DashBoard

FACILITY CONTROL SYSTEM

CustomPanel Development Guide

OGLML and ogScript

Version 9.5

Copyright Ross Video – All Rights Reserved

Thank You for Choosing Ross

You've made a great choice. We expect you will be very happy with your purchase of Ross Technology. Our mission is to:

1. Provide a Superior Customer Experience
- offer the best product quality and support
2. Make Cool Practical Technology
- develop great products that customers love

Ross has become well known for the Ross Video Code of Ethics. It guides our interactions and empowers our employees. I hope you enjoy reading it below.

If anything at all with your Ross experience does not live up to your expectations be sure to reach out to us at solutions@rossvideo.com.

David Ross CEO, Ross Video

dross@rossvideo.com

Ross Video Code of Ethics

Any company is the sum total of the people that make things happen. At Ross, our employees are a special group. Our employees truly care about doing a great job and delivering a high quality customer experience every day. This code of ethics hangs on the wall of all Ross Video locations to guide our behavior:

1. We will always act in our customers' best interest.
2. We will do our best to understand our customers' requirements.
3. We will not ship crap.
4. We will be great to work with.
5. We will do something extra for our customers, as an apology, when something big goes wrong and it's our fault.
6. We will keep our promises.
7. We will treat the competition with respect.
8. We will cooperate with and help other friendly companies.
9. We will go above and beyond in times of crisis. If there's no one to authorize the required action in times of company or customer crisis - do what you know in your heart is right. *(You may rent helicopters if necessary.)*

DashBoard CustomPanel Development Guide

- Ross Part Number: 8351DR-007-9.5
- Release Date: November 03, 2022

Copyright

© 2022 Ross Video Limited. Ross®, openGear®, and any related marks are trademarks or registered trademarks of Ross Video Ltd. All other trademarks are the property of their respective companies. PATENTS ISSUED and PENDING. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, mechanical, photocopying, recording or otherwise, without the prior written permission of Ross Video. While every precaution has been taken in the preparation of this document, Ross Video assumes no responsibility for errors or omissions. Neither is any liability assumed for damages resulting from the use of the information contained herein.

Patents

Patent numbers US 7,034,886; US 7,508,455; US 7,602,446; US 7,802,802 B2; US 7,834,886; US 7,914,332; US 8,307,284; US 8,407,374 B2; US 8,499,019 B2; US 8,519,949 B2; US 8,743,292 B2; GB 2,419,119 B; GB 2,447,380 B; and other patents pending.

Company Address

Ross Video Limited

8 John Street

Iroquois, Ontario, K0E 1K0

Canada

Ross Video Incorporated

P.O. Box 880

Ogdensburg, New York

USA 13669-0880

General Business Office: (+1) 613 • 652 • 4886

Fax: (+1) 613 • 652 • 4425

Technical Support: (+1) 613 • 652 • 4886

After Hours Emergency: (+1) 613 • 349 • 0006

E-mail (Technical Support): techsupport@rossvideo.com

E-mail (General Information): solutions@rossvideo.com

Website: http://www.rossvideo.com

Contents

Introduction 11

About this Guide 11
CustomPanel Overview....11
PanelBuilder 11
CustomPanel Framework....12
Getting Started 15
Building a CustomPanel Application....15

DashBoard Data Model 16

In This Section 16
Device Data Model 16
Data Object Hierarchy 16
Device / Card 17
Parameters....18
Constraints 20
Parameter Structure Objects 22
Parameter References 22
Menus 22
Customizing Menus Using Display Hints....24
Universal Hints 24
Separators, Titles and Layout Hints....24
Array Layout Hints 26
INT16/INT32 Parameters with Choice Constraints....28
Hints for Numeric Parameters with Other Constraints 34
Hints for String Parameters....44
Hints for STRUCT Types 48
Data Types 50
Endianness 50
Number Encoding....50
String Encoding 51
External Data Objects 51
Constraint....51
Arbitrary File 51
Image 52
OGLML Descriptor or Index XML 52
OGLML Documents 52
Containers....52
Contexts 52
OGLML Document Structure 52
OGLML URLs....53
OGLML Descriptor Format....54
Custom Widgets....55
Creating Widgets 55
Widget Samples 58
Descriptor Location 64
Parameter Mapping....65
Using DashBoard Prebuilt Custom Widgets....65
Custom APIs Within CustomPanels 76
Lexical Order and Loading Order....76
Enabling Reuse by Keeping APIs in Separate Files 82

Managing Scope 84

OGLML Reference 86

In This Section 86
General Attributes 86
Using OGP Devices that Support Subscriptions Protocol 89
subscriptions 89
Examples 92
openGear Style Hints 93
Style Hint Reference 93
style Style Hint 94
Component Color 95
Predefined Colors 96
Border Styles 98
Text/Font Styles 98
Icon Styles 99
Tooltip Style 100
Inset Style 100
Background Styles 101
Button Style Modifiers 101
Layout/Container Tags 102
abs 103
borderlayout 105
flow 108
popup 109
pager 110
simplegrid 112
split 112
tab 115
table 116
Top Level Attributes 120
editlock 120
encrypt 121
gridsize 121
keepalive 122
Widget Tags 122
drawer 123
wizard 124
reveal 124
ext 125
exit 126
help 127
image 128
label 129
button 130
browser 130
blank 131
lock 132
memory 132
widget 133
Non-UI Tags 135
api 136
context (device context) 136
subscription 137
meta 138
widgets 139

widgetdescriptor 139
lookup 141
style 143
color 143
ogscript 144
constraint 147
params 155
timer 156
listener 157
task 158
timertask 159
include 160
Device Resource Declarations 160
Resource XML File 160
config 164
constraint 166
card 167
frame 168
menu 168
menugroup 169
statusmenu 170
configmenu 171
params 172
param 172
param (struct) 174
Device Resource Tags 176
menugroup 177
menu 177
param 178
constraint 180
buttonbar 180
editor 181
summary 182
statuscombo 182
Macro Expansion 184
%frame% 185
%device% 185
%slot% 185
%value% 186
%widget% 187
%const% 187
%baseoid% 188
%fully-qualified-id% 188
%panel-path% 188
%app-path% 189
%id% 189
%eval[ogscript]% 189

ogScript Reference 191

About ogScript 191
JavaScript 191
Commonly Used Functions 192
Functions Set in the User Interface 192
multiSetScriptable Object 192
nkScript Object 192
createFileInput 201

createListener....201
createVDCPSender....202
ftp....203
ftpGet....203
ftpListFiles....204
getApplicationPath....205
getAsyncExecById....205
getBrowserById....206
getContextId....206
getFile....207
getFileSize....207
getImageById....208
getPanelPath....208
getPanelRelativeURL....208
hslToColorString....209
http....209
installTimer....210
isClosed....211
jsonToString....211
pasteText....212
addRemoteTrigger....212
ogscript Object....213
addOnClose....218
addRemoteTrigger....218
appendXML....218
asyncExec....219
asyncFTP....222
asyncFTPGet....224
asyncPost....225
cancelTimer....225
copyByteArray....226
createByteArray....226
createFileInput....227
createFileOutput....227
createMessageBuilder....228
createMessageParser....228
debug....229
fireGPI....229
getAllById....230
getAttribute....230
getBuild....231
getComponentsById....231
getCurrentUser....231
getIncludeById....232
getListenerById....232
getModificationDate....233
getObject....233
getPosition....234
getPrivateString....235
getScopedAttribute....236
getSize....236
getString....237
getTimerManager....238
hide....241
installTimer....242
isTimerRunning....243
parseXML....243

putObject....244
putPrivateString....245
putString....246
reload....246
rename....247
reposition....248
repositionByPercent....248
reveal....249
runXPath....250
saveToFile....250
sendUDPAsBytes....251
sendUDPBytes....251
sendUDPString....252
setAnchorPoints....252
setSize....253
setStyle....253
setXML....254
toBottom....257
toTop....257
upload....258
params Object....259
params Functions....259
createCopy....261
createIntChoiceConstraint....261
createLinkedCopy....262
createMultiSet....262
createParam....263
deleteParam....264
getAllValues....264
getConstraint....264
getDeviceStatus....265
getElementCount....265
getIdentifiedConstraint....265
getParam....266
getParam (OID, Index).remove....266
getStream....267
getValue....267
getValueAsString....267
isDeviceOnline....268
isPrivateParamContext....268
replaceConstraint....269
replaceViewConstraint....269
resetAllValues....269
setAccess....270
setAllValues....270
setMenuState....271
setPrivateParamContext....271
setStream....272
setValue....272
setValueRelative....273
subscribe....273
unsubscribe....274
toOid....275
ParamScriptable Object....277
rosstalk Object....278
rosstalkex Object....280
robot Object....281

vdcp Object 281
nkScript Object 282

Appendices **286**

Appendix A: Widget Hint Definitions 286
Appendix B: Reserved Object IDs 288
Reserved OIDs 288
Reserved MFC and DashBoard Connect (slot 0) OIDs 291

Introduction

About this Guide

The CustomPanel Development Guide is part of the DashBoard Help Guide series. These guides aim to help you get the most out of your DashBoard control management system.

