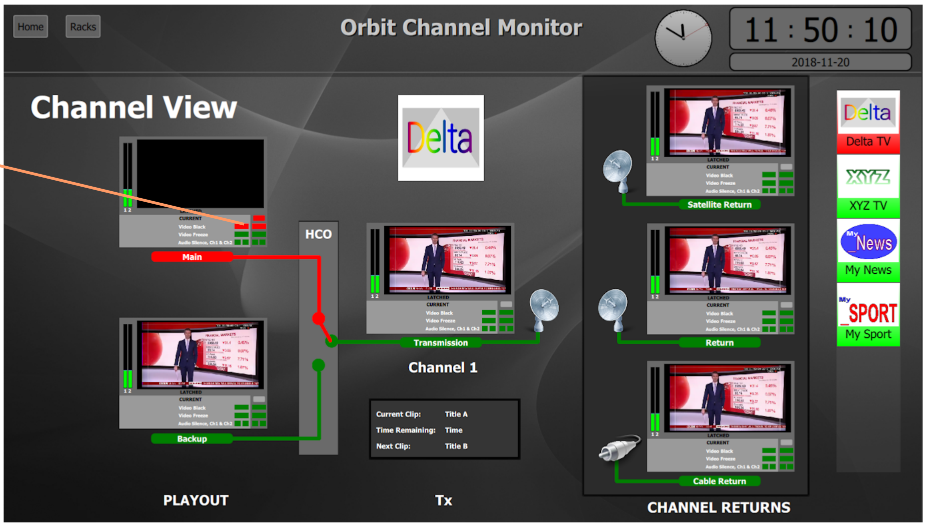
Channel Play-Out Failure

If a failure occurs at the main play-out server, then:

1. An alarm ‘Failure’ condition is reported by the server.
2. The ‘failure’ condition is detected in the Orbit screen is shown:
   - in the alarm state indicators in the ‘Channel Monitor Point’ component; and
   - on the corresponding line segments.

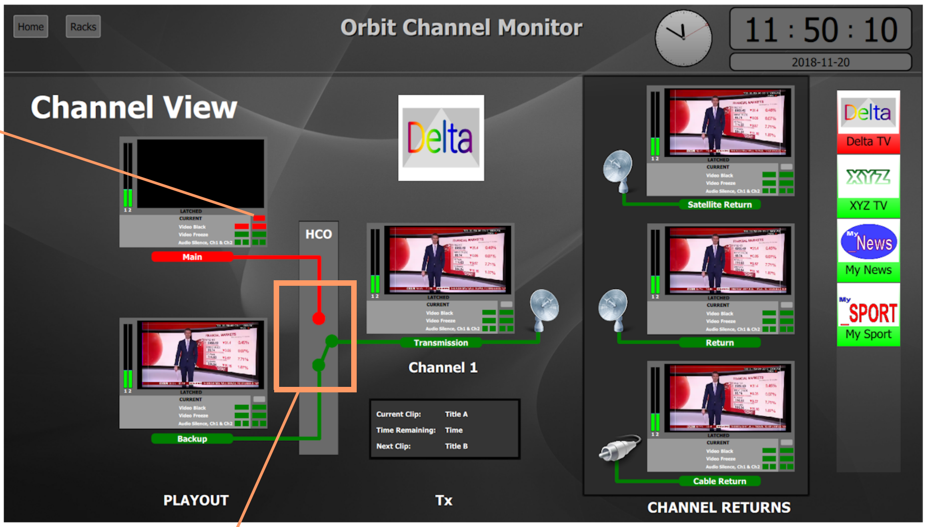
   (See Figure 11-13a.)
3. The physical HCO module detects an error at its input and switches over to using its second input.
4. The physical HCO’s ‘input selected’ status is read by the Orbit project which sets the state of the HCO two-way switch graphic accordingly. (See Figure 11-13b.)

Video black alarm



a) Failure on main play-out server

Click to clear
latched alarm



b) HCO switches over to backup play-out server

Fig. 11-13: Channel View:
a) Main Payout server Fails.
b) Channel Play-out graphic shows Backup Play-out Server being Used.

When the failure condition is cleared, the main play-out server output is used by the HCO and the Orbit screen reverts to its ‘normal’ view.

5 Click the button on the corresponding ‘channel monitor point’ component to clear any latched alarm failure state.

6 Click the ‘Home’ button to go to the top level screen.

A Design Tips and Shortcuts

Summary

Design Tips and Shortcuts

Some useful design tips and shortcuts are presented below for use when designing Orbit MapView screens.

Design Tips

Design Structure

As with any design, careful planning and a structured approach work the best. A design can be got working more easily and subsequently modified as required more simply with a thought-through structured design.

Break down functionality into easily-testable blocks and sub-blocks.

Manage Access Control

Implement 'users' for projects with appropriate permissions.

Keep records of master 'admin' passwords and user names.

Keep master copies of an Orbit project.