DashBoard Help Guides include the following:

- DashBoard User Guide / Help – The complete reference guide for DashBoard.
- DashBoard CustomPanel Development Guide (this guide) – Learn how to develop custom panel applications within DashBoard.
- DashBoard Server and User Rights and Management User Manual- Provides general information on the DashBoard server, user rights, functions, and possible applications.
- NK Plugin Guide – Learn about NK plugins.

This guide describes the tools available for developing CustomPanel applications within DashBoard.

The following sections are included:

- DashBoard Data Model – An overview of how data and UI elements are stored in DashBoard.
- OGLML Reference – Describes OpenGear Layout Markup Language, which is an XML specification for describing how UI elements are presented within the DashBoard client.
- ogScript Reference – Describes how to use ogScript, a JavaScript-based scripting language, to define advanced behavior of CustomPanel applications.

CustomPanel Overview

CustomPanels are applications which run within the DashBoard client. These may be served up by a device directly, or created by a user using DashBoard's PanelBuilder feature, by writing XML code, or a combination of both. CustomPanels may integrate control of multiple connected devices to provide complete solutions to many workflow problems.

PanelBuilder

PanelBuilder is a DashBoard tool for creating custom interfaces for products from Ross Video and partner companies, such as openGear cards, DashBoard Connect devices, CamBot robotic camera systems, XPression graphics systems, Ultritouch, and Carbonite and Vision Production Switchers.

PanelBuilder allows users to create custom control interfaces with any combination of openGear control and monitoring parameters from any combination of openGear cards and DashBoard Connect devices. Users can build graphical navigation layouts based on signal flow or equipment

location for efficient device and signal monitoring. Custom control panel layouts can provide user, or function specific control windows for specific events or situations that require quick access to various parameters from multiple devices.

Benefits:

- Create custom control panels. By eliminating unused controls, the operator can work with an uncluttered, efficient GUI that's perfect for the task at hand.
- Group various controls together from multiple products. Focus on the production, not how it's being produced.
- Create graphical navigation layouts. Present an overview of your facility with simple status indicators that can be drilled into to get to the details.

With CustomPanels, you can:

- Allow your operators to focus on the production, and not on the equipment being used. This is especially useful when operators are experts in what the production needs to be, but not how it's made such as in a House of Worship, School, or Corporate setting.
- Support a new workflow using existing equipment. For example, you can select, preview, and display static graphics using a Ross Video Master Control MC1-MK.
- Create a Network Operations Center view of geographically dispersed production equipment, with system health status aggregating up through each level so that you can quickly drill down to where the trouble is when faults occur.
- Integrate control of multiple devices into a single, logically laid out control surface. For example, you can trigger graphics, video servers, and transitions from the same interface.
- Control other vendors' equipment. With over 50 openGear and DashBoard Connect partners, it's quite likely that the equipment you want to control already understands DashBoard. Otherwise, advanced users can take advantage of PanelBuilder's rich and powerful scripting support to communicate with third-party equipment using UDP.

CustomPanel Framework

Applications built in DashBoard's PanelBuilder are referred to as CustomPanels. Application development in DashBoard employs a number of complementary technologies to provide user interface applications. These include:

- openGear Protocol (OGP)
- Resource XML files
- openGear Layout Markup Language (OGLML)
- ogScript
- Other control protocols (such as VDCP, RossTalk, etc.)

The openGear ecosystem, in general, consists of Devices (such as openGear Cards, or stand-alone products) and the DashBoard client. Devices communicate via network connection, and in the case of openGear cards, through a CANBus interface.

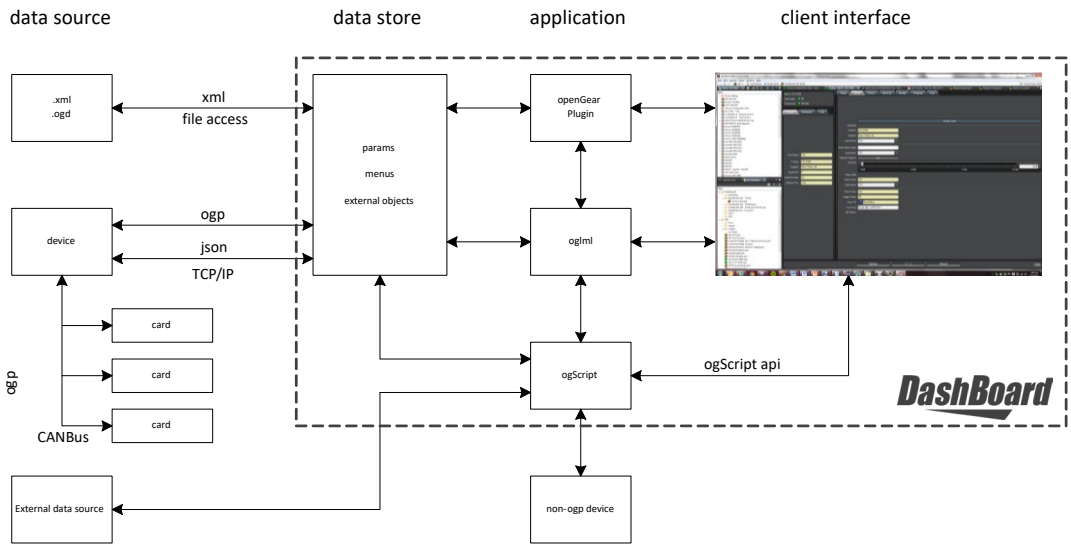


Figure 1 – DashBoard Application Framework

DashBoard Panel applications consist of a number of elements that the designer uses to create an application. These are:

- Data sources
- Internal data store
- Application
- Client Interface

Data Sources

Data may be sourced from several sources. These include:

- Physical devices connected via ogp
- XML files (.xml or .ogd)
- OGLML document with embedded parameter XML data
- Other external data sources

DashBoard manages synchronization between XML sources and, via OGP, physical devices. The data is stored in DashBoard's internal data store. The details of OGP and JSON protocols are available to registered openGear partners, and are detailed in ***openGear Development Guide Part II - Software (8200DR-06)***.

External data sources, not connected via OGP or a DashBoard xml file, must be managed by the user application via ogScript.

DashBoard allows for multiple data sources to be connected to any application. This allows for multiple devices in addition to local parameters and resources to be incorporated into a CustomPanel application.

Datastore

DashBoard maintains an internal data store of information. Using OGP or JSON protocol, DashBoard retrieves information about the descriptor and value of parameters, menus, and external objects. Any changes to the Data store from the client or application is transmitted back to the device. Any changes to the Data store from the device are propagated to the Client. Code may

be triggered when a parameter changes based upon an ogScript `onchange` event registered against the parameter.

Application

The application can be implemented using a variety of tools, depending upon the particular requirements. The application uses the data store to access device information. The following tools are available for developing applications:

- **openGear plug-in:** The basic plug-in automatically generates a user interface based upon the parameters and menus defined in the data store. The plug-in also supports OGP messaging to allow other basic device control.
- **OGLML:** OGLML is a markup language that may be used to create CustomPanel control layouts within DashBoard, beyond the default control layout provided by the openGear plug-in. Applications built in OGLML may include customization of location, size, and appearance of controls. The controls in an OGLML application manipulate parameters stored in the data store.
- **ogScript:** ogScript provides a JavaScript engine to extend the capability of OGLML-based applications. ogScript may also be used to access external data sources (either file or network-based) as well as provide for custom interface to non-OGP devices.

Client Interface

The application is presented within the DashBoard client. DashBoard provides services to display the application, interface with devices, and maintain the data store. DashBoard also provides mechanisms for device discovery, logging, and alarms, and features an interactive GUI named PanelBuilder for the creation of CustomPanel applications.

openGear Protocol

openGear Protocol (OGP) is a basic communication protocol between DashBoard and devices. It provides a mechanism to communicate the basic Data Model, manage parameter changes and describe a basic user interface. With OGP, devices can present a rich user interface using a standardized layout.

There are several variants of OGP, the details of which are described in in ***openGear Development Guide Part II - Software (8200DR-06)***, available to registered openGear partners. The knowledge of the details of the protocol mechanics is not required to develop applications within DashBoard; OGP is simply a mechanism which communicates the Data Model between devices and DashBoard.

Resource XML File

The structure of a device's parameters and menus may be expressed in XML format. This file can be generated in DashBoard from an existing device by right-clicking the device and selecting "Save Configuration to file". This will generate a "`.ogd`" file containing the XML representation of the device.

A resource XML file is also generated by PanelBuilder, if "External Data Source Panel File" is selected when creating the CustomPanel. This file will be given the extension `.xml`.

It is also possible to declare resources directly within an OGLML document using the Resource XML syntax.

openGear Layout Markup Language (OGLML)

OGLML is an XML layout language which augments OGP by providing a set of tools to customize the layout and behaviour of a user interface presented in DashBoard. An OGLML document also allows controls from multiple devices to be combined into a single user interface,

called CustomPanels. CustomPanels may be designed interactively using DashBoard's internal PanelBuilder feature. PanelBuilder provides a GUI to customize the user interface, and generates an OGLML document.

When a new CustomPanel file is created within DashBoard's PanelBuilder, an OGLML file with an extension `.grid` is created.

OGLML is strictly a layout tool for tailoring the presentation of a device's user interface within DashBoard. It simply specifies how a devices' resources are displayed, and relies upon resources in the data store to provide the values for the content. The data store must be backed by a data source, through one of the mechanisms discussed above.

ogScript

ogScript is a programming language developed to interact with DashBoard-enabled devices. It uses JavaScript functions, syntax, and primitive object types. To enable CustomPanel developers to interact with panels and devices, ogScript adds some new global objects to JavaScript. Most JavaScript works in ogScript scripts, although you might run across an occasional item that does not work.

ogScript may be embedded into an OGLML document to add additional functionality based on a set of trigger events (for example, when a page loads, when a parameter changes, mouse clicks, etc.). There are a number of API definitions to allow control of DashBoard's features, access to the data store, and connect to external devices and data sources.

Getting Started

Building a CustomPanel Application

There are several steps in creating a CustomPanel application. The easiest way to get started is to interactively design a layout with DashBoard's PanelBuilder. The basic steps involved are:

- Define data sources
- Define local parameters
- Add controls to the layout in PanelBuilder
- Edit OGLML file for fine-tuning
- Add ogScript to the CustomPanel for advanced functionality

PanelBuilder is an interactive tool that allows quick and easy layout of control; its output is an OGLML document (with a `.grid` extension).

Refer to ***DashBoard User Guide*** help topic or the ***DashBoard User Guide (8351DR-004) PDF*** for detailed instructions on building CustomPanels in PanelBuilder.

DashBoard Data Model

In This Section

This section describes the underlying data model for openGear and DashBoard Connect devices.

This section includes the following topics:

- Device Data Model
- Customizing Menus Using Display Hints
- Data Types
- External Data Objects
- OGLML Documents
- Custom Widgets
- Custom APIs Within CustomPanels

Device Data Model

This section includes the following topics:

- Data Object Hierarchy
- Device / Card
- Parameters
- Constraints
- Parameter Structure Objects
- Parameter References
- Menus

Data Object Hierarchy

DashBoard stores a device's data representation in an object hierarchy.

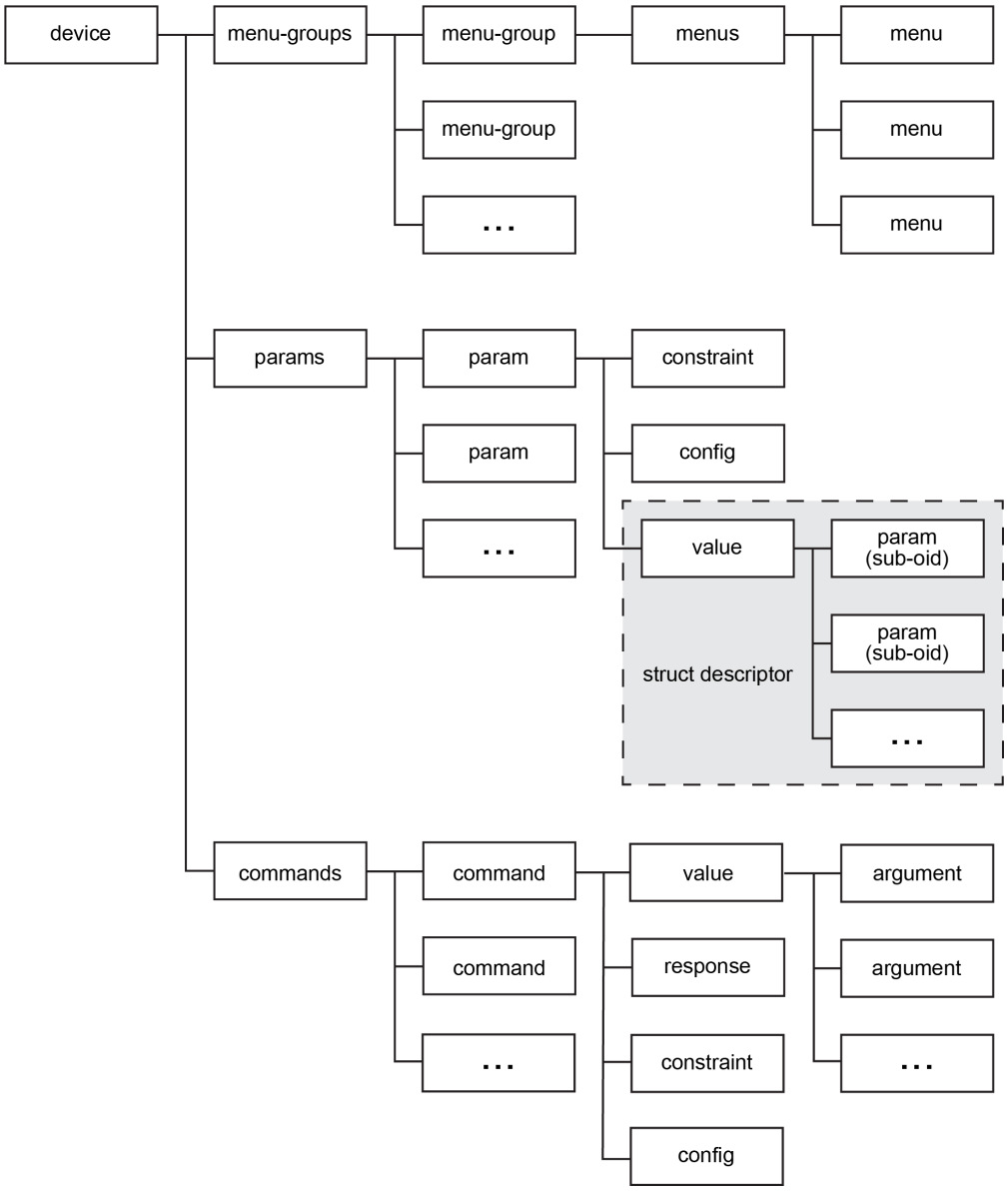


Figure 2 - Data Object Hierarchy in DashBoard

This hierarchy is explicitly exposed in the XML representation. OGP does not explicitly reference the data through the object hierarchy, but individual data elements may be accessed via their OIDs.

Device / Card

All information regarding a device is encapsulated within the device object. This is encapsulated with a `<card>` tag in the XML representation. Each node in the DashBoard tree is treated as an independent device object. The device object contains a list of parameters and menu-groups.

Each device node in the DashBoard tree has a unique **node-id**. This **node-id** is used by DashBoard to reference parameters from multiple devices within the same client interface. The **node-id** can be determined by selecting the node in the DashBoard tree and selecting “View Connection Information” from the context menu.

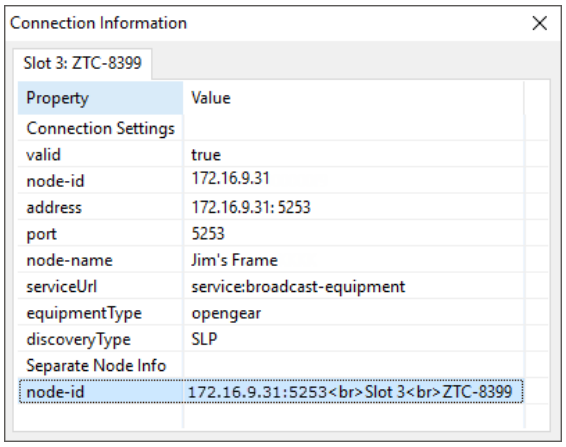


Figure 3 - Connection Information

Parameters

The configuration and state of any device can be represented by a list of parameters holding information about the device, including:

- **identification:** device type and supplier name, software revision, etc.
- **status:** alarms, voltage, current, temperature, input signal presence and format, etc.
- **configuration:** user-specified setup parameters (gain, delay, output video format, etc.)

Each parameter is identified with an Object Identifier (OID), and consists of two parts: the **descriptor** and the **value**. The **descriptor** defines the structure of the data, and the **value** is the content, which is dependent on the descriptor. The descriptor may also specify a **constraint**, which limits the **value** to a certain set of valid values.

Object Identifiers (OIDs)

Each parameter is identified by a unique object ID (OID). There are 2 types of OIDs supported: numeric and string. All devices must support numeric OIDs, and may optionally support string OIDs. However, use of meaningful string OIDs is strongly recommended for new designs, as it clarifies code and simplifies the development of CustomPanels. Handling of Numeric and String OID parameters utilize different message types. Devices implementing String OIDs must support both message types.

Numeric OIDs

Numeric OIDs are 2-byte integers and referenced in this document as a 16-bit hex value, for example: `0x0105`. In JSON messaging, numeric OIDs are encoded as strings. For example, the OID `0x0105` is encoded as the string `"0x105"`.