Project Structure

Use a hierarchy of screens.

Re-use screens and components to reduce the number of source files.

Note: Project items can be imported from other Orbit projects.

Use components to capture commonly-used graphical arrangements.

Use higher-levels of a hierarchy to over-ride variable values at lower levels. This will help when making changes to a project.

Use variables for flexibility.

Keep variables for different component instances in variables files.

Naming

Use clear and concise names when naming screens, components and variables.

Use a naming convention.

Variables

Use Orbit variables for more flexible screen designs and re-using components.

Variables in Orbit are powerful features. They can be defined at lower levels of a design and then over-ridden at higher levels. They enable a (generic) screen to be re-used many times, for example, for different RollCall addresses, multiviewer input numbers, Log Fields, or channels/ devices etc.

The scope of a variable is the component, or screen, on which it is declared. In the component/ screen hierarchy of a project, variable values can be set and over-ridden.

A Variable can be:

- Defined at run-time when the Orbit client application is run.
- Defined in a high-level screen.
- Defined on a low-level screen.
- Defined on a Component.

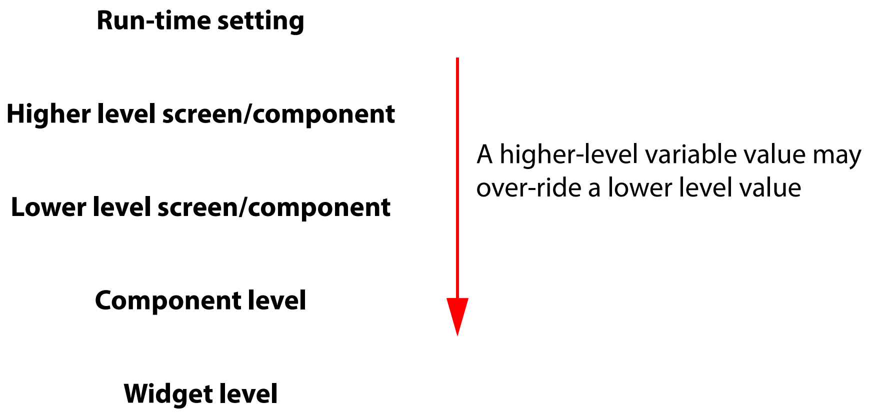

Run-time setting
Higher level screen/component
Lower level screen/component
Component level
Widget level
A higher-level variable value may over-ride a lower level value

Fig. A-1: Over-riding Variables

The determined variable value may be assigned to a Behaviour value or widget property.

Variable File

Use the **Variable File** feature of Orbit to hold sets of variable values for Behaviours.

Components

Use Orbit components for re-using graphical screen designs.

Custom Logic

Use the **Custom Logic** feature of Orbit to encapsulate frequently-used functionality.