String OIDs

String OIDs allow text-based parameter identifiers, and must follow the following encoding rules:

- Must not contain spaces
- May only contain the following characters: a-z A-Z 0-9 .(dot) _ (underscore)
- Are case sensitive
- There is no set limit to the String OID identifier length; however, string OIDs over 255 characters cannot be carried over CAN or TCP/IP binary protocol.

A string OID identifier should not be confused with the parameter name. A string ID is the

variable name, the Parameter Name is the display name for the parameter. For example a parameter may have the OID “mle.2.keyer.3.ckey-state” and the parameter name could be “Chroma Key”. Software refers to the value “mle.2.keyer.3.ckey-state”, but the default label on the DashBoard GUI would be “Chroma Key”.

Descriptors

Parameters are defined using a ***descriptor*** containing its name, data type, data length, constraint (set of permitted values) and other information. When DashBoard first contacts a device, it requests the list of parameters for that device, and the descriptor for each parameter. This information is used to create an appropriate user interface for the device and to properly interpret and display parameter values reported by the device.

In JSON messaging, descriptor objects are identified by the naming convention `_d_`*oid*.

The descriptor for each parameter contains the following fields:

Field	Description
oid	Object Identifier for this Parameter
version	Version of the descriptor
name	Parameter name to be displayed in a user interface
data type	Data type (integer, float, string, or array thereof)
data size	Nominal size of the data field
access	Read/write access indicator
precision	Precision to displayed for printed numbers
widget	Graphical display hint for this parameter
constraint	An object specifying the set of permitted values for the parameter

Version

The current version is 2. Permitted versions are 0, 1 and 2. Versions 0 and 1 are identical to version 2, except that widget hints are ignored.

Name

This field provides the parameter name to be displayed in DashBoard. The name does not need to be unique. It may be ignored by some software (e.g. the SNMP agent).

Data Type and Size

Data type indicates the storage type for the parameter ***value***.

Access

This field indicates whether the parameter can be modified. This enables the control software to display an appropriate control for read-only values, or to disallow edits. In OGP, the supported values are:

Access	Value	Description
ACCESS_READWRITE	0x01	Parameter may be modified by the control client
ACCESS_READONLY	0x00	Parameter is read-only, and may not be set by the client

Precision

When used with numbers — this field defines the number of digits following the decimal point displayed for printed numbers. It applies mainly to floating point numbers.

When used with string arrays —this field defines the maximum number of bytes reserved for a single element in the array. If it is 0, no limit is set for each element, and the maximum number of bytes in a parameter value is shared arbitrarily amongst all elements in the array.

Constraint

Constraints allow data to be limited to a certain range or certain values.

Widget Hint

The widget hint specifies the type of graphical control that should be used to display this parameter. To ensure backward compatibility with DashBoard 1.0, widget hints are ignored if the **version** field is less than 2.

Constraints

Constraints are an important part of the parameter descriptor. It specifies a legal range of values which the **value** of the parameter may take. Certain constraints also impact how the parameter is displayed within DashBoard. Certain widgets require specific constraints, while others may behave in different manners depending upon the constraint applied to the parameter. For array parameters, the same constraint applies to each element of the array.

Constraints are specified through a numeric identifier called **ctype.** The supported constraint types are:

Constraint Name	ctype	Param Types	Description
NULL_CONSTRAINT	0	All	Parameter is unconstrained.
RANGE_CONSTRAINT	1	INT16_PARAM INT32_PARAM INT16_ARRAY INT32_ARRAY FLOAT_PARAM FLOAT_ARRAY	Parameter is bounded by a min-max range. Display min-max range may be different from the value range.
CHOICE_CONSTRAINT	2	INT16_PARAM INT16_ARRAY	Parameter must be selected from a set (enumeration) of name-value pairs (up to 255 choices)
EXTENDED_CHOICE	3	INT16_PARAM INT16_ARRAY	Parameter must be selected from a set (enumeration) of name-value pairs (more than 255 choices)
STRING_CHOICE	4	STRING_PARAM STRING_ARRAY	Provides a set of available choices. Parameter may be selected from this set, but arbitrary values are also permitted.
RANGE_STEP_CONSTRAINT	5	INT16_PARAM INT32_PARAM INT16_ARRAY INT32_ARRAY FLOAT_PARAM FLOAT_ARRAY	Parameter is bounded by a min-max range. Step size indicates the amount to increment/decrement the value each time it is changed.
ALARM_TABLE	10	INT16_PARAM INT32_PARAM INT16_ARRAY	Each bit in the parameter is a status flag, so param can display 16 or 32 concurrent named error conditions.

Constraint Name	ctype	Param Types	Description
EXTERNAL_CONSTRAINT	11	All	Indicates that the constraint is encoded in an external object, rather than encoded within the descriptor.

Constraints are normally embedded within the parameter descriptor however; they may also be encoded separately as external objects (which allow longer choice lists, etc.).

A detailed definition of each constraint type, and rules for encoding each constraint, are provided below.

Note: *The constraint is considered to be a **contract** for the parameter. DashBoard will not attempt to set a parameter to a value that violates the constraint. Similarly, the **device** must ensure that the value reported for each parameter complies with the constraint. Behavior of some control software may be unpredictable if the reported value violates the constraint.*

Unconstrained

To leave a parameter unconstrained, use the **NULL_CONTRAINT** constraint. Any parameter which does not have any other constraint applied must specify the **NULL_CONSTRAINT**.

Range Constraints

To constrain a numerical parameter to a specific range of values, the **RANGE_CONSTRAINT** or **RANGE_STEP_CONSTRIANT** must be specified. Both constraint types allow a minimum and maximum parameter value (minValue, maxValue). Additionally, an optional display minimum and maximum value (minDisp, maxDisp) may also be specified. This allows the display range to map to normalized parameter range. The value to be displayed is determined by the following linear mapping:

$$displayed\ value\ = minDisp + \frac{(value - minValue) \times (maxDisp - minDisp)}{(maxValue - minValue)}$$

Note that minDisp and maxDisp must be the same data type as the parameter. For example, to display the value of a 12-bit register (0-4095) as a percentage, set

- (minValue, maxValue) = (0, 4095)
- (minDisp, maxDisp) = (0, 100)

The difference between RANGE_CONSTRAINT and RANGE_STEP_CONSTRAINT is the latter also allows a step size to be specified. The step is specified in the same data type as the parameter and is the minimum change increment on the parameter value (not necessarily the display value).

Note *It is strongly recommended that the range (maxValue – minValue) be evenly divisible by the provided step size. Otherwise, when starting from the minimum, the parameter will use values of minValue + n * stepSize and when starting from the maximum, the parameter will use values of maxValue – n * stepSize.*

Range constraints applied to an array parameter apply to all members of the array.

Choice Constraints

Choice constraints allow a parameter to provide a list of choices. **CHOICE_CONSTRAINT** and **EXTENDED_CHOICE** constraints provide a mechanism to create a set of enumerated values for an **INT16** or **INT32** parameter. This allows integer types to be limited to a specific set of valid

values, as well as providing a mechanism to provide text choices in the DashBoard UI for these parameters.

STRING_CHOICE constraint provides a set of default values which may be populated in a **STRING_PARAM**, however unlike **CHOICE_CONSTRAINT** and **EXTENDED_CHOICE**, it does not limit the user to only these values, any value may be used in the string.

Alarms

Assigning an ALARM_TABLE constraint to an integer parameter tells DashBoard to treat the integer as an array of alarms. When alarms are set, they will impact the overall status reporting of the device.

External Constraints

An ***EXTERNAL_CONSTRAINT*** is used to indicate that the constraint for this parameter is provided in an external object, rather than embedded within the parameter descriptor.

This constraint simply provides a reference to the external object, encoded as shown in the following table.

Parameter Structure Objects

Parameter structure objects, or **structs**, are user-defined structures defined within parameters. They are defined by encoding a struct descriptor within the value object of a parameter. This is done by inserting an array of sub-OID descriptors (param objects) into the value field of a parameter. Structs must have their type set to `STRUCT` or `STRUCT-ARRAY`.

A parameter may inherit the struct descriptor from another parameter through use of a STRUCT constraint which specifies a **templateoid**. The **templateoid** specifies the OID of a parameter whose descriptor will be inherited, thus eliminating the need to define identical struct descriptor for each instance of a struct parameter.

Parameter References

Sub-params within a structure may also be defined as **references** to other parameters. These behave much like C++ or Java variable references. A parameter reference inherits the referenced parameter's type, attributes and constraints.

Menus

How a device is displayed in DashBoard is determined by the menu data provided by the device. DashBoard provides two methods for a device to specify menu layout and structure:

- Default openGear layout
- openGear Layout Markup Language (OGLML)

Default Menu Layout

The default menu layout is designed to make it very simple for devices to display a menu structure. Each menu comprises a name and a list of object identifiers specifying the parameters to be displayed in the menu. Menus are organized into groups, where each group comprises a name and an array of menus.