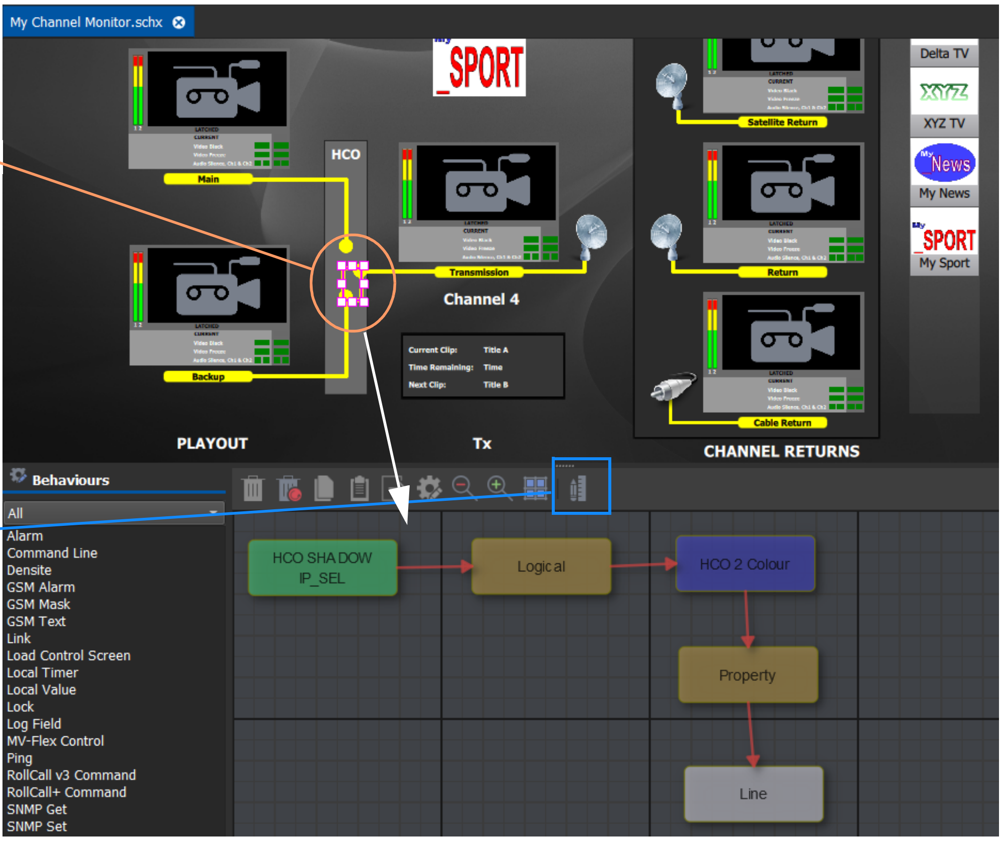
Exercising

Exercise parts of the design using the Test button in the Behaviour and Bindings graphical editor. This enter Run Mode and focuses on the selected widget. See Figure A-2.

a) In Design Mode:

Select a graphical item (Line Segment is shown here) to show its associated Behaviour and Bindings'.

Click **Test** button for 'Test Mode'



b) In Test Mode:

Behaviour values and widget Property values are shown.

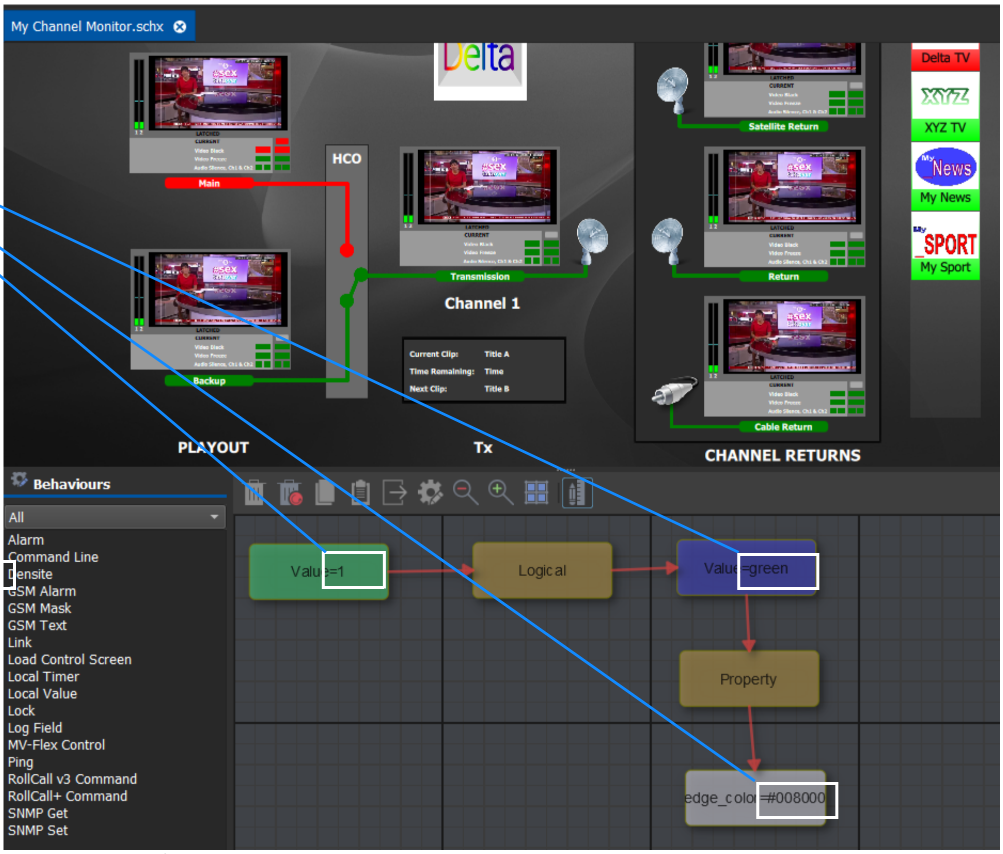


Fig. A-2: Exercise Parts of Designs:
a) In Design Mode Select Focus;
b) In Test Mode - Run-Time Behaviour Values Shown.

Debug

The ‘logic’ behind the scenes of a set of Orbit MapView screens is defined with variables, Behaviours and Bindings.

Incorporate some debug messages(e.g. Label widget) at key points in a project. When the debug messaging is not needed, hide them using a variable and widget visibility property.

Design Shortcuts

Curly Bracket { } Syntax

Use the curly bracket syntax to add the value of an Orbit string or address variable into a property value.
For example, into a **Label** widget caption.

{Address}

Hash Field # # Syntax

Use the hash field short-hand to add the value of a Log Field header from a RollCall address to a property value. For example, to a Label widget caption.

#Address, LOG_HEADER#

See #Hash Field# Syntax, on page 164.

Angle Bracket < > Syntax

Use the angle bracket syntax to add the value of a Log Field header from a RollCall address to a property textual value and to represent the Log Field alarm state via text font color. For example, to a Label widget caption.

<Address, LOG_HEADER>

See <Angle-Bracket> Syntax, on page 166.

Copy a Device Address to Clipboard

1 Right-click on a device in the **Network View**.

2 Select 'Copy Address'.

The device's address (for example, 1000:B0:01) is copied to the clipboard.

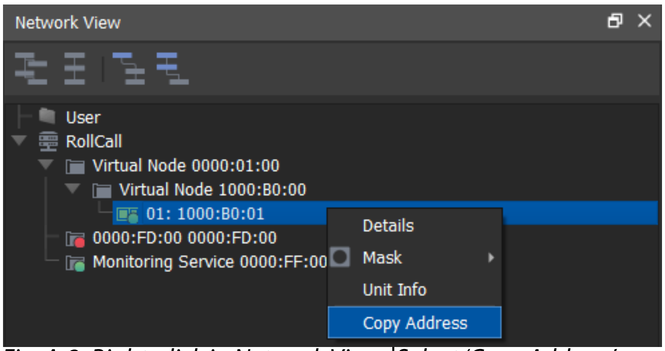


Fig. A-3: Right-click in Network View, |Select 'Copy Address'

Copy a Device Parameters

1 Open an Orbit MapView screen in Design Mode and open the 'Behaviours and Bindings' graphical editor.

For a Grass Valley Densité device, in the **Network View**:

2 Right-click on the Densité device and select 'Display Card Parameters'.
The device's Densité parameters are listed.

3 Select a parameter in the list, right-click on it, and select 'Copy as Behaviour'.

Parameter information is copied to the clipboard.

4 In the 'Behaviours and Bindings' graphical editor, click the 'Paste' button.
A Behaviour is automatically instantiated to bind to the (remote) device parameter value.

Copy a Device Log Field to Clipboard

1 Right-click on a device in the **Network View**.

2 Select 'Details'.
The device's **Details** window is shown.

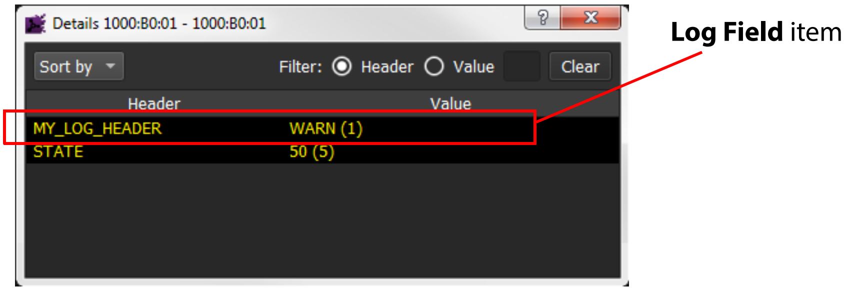


Fig. A-4: Details Window

3 Right-click on a **Log Field** item
and a select an option under the'Copy to Clipboard' menu item:

- **Header and Value** - for example, MY_LOG_HEADER=WARN
- **Header** - for example, MY_LOG_HEADER
- **Value** - for example, WARN

The item is copied to the clipboard.