Menus are divided into menu groups. The default layout displays only 2 groups:

- Group 0: Status (read-only)
- Group 1: Configuration

Below is an example of the default layout:

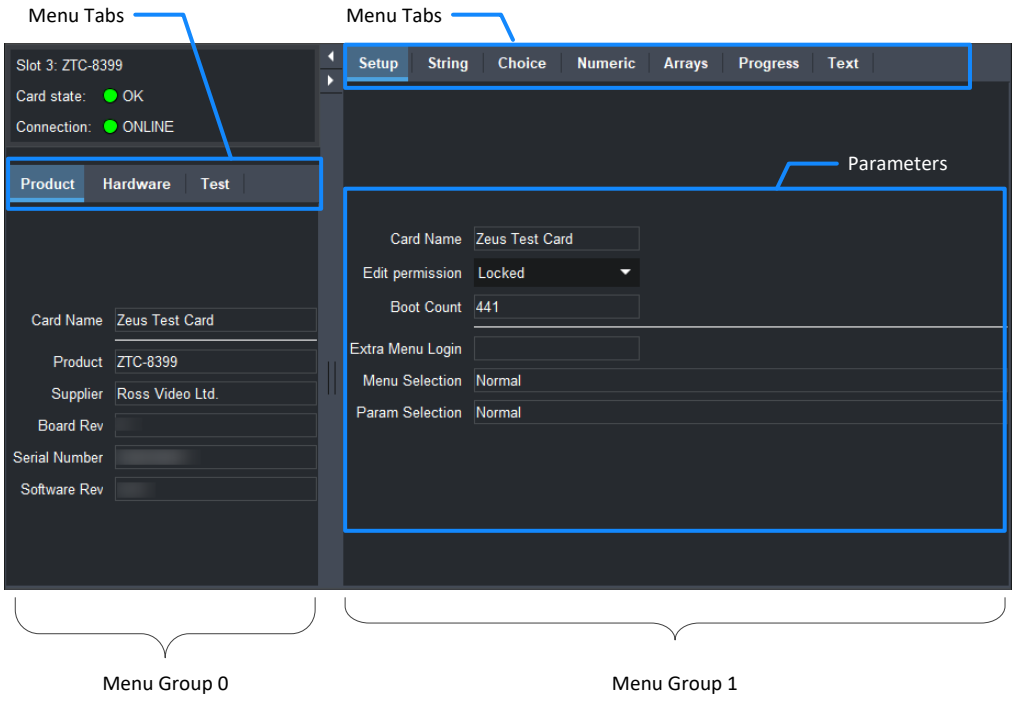


Figure 4 - Menu Layout

Each product may define any number of menus and groups; however, the DashBoard control system recognizes two groups in the default UI layout: group 0 = status parameters (read only), and group 1 = configuration parameters. Other menu groups are not displayed in the default UI layout presented by DashBoard, but may be used in OGLML UI layouts

OGLML Menu Layout

Advanced menu layouts are available with openGear Layout Markup Language (OGLML). OGLML documents can replace an individual menu or the entire device configuration in DashBoard

Customizing Menus Using Display Hints

The descriptor for each parameter includes a widget hint to allow the device designer to specify the type of control to be used to display the parameter. The hints available depend on the parameter type, the constraint type, and the values in the constraint for each parameter. This allows the designer to customize the menu for each device.

DashBoard 1.0 ignored widget hints and provided a default control based on parameter and constraint type. For backwards compatibility, DashBoard 2.0 (and later) ignores widget hints for parameters with the version field set to 0 or 1, providing the same default behavior as DashBoard 1.0. To use widget hints, it is necessary to set the version field within the parameter to 2.

When a read-only parameter provides a widget hint, a read-only version of the parameter's preferred widget is used. The exceptions are WIDGET_DEFAULT (displays like DashBoard 1.0) and Alarm Tables (display the alarm). Hints for status menu parameters are overridden for correct display in that space.

Universal Hints

The following widget hints may be used for any parameter type:

Widget Name	Value	Description
WIDGET_DEFAULT	0	DashBoard will choose what it thinks is the best widget to use for the parameter type and constraint (makes the parameter work like it does with DashBoard 1.0).
WIDGET_TEXT_DISPLAY	1	shows a read-only version of the parameter value (uses same widget that is shown when WIDGET_DEFAULT parameter is set to read-only).
WIDGET_HIDDEN	2	still uses space on the menu page and shows the label for the parameter but show a blank area on the menu page where the widget would be.
WIDGET_LABEL	100	Displays the value of the parameter as a read-only label

Separators, Titles and Layout Hints

The following hints are used with string parameters to provide separators, titles, and extended layout options for menus. Parameters using these widget hints are treated as read only and constant – they do not update live on the screen. Examples of each hint are shown below.

Widget Name	Value	Description
WIDGET_TITLE_LINE	5	displays the value of the String parameter as a label with all other parameter labels and a line across the content area of the menu page.
WIDGET_LINE_ONLY	6	displays a line across the content area of the menu page with no label on the left.
WIDGET_TITLE_ONLY	7	displays the value of the String parameter as a label with empty space in the content area of the menu page.
WIDGET_PAGE_TAB	8	creates a 3rd-level tab within the menu page. The value of the parameter is used as the tab label.
WIDGET_TITLE_HEADER	10	displays a title over the content area of the menu with the value of the parameter used as the header text.

WIDGET_TITLE_LINE (5)

This displays the value of the String parameter as a label aligned with all other parameter labels, and a line across the content area of the menu page. The name of the parameter is ignored.

Figure 5 - WIDGET_TITLE_LINE hint.

WIDGET_LINE_ONLY (6)

This displays a line across the content area of the menu page with no label on the left. The name and value of the parameter are ignored.

Figure 6 - WIDGET_LINE_ONLY hint.

WIDGET_TITLE_ONLY (7)

This displays the value of the String parameter as a label with empty space in the content area of the menu page. The name of the parameter is ignored.

Figure 7 - WIDGET_TITLE_ONLY hint

WIDGET_PAGE_TAB (8)

Whenever a new String parameter with a WIDGET_PAGE_TAB hint is found on a menu page, a new 3rd-level tab is created inside of that menu page. The label on that tab will be the value of the String parameter. All parameters listed after each WIDGET_PAGE_TAB String parameters (until the next such parameter) are placed on a menu page inside of that 3rd-level tab.

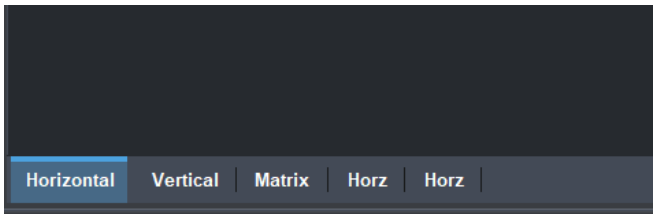


Figure 8 - A menu with WIDGET_PAGE_TAB hints.

Note	*Whenever WIDGET_PAGE_TAB hints are used on a menu, the first OID in the menu should be for a String parameter with a widget hint defining the first tab's label.*

WIDGET_TITLE_HEADER (10)

Displays a title over the content area of the menu with the value of the parameter used as the header text. No label is shown on the left and the name of the parameter is ignored.

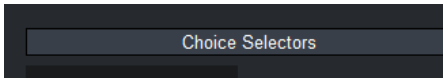


Figure 9 - WIDGET_TITLE_HEADER hint.

Array Layout Hints

By default, all array parameters are displayed horizontally across a menu page. Adjacent OIDs of the same size will format in DashBoard in a tabular format. For example, 3 array parameters with 4 elements each, the layout would appear as:

Array1 Name	Array1[0]	Array1[0]	Array1[0]	Array1[2]
Array2 Name	Array2[0]	Array2[0]	Array2[0]	Array2[2]
Array3 Name	Array3[0]	Array3[0]	Array3[0]	Array3[2]

Figure 10 - Default array layout.

Column headers can be added by adding a read-only INT16_ARRAY parameter to the menu immediately before the other arrays (widget hint WIDGET_ARRAY_HEADER_HORIZONTAL). The parameter is expected to have a choice constraint. The string values of the elements of this parameter provide the column headers. The resulting display is:

Header Name	Header[0]	Header[0]	Header[0]	Header[2]
Array1 Name	Array1[0]	Array1[0]	Array1[0]	Array1[2]
Array2 Name	Array2[0]	Array2[0]	Array2[0]	Array2[2]
Array3 Name	Array3[0]	Array3[0]	Array3[0]	Array3[2]

Figure 11 - WIDGET_ARRAY_HEADER_HORIZONTAL hint.

Array elements can also be given a vertical layout. Changing the widget hint for the header array to `WIDGET_ARRAY_HEADER_VERTICAL` provides the following layout:

Header Name	Array1 Name	Array2 Name	Array3 Name
Header[0]	Array1[0]	Array2[0]	Array3[0]
Header[0]	Array1[0]	Array2[0]	Array3[0]
Header[0]	Array1[0]	Array2[0]	Array3[0]
Header[2]	Array1[2]	Array2[2]	Array3[2]

Figure 12 - WIDGET_ARRAY_HEADER_VERTICAL hint.

Array layout can be specified by including a read-only INT16_ARRAY parameter as a header, with one of the following widget hints:

Widget Name	Value	Description
WIDGET_ARRAY_HEADER_VERTICAL	15	indicates that the associated array parameter and all subsequent parameters should be displayed in a vertical layout
WIDGET_ARRAY_HEADER_HORIZONTAL	16	indicates that the associated array parameter and all subsequent parameters should be displayed in a horizontal layout

Normally sequential array OIDs will be formatted as a single table. If it is desired to break a block of sequential array OIDs into multiple tables, it is necessary to insert a non-array OID, or switch from a horizontal layout hint to a vertical layout hint (or vice versa). If multiple arrays of different size are encoded with different sizes, the layout may be unpredictable.

WIDGET_ARRAY_HEADER_VERTICAL (15)

This hint indicates that the associated array parameter and subsequent parameters should be displayed in a vertical layout. The elements of the parameter will be used as row labels for display. The names of the following arrays are used as column labels. The header should be a read-only INT16_ARRAY parameter with a choice constraint to allow meaningful text labels. The elements of each array are displayed as specified by the widget hint for that array.

The vertical array layout will be applied until another WIDGET_ARRAY_HEADER_VERTICAL starts a new set of vertical columns, a WIDGET_ARRAY_HEADER_HORIZONTAL declares that subsequent arrays should be laid out horizontally, a non-array element in found on the page, or the end of the menu page is reached.

Figure 13 shows an INT16_ARRAY parameter named "Channel", provides a vertical layout and row labels for 7 array parameters named “Channel Update”, “Source”, “Vertical Channel”, “Delay Array (ms)”, “Gain (dB)”, “Invert”, and “Destination”.

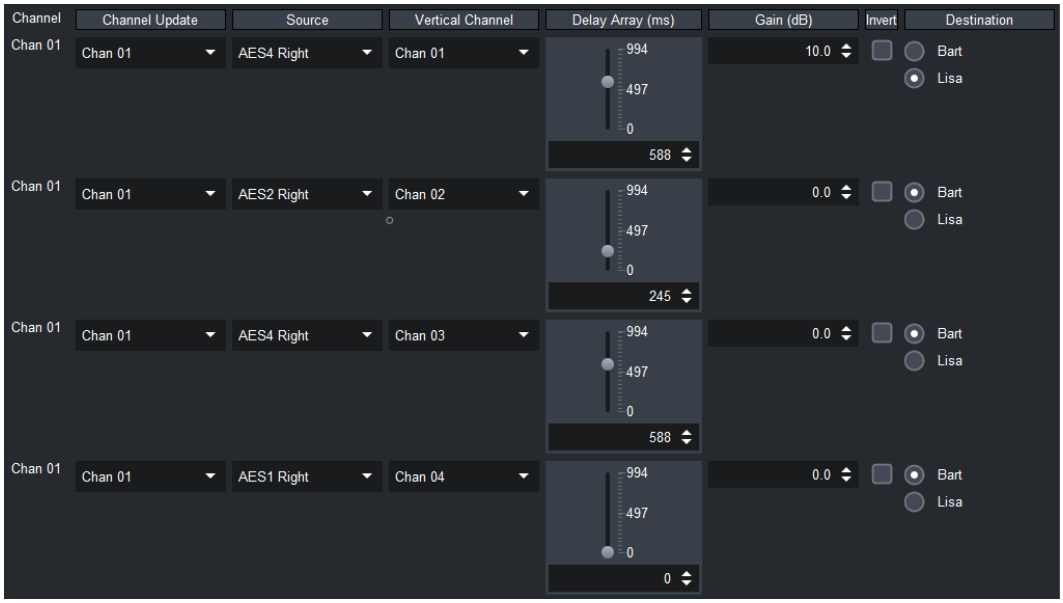


Figure 13 - INT6_ARRAY vertical layout example.

WIDGET_ARRAY_HEADER_HORIZONTAL (16)

The WIDGET_ARRAY_HEADER_HORIZONTAL is used to create a header over a horizontal array. It will also will end a block of vertical array elements. Each element in the header parameter will be displayed as a column header.

Figure 14 shows an INT16_ARRAY parameter named "Channel" providing a horizontal layout and column labels for 7 array parameters named “Horizontal Channel”, “Source”, “Delay Array (ms)”, “Gain (dB)”, “Invert”, “Destination” and “Transition”.

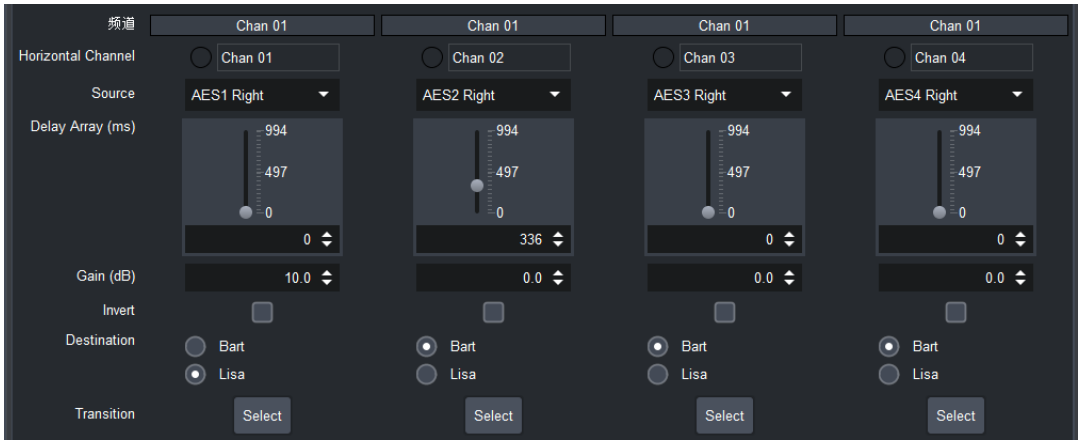


Figure 14 - INT16_ARRAY horizontal layout example.

INT16/INT32 Parameters with Choice Constraints

The following hints apply to INT16, INT16_ARRAY, INT32, and INT32_ARRAY Parameters provided that they use a constraint of type CHOICE or EXTENDED_CHOICE. There are some restrictions for certain hints (checkboxes and toggle buttons are only valid for 2-choice constraints, buttons with and without prompts are only valid for single-choice and 2-choice constraints). If a widget hint is used incorrectly, the combo box will be substituted in place of the chosen widget. Display examples are provided below.

Widget Name	Value	Description
WIDGET_COMBO_BOX	7	Displays a dropdown list of selectable options. This is the default widget used for any choice parameter with more than 1 choice provided.
WIDGET_CHECKBOX	8	Displays a checkbox. Checkboxes only apply to parameters with exactly 2 choices. The first choice is considered false or unchecked; the second choice is considered true or checked.
WIDGET_RADIO_HORIZONTAL	9	Displays a radio button for each integer value option. The radio buttons are placed beside each other horizontally on the page.
WIDGET_RADIO_VERTICAL	10	Displays a radio button for each integer value option. The radio buttons are placed in a vertical column.
WIDGET_BUTTON_PROMPT	11	Provides a button with confirmation prompt. Whenever the button is pressed and confirmed, the parameter value is sent to the device.
WIDGET_BUTTON_NO_PROMPT	12	Provides a button *without* confirmation prompt. Whenever the button is pressed, the parameter value is sent to the device.
WIDGET_BUTTON_TOGGLE	13	Displays a toggle buttons. This hint applies only to parameters with exactly 2 choices. The first choice is

Widget Name	Value	Description
		shown when the button is up (not pressed); the second choice is shown when the button is down (pressed).
WIDGET_FILE_DOWNLOAD	18	Displays a file download widget. This hint requires an external object with an OID matching the value of the parameter.
WIDGET_MENU_POPUP	20	Each value in the parameter must refer to the menu ID of an OGP Menu. The choice corresponding to the parameter value has its name used as the value displayed on a button. When the button is pressed, the menu with an OID corresponding to the parameter value is displayed in a popup menu.
WIDGET_RADIO_TOGGLE_BUTTONS	22	Displays a toggle button for each integer value option. The toggle buttons are placed beside each other horizontally on the page.
WIDGET_TREE	31	Displays a tree control. Tree elements are defined by the elements of the choice constraint. The tree hierarchy is defined by "-" characters at the beginning of the choice. See detailed description below for more information.
WIDGET_TREE_POPUP	32	Displays the tree (same definition as WIDGET_TREE) in a combo box control. See detailed description below for more information.

WIDGET_COMBO_BOX (7)

Display a dropdown list of selectable options. This is the default widget used for any choice parameter with more than 1 choice provided.