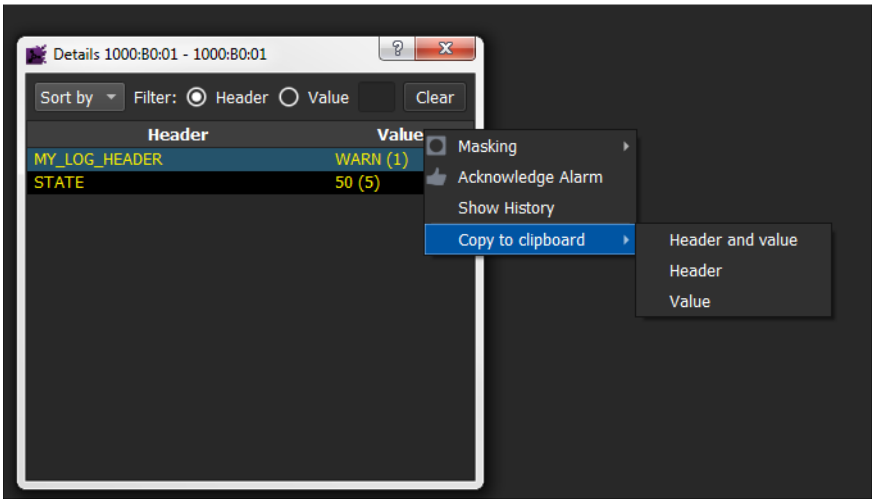


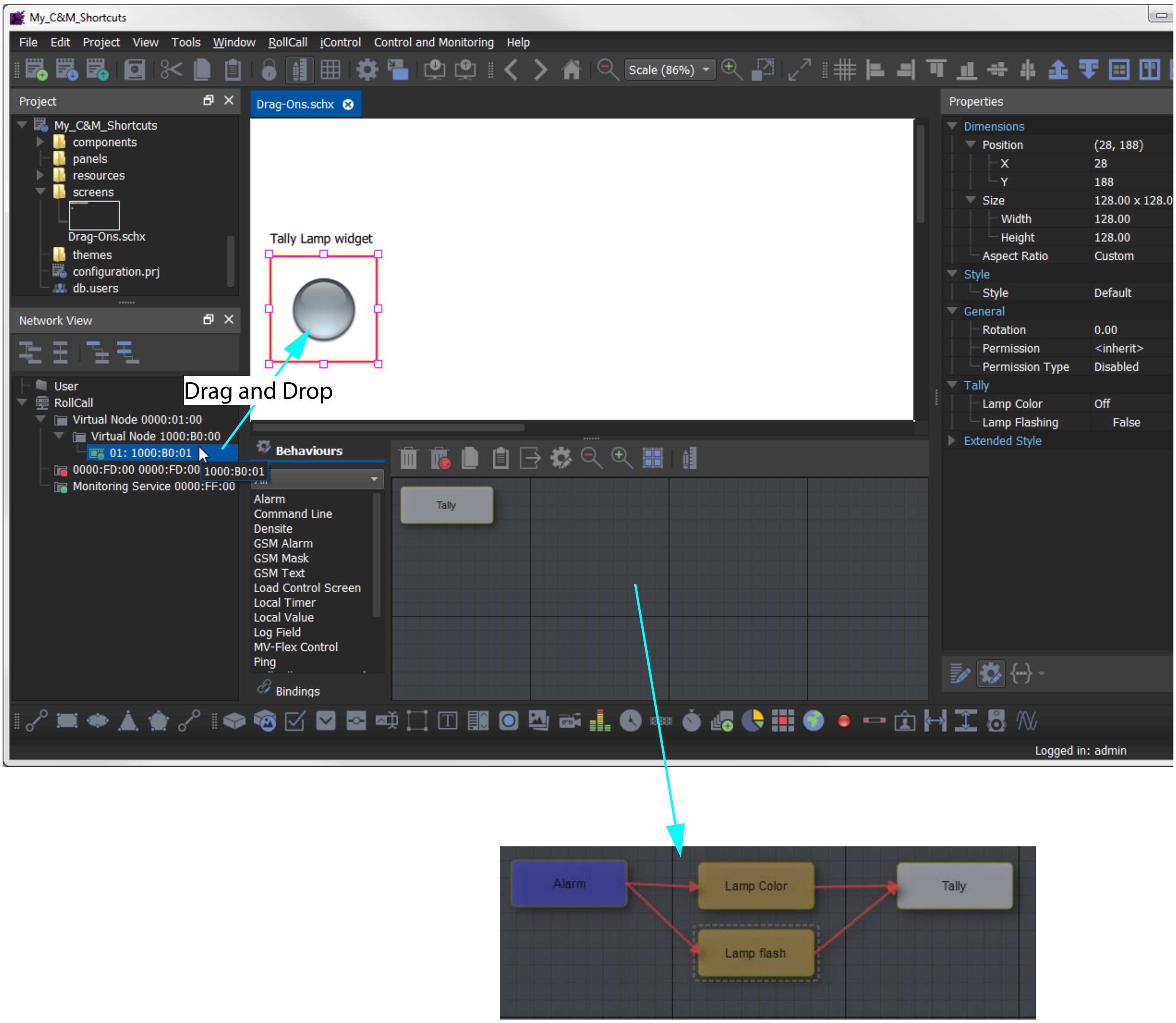
Fig. A-5: Right-click in Details Window

Drag/Drop Device onto Tally Lamp Widget

1 Drag a device from the **Network View** and drop onto a **Lamp** widget.

The following is automatically done:

- An **Alarm** Behaviour is instantiated to monitor the device's address.
- The **Lamp** widget's color and flashing are configured to be controlled from the **Alarm** Behaviour using **Property** Bindings.



Properties	
Dimensions	
Position	(28, 188)
X	28
Y	188
Size	128.00 x 128.0
Width	128.00
Height	128.00
Aspect Ratio	Custom
Style	
Style	Default
General	
Rotation	0.00
Permission	<inherit>
Permission Type	Disabled
Tally	
Lamp Color	Off
Lamp Flashing	False
Extended Style	