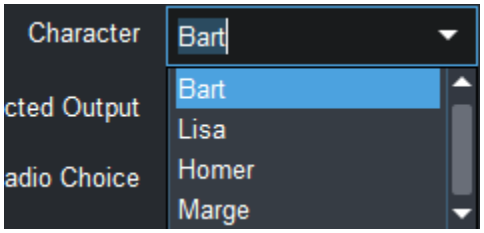


Figure 15 - WIDGET_COMBO hint

WIDGET_CHECKBOX (8)

Displays a checkbox. Checkboxes **only apply to integer choice constraints with exactly 2 choices**. The first choice is considered false or unchecked; the second choice is considered true or checked.

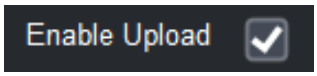


Figure 16 - WIDGET_CHECKBOX hint.

WIDGET_RADIO_HORIZONTAL (9)

Displays a radio button for each element in the choice constraint. The radio buttons are placed beside each other horizontally on the page.

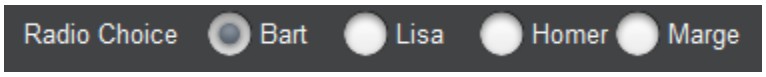


Figure 17 - WIDGET_RADIO_HORIZONTAL hint

WIDGET_RADIO_VERTICAL (10)

Displays a radio button for each element in the choice constraint. The radio buttons are placed in a column vertically on the page.

Figure 18 - WIDGET_RADIO_VERTICAL hint

WIDGET_BUTTON_NO_PROMPT (12)

This hint can only be used for a parameter having a choice constraint with one or two choices. It displays a button with the name of the first choice as the button label. When the button is pressed, a parameter set request is sent to the device immediately (without user confirmation). If the parameter has only one choice, the value of that choice is sent to the device. If the parameter has two choices, the value of the second choice is sent. The device should normally reset the parameter value to the first choice when it acknowledges the set request.

Figure 19 shows a single-choice parameter named "Factory Defaults" with a hint of WIDGET_BUTTON_NO_PROMPT and a value of "Reset". There will be no confirmation dialog.

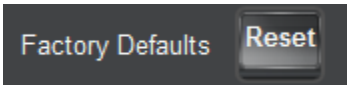


Figure 19 - WIDGET_BUTTON_NO_PROMPT hint.

WIDGET_BUTTON_PROMPT (11)

This hint can only be used for a parameter having a choice constraint with one or two choices. It is the default widget used when **only one choice is available**. It displays a button with the name of the first choice as the button label. When the button is pressed, a confirmation dialog is displayed before sending anything to the device. The dialog uses the format: “[Button Label] [Parameter Name]?” So a choice called “Reset” with a parameter named “Parameter Values” would display “Reset Parameter Values?” as the prompt. When the button is pressed and confirmed, a parameter set request is sent to the device. If the parameter has only one choice, the value of that choice is sent to the device. If the parameter has two choices, the value of the second choice is sent. The device should normally reset the parameter value to the first choice when it acknowledges the set request. If a two-state button is desired, see WIDGET_BUTTON_TOGGLE (13) on page 31.

Figure 20 shows single-choice parameter named "Factory Defaults" with a hint of WIDGET_BUTTON_PROMPT and a value of "Reset".

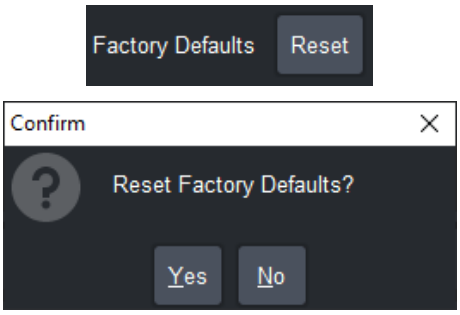


Figure 20 - WIDGET_BUTTON_PROMPT hint

Note	*Two choices are necessary for using **WIDGET_BUTTON_PROMPT** and **WIDGET_BUTTON_NO_PROMPT** with array parameters.*

WIDGET_BUTTON_TOGGLE (13)

Toggle buttons work exactly the same was a checkbox. The toggle button **applies only to integer constraints with exactly two choices**. The name of the first choice is shown when the button is up (not pressed) and the name of the second choice is shown when the button is down (pressed).

Figure 21 shows a two-choice integer parameter named "Bold Toggler" with choice 1 set to "First Value" and choice 2 set to "Second value" The figure shows the button's display for both before and after a button toggle.

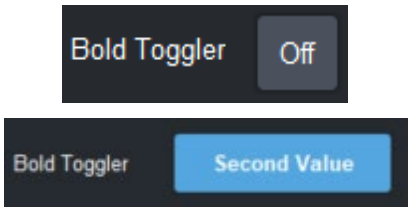


Figure 21 - WIDGET_BUTTON_TOGGLE hint

WIDGET_FILE_DOWNLOAD (18)

This hint requires that an external object with an OID matching the value of the parameter be available. For each choice in the parameter's choice constraint, the choice value represents an external object's OID and the value represents the filename to display. When the 'save' button is pressed, DashBoard requests the external object with the given OID and save the external object's bytes to the filename/location defined by the user (default filename is defined in the choice constraint).

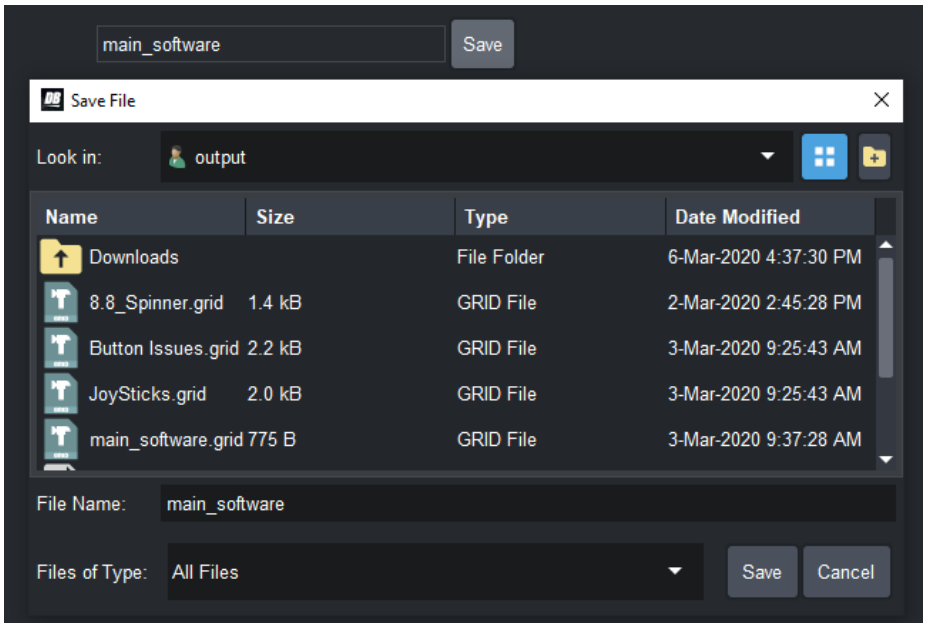


Figure 22 - WIDGET_FILE_DOWNLOAD hint

WIDGET_MENU_POPUP (20)

This hint requires that an OID Menu with a menu ID matching the value of the parameter be available. For each choice in the parameter's choice constraint, the choice value represents a menu's ID and the choice name represents the label to display on the button. When the button is pressed, DashBoard displays the menu with the given ID as a popup menu.

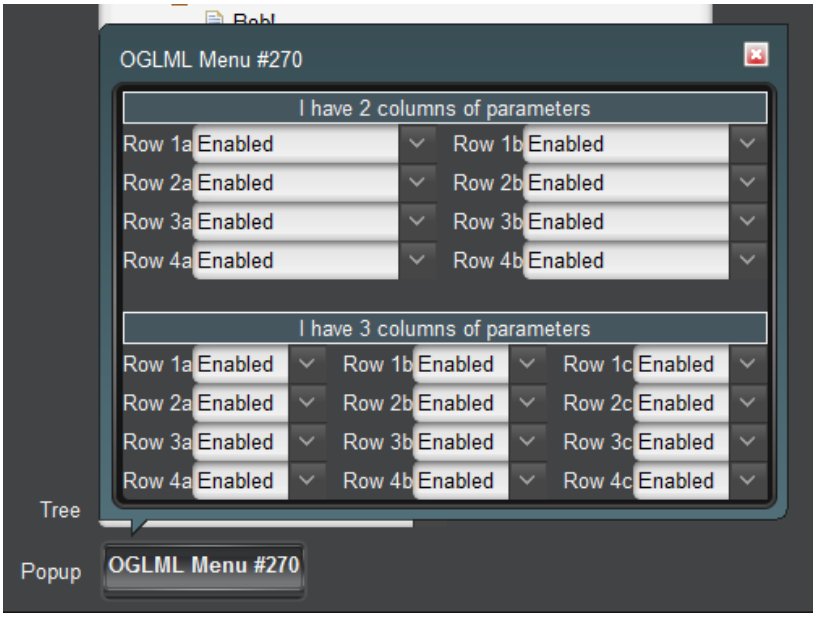


Figure 23 - WIDGET_MENU_POPUP hint

WIDGET_RADIO_TOGGLE_BUTTONS (22)

Displays a radio toggle button for each integer value option. The radio toggle buttons are placed beside each other horizontally on the page.

Figure 24 - WIDGET_RADIO_TOGGLE_BUTTONS hint

WIDGET_TREE (31)

Displays a tree control. Tree elements are defined by the elements of the choice constraint. The tree hierarchy is defined by "-" characters at the beginning of the choice. When an element in the tree is selected, the parameter value is set to the value of the selected choice. All other expand/collapse changes are local only to the DashBoard on which the change occurred.

"+" indicates that an element should be expanded by default.

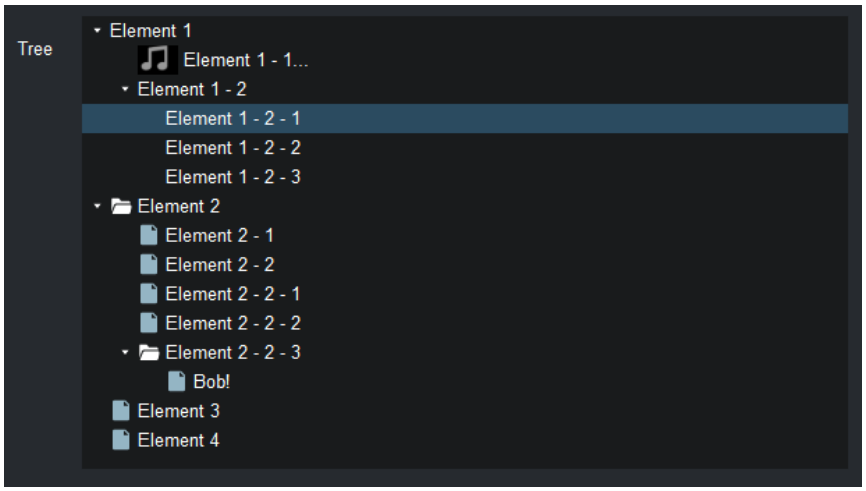


Figure 25 - WIDGET_TREE hint

The tree pictured above is defined by the following list of choices:

1. Element 1<i:>
2. +Element 1 - 1<i-u:http://127.0.0.1/icons/small/sound2.png>
3. +Element 1 - 2<i:>
4. +-Element 1 - 2 - 1<i:>
5. +-Element 1 - 2 - 2<i:>
6. +-Element 1 - 2 - 3<i:>
7. Element 2
8. +-Element 2 - 1
9. +-Element 2 - 2
10. +-Element 2 - 2 - 1
11. +-Element 2 - 2 - 2
12. +-Element 2 - 2 - 3
13. +--Bob!
14. Element 3
15. Element 4

WIDGET_TREE_POPUP (32)

Displays the tree (same definition as WIDGET_TREE) in a combo box control. This functions the same as WIDGET_TREE, with the difference that only the currently selected item shows by default. When the user clicks on the value, a popup appears, allowing selection to be made.

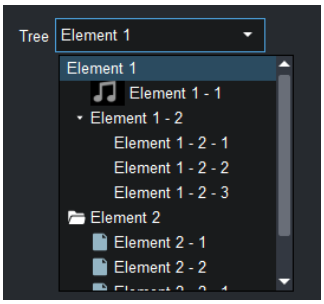


Figure 26 - WIDGET_TREE_POPUP hint

Hints for Numeric Parameters with Other Constraints

The following hints are for INT16, INT16_ARRAY, INT32, INT32_ARRAY, FLOAT, and FLOAT_ARRAY parameters and arrays with constraints other than choices. Most hints have specific restrictions. Details for each hint are provided below.

Widget Name	Value	Description
WIDGET_SLIDER_HORIZONTAL	3	Displays a horizontal slider control. This is the default control for range-bounded integer and floating point parameters when they are not used in an array
WIDGET_SLIDER_VERTICAL	4	Displays a vertical slider control. This is the default control for range-bounded integer and floating point array parameters
WIDGET_SPINNER	5	Displays a spinner (entry field plus up/down arrows). This is the default for unbounded INT16 parameters. This cannot be used for unbounded FLOAT or INT32 parameters.
WIDGET_TEXTBOX	6	Displays a numeric entry field. This is the default for unbounded FLOAT and INT32 parameters.
WIDGET_IP_ADDRESS	14	Displays an IP Address entry field. Only works with unconstrained INT32 parameters.
WIDGET_PROGRESS_BAR	17	Displays a read-only progress bar control.
WIDGET_AUDIO_METER	19	Displays a read-only audio level meter control with green, yellow, and red markers.
WIDGET_TIMER	21	Displays a label that counts down from the parameter value to 0 when double-clicked.
WIDGET_COLOR_CHOOSER	23	Put a colour chooser as an element in the UI. Changes made to the colour chooser are instantly sent to the device. Color values are INT32 values in ARGB format.
WIDGET_SLIDER_HORIZONTAL_NO_LABEL	24	Displays a horizontal slider control with no label
WIDGET_SLIDER_VERTICAL_NO_LABEL	25	Displays a vertical slider control with no label
WIDGET_VERTICAL_FADER	26	Displays a vertical slider that looks like a fader bar

Widget Name	Value	Description
WIDGET_TOUCH_WHEEL	27	Displays a touch wheel control
WIDGET_HEX_SPINNER	28	Displays a spinner (entry field plus up/down arrows). Display the value in Base 16.
WIDGET_ABSOLUTE_POSITIONER	29	Provides a 2-axis absolute positioning element. When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate. When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate. A crosshair in a box can be dragged to the absolute position of the value in 2-D space.
WIDGET_ABSOLUTE_CROSSHAIR	30	Position a value in 2-D space. When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate. When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate. A crosshair that snaps to the center when released makes changes in +/- X, +/- Y relative to the offset from the center.
WIDGET_JOY_STICK	34	Position a value in 2-D space. When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate. When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate. Displays a joystick and modifies the X, Y values as the joystick is dragged north, south, east, and west of the center.
WIDGET_COLOR_CHOOSER_POPUP	33	Display a combo box control showing the 'current' colour. On click, show the colour chooser. If "Live" is togged on, update the parameter value immediately. If "Live" is toggled off, update the parameter value when the popup is closed. Color values are INT32 values in ARGB format.
WIDGET_GRAPH	256	Displays a plot graph of a parameter's value over time.
WIDGET_EQ_GRAPH	46	Displays an EQ Graph that provides a visual representation of how bands effect frequencies across a given range.

WIDGET_SLIDER_HORIZONTAL (3)

Horizontal sliders are the default widgets used for range-bounded integer and floating point numbers when they are not used in an array. Sliders are not available for unbounded (null constraint) parameters.

Figure 27 shows an integer parameter with a range constraint bounded by (0, 200) and a WIDGET_SLIDER_HORIZONTAL hint.

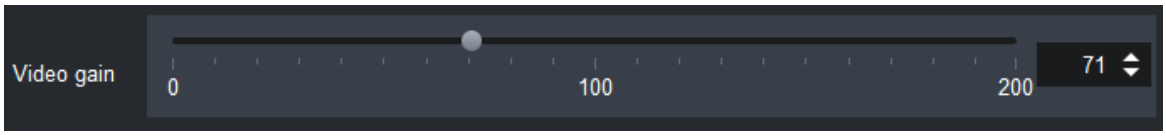


Figure 27 - WIDGET_SLIDER_HORIZONTAL hint

WIDGET_SLIDER_VERTICAL (4)

Vertical sliders are the default widgets used for range-bounded integer and floating point numbers when they are used in an array. Sliders are not available for unbounded (null constraint) parameters.

The following is an integer parameter with a range constraint bounded by (0, 994) and a WIDGET_SLIDER_VERTICAL hint.

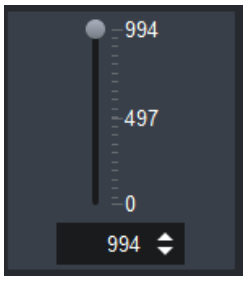


Figure 28 - WIDGET_SLIDER_VERTICAL hint

WIDGET_SLIDER_HORIZONTAL_NO_LABEL (24)

Figure 29 - WIDGET_SLIDER_HORIZONTAL_NO_LABEL hint

WIDGET_SLIDER_VERTICAL_NO_LABEL (25)

Figure 30 - WIDGET_SLIDER_HORIZONTAL_NO_LABEL hint

WIDGET_VERTICAL_FADER (26)

This hint specifies that the number shall be displayed as a vertical fader bar. The user can adjust the level by dragging the handle of the fader up or down.

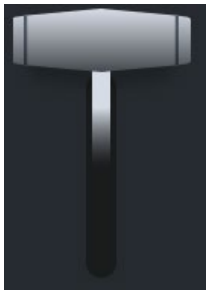


Figure 31 - WIDGET_VERTICAL_FADER hint

WIDGET_TOUCH_WHEEL (27)

This hint specifies that the number shall be displayed as a touch wheel (or circular slider). The user grabs the dot on the circle and drags clockwise to increment the value and counter