Fig. A-6: Drag/Drop Device onto Lamp Widget

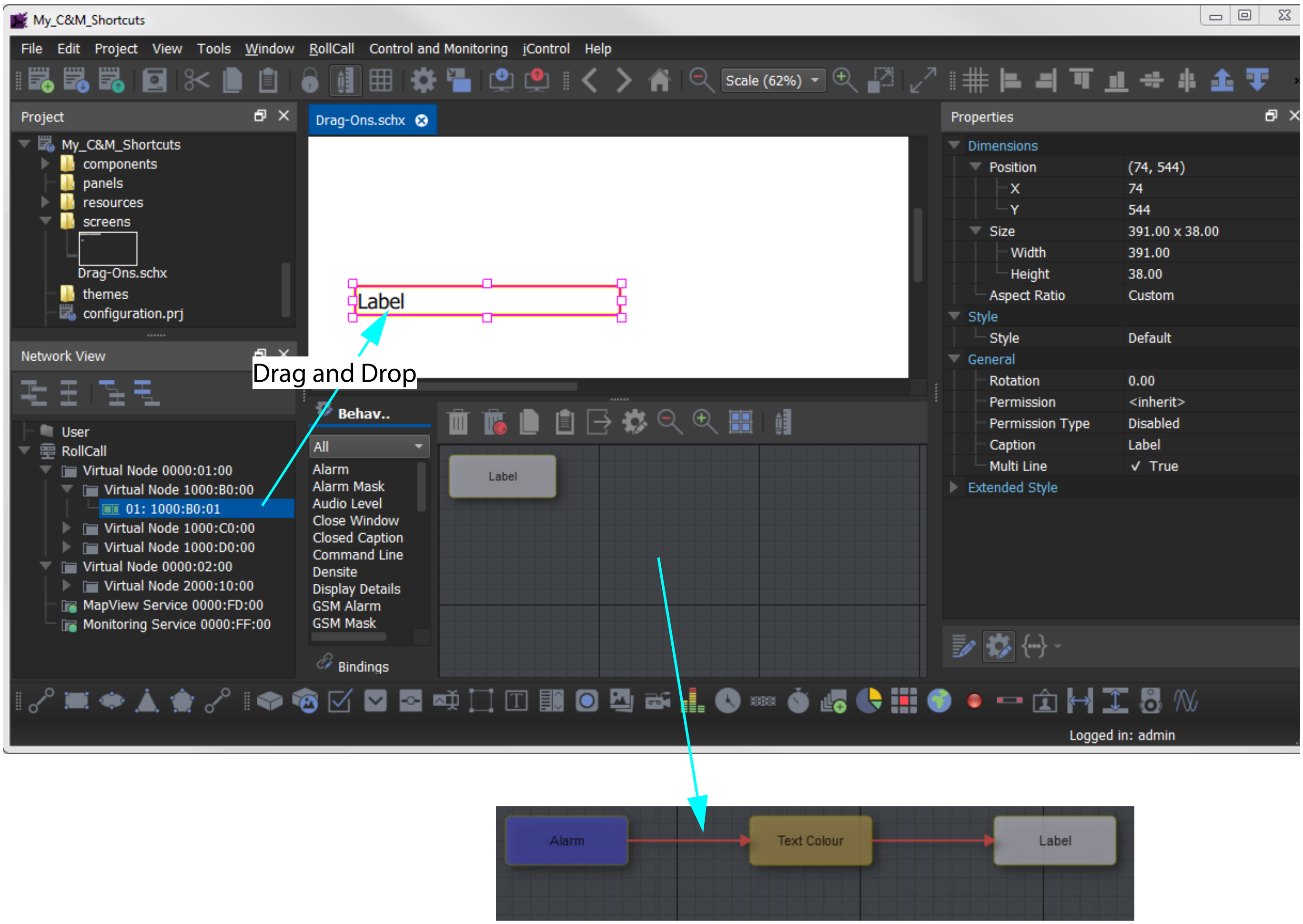
Drag/Drop Device onto Label Widget

1 Drag a device from the **Network View** and drop onto a **Label** widget.

The following is automatically done:

- An **Alarm** Behaviour is instantiated to monitor the device's address.
- The **Label** widget's caption text color is configured to be controlled from the **Alarm** Behaviour using a **Property** Binding.

  For text alarms, the alarm text is shown on a caption.



Dimensions
- Position: (74, 544)
  - X: 74
  - Y: 544
- Size: 391.00 x 38.00
  - Width: 391.00
  - Height: 38.00
- Aspect Ratio: Custom

Style
- Style: Default

General
- Rotation: 0.00
- Permission: <inherit>
- Permission Type: Disabled
- Caption: Label
- Multi Line: ✓ True

Extended Style

Fig. A-7: Drag/Drop Device onto Label Widget

Shortcut Keystrokes

These shortcut keystrokes are available when editing screens or components (or walls or tiles) in a schematic editor in Orbit:

Table A-1: Shortcut Keystrokes

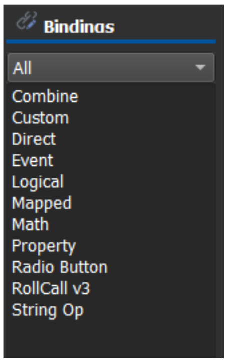
Shortcut Keystroke	Action
Ctrl + C	Copy.
Ctrl + X	Cut.
Ctrl + V	Paste.
Ctrl + S	Save.
Alt + B	Toggles the Orbit **Behaviours** window when graphic-editing.
Alt + V	Opens the **Variables** dialog
Ctrl + Alt + V	Create a variable value for a selected property.
Alt + Enter	Toggle full screen mode.
Ctrl + Alt + Enter	Toggle design mode / run mode.
Ctrl + Mouse wheel	Zooms in/out at the location of the mouse.
Mouse wheel	Scrolls graphical schematic view up/down.
Shift + Mouse wheel	Scrolls graphical schematic view left/right.
L + Drag Widget	With a selected widget, this keystroke locks a widget into moving either in a horizontal or in a vertical direction when moving it.
Shift + Space Bar	Enters widget connection point mode.
Shift + drag widget into group	Adds a widget to a group.
Alt + click on overlapped widgets	Cycles selection through overlapping widgets. Allows widgets that beneath other widgets or widgets in a group to be selected.
Click widget in group	Selects the whole group.
Alt + click widget in group	Select a widget inside a group.
Hold down left mouse button and drag.	Selection rectangular area.

B List of Behaviours and Bindings

List of Behaviours and Bindings

This appendix lists Orbit MapView Behaviours and Bindings for use in MapView applications:

Bindings



Bindings

All

Combine
Custom
Direct
Event
Logical
Mapped
Math
Property
Radio Button
RollCall v3
String Op

Fig. B-1: Bindings List (in Behaviours and Bindings graphical editor)

Table B-1: List of Bindings

Binding	Description
Combine	Concatenate two or more Behaviour values into one value.
Custom	Binding comprising other Bindings to form a user-defined, complex Binding function.
Direct	Binds a widget value to a Behaviour value.
Event	Detect a user interface event (for example a button click) and performs an action on a target Behaviour.
Logical	Uses a logic operation on two or more input values to form one resultant value.
Mapped	Creates a mapping between two values.
Math	Uses a mathematical operation on one or more input values to give a resultant value.
Property	Binds a Behaviour value indirectly to a selected widget display property.
Radio Button	Binds a Radio Button widget directly to a Behaviour value.
RollCall v3	Binds a RollCall v3 Behaviour to a widget. This understands device menus.
String Op	Performs a string operation on an input value.

Behaviours

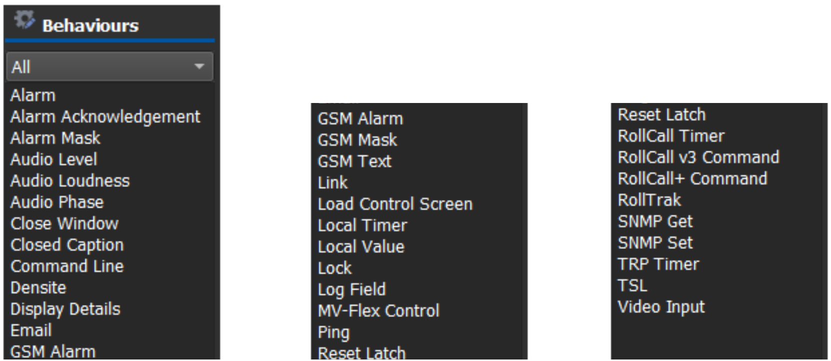


Fig. B-2: Behaviours List (in Behaviours and Bindings graphical editor)

Table B-2: List of Behaviours

Behaviour	Description
Alarm	Connects to an Alarm state or remote Log Field.
Alarm Acknowledgment	Acknowledges an alarm or set of alarms.
Alarm Mask	Mask/Unmask Alarms on devices (units and log headers)
Audio Level	Subscribes to current audio levels.
Audio Loudness	Subscribes to current loudness values.
Audio Phase	Subscribes to audio phase values.
Close Window	Closes current schematic.
Closed Caption	Closed Caption display Behaviour.
Command Line	Runs a separate process on the command line.
Densité	Connects to the Grass Valley Densité REST API.
Display Details	Displays the 'Details' window for a specific RollCall address.
Email	Sends out emails via the Node back-end service.
GSM Alarm	Connects to one or more GSM alarms.
GSM Mask	Masks/Unmasks a GSM alarm.
GSM Text	Connects to one or more GSM text alarms and the text value(s).
Link	Links to another Orbit project file (screen/panel)
Load Control System	Requests Orbit loads a control screen for an address. Address can be fixed or passed as an argument.
Local Timer	Creates a timer local to the screen. Arguments: start, reset, stop, and pause.
Local Value	Values can be stored, client-side, in a screen.
Lock	Provides access to application screen Lock/Unlock function. Arguments: lock, and unlock.

Table B-2: List of Behaviours (continued)

Behaviour	Description
Log Field	Connects to a remote Log Field value.
MV-Flex Control	Controls MV-Flex multiviewer product range.
Ping	Pings an IP address.
Reset Latch	Resets the latched state of an alarm or set of alarms.
RollCall Timer	Listens for RollCall timer updates.
RollCall v3 Command	Connects to a remote RollCall command with RollCall v3 protocol.
RollCall+ Command	Connects to a remote RollCall command with RollCall+ protocol.
RollTrak	Sets or Gets a RollTrak with RollCall v3 protocol.
SNMP Get	Polls an SNMP-capable device for an SNMP MIB OID value(s).
SNMP Set	Set an SNMP MIB OID value on a SNMP-capable device.
TRP Timer	Listens for TRP timer updates.
TSL	Listens for TSL message updates.
Video Input	Connects to a video input of a device proving input status etc.

C Troubleshooting

Troubleshooting

If any problem is encountered with Orbit, then please contact Grass Valley support.

Provide as much information as possible because this is key to problem resolution.

Grass Valley support engineering will use information provided to investigate the root cause of the problem and a fix. They can also advise on any temporary workaround to continue operations.

Information to Provide

Information should include:

• Problem description.	A precise and accurate description of the problem.
• Problem impact.	Impact of the problem.
• Steps performed.	The steps performed when the problem occurred.
• A concise list of steps which reproduce the problem.	If a problem is reproducible with a concise set of steps, then it is far, far easier to reproduce at Grass Valley and hence investigate and resolve etc.
• Any system changes.	Provide details of any changes to a working system that have been done. These may be relevant to the introduction of the problem.
• Version number of Orbit.	This can be found via **Help** > **About**
• The Orbit project used.	This may be zipped up, once Orbit is closed, and sent via email attachment or via ftp.
• Any Log Server CurStat file.	Where applicable when Orbit services are used.
• In the case of an Orbit crash, supply a crash dump file.	File name orbit_xxxxx.dmp ; file size approximately 460MB.
• Any other supporting information.	

Crash Dump Files

Orbit has built-in crash detection; if an internal error is detected, it is trapped by the program and the current program state is stored in a crash dump file (*.dmp). This file can be used by Grass Valley support engineering to facilitate problem investigation. When contacting Grass Valley customer support, inform them of the version number of the Orbit being used (Help > About).

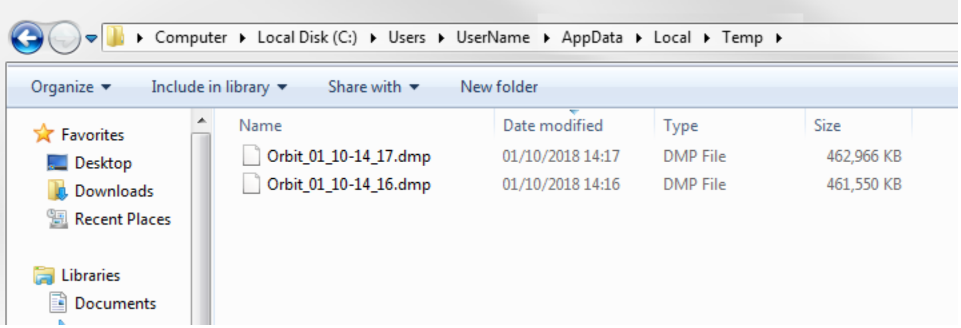
The location of the crash dump files on your computer differs depending on whether it is the Orbit application, or an Orbit service which crashes.

File name format is   Orbit_DD_MM-hh_mm.dmp

where

DD = day of the month; MM = month number; hh = hour (00 to 23); mm = minute

For example, Orbit_01_10-14_17.dmp



Name	Date modified	Type	Size
Orbit_01_10-14_17.dmp	01/10/2018 14:17	DMP File	462,966 KB
Orbit_01_10-14_16.dmp	01/10/2018 14:16	DMP File	461,550 KB

Fig. C-1: Crash Dump File Example

Application Dump File

These files can be found in sub-folder: C:\Users\UserName\AppData\Local\Temp

Orbit Services Dump File

These files can be found in sub-folder: C:\Windows\Temp

Files are named: OrbitNode_DD_MM-hh_mm.dmp

Known Issues

Black Screen when opening Orbit via remote desktop

This has been seen on a few occasions when running Orbit Client over a remote desktop or on Windows Server 2008 where the correct OpenGL drivers are not in place.

Workaround:

Start Orbit using the Windows command script file 'orbit remote desktop mode.cmd' which is found in the 'c:\program files\sam\orbit' installation folder.

Contact Us

Grass Valley Technical Support

For technical assistance, contact our international support center, at 1-800-547-8949 (US and Canada) or +1 530 478 4148.

To obtain a local phone number for the support center nearest you, please consult the 'Contact Us' section of Grass Valley's website (www.grassvalley.com).

An on-line form for e-mail contact is also available from the website.

Corporate Head Office

Grass Valley
3499 Douglas-B.-Floreani
St-Laurent, Quebec H4S 2C6
Canada
Telephone: +1 514 333 1772
Fax: +1 514 333 9828
www.grassvalley.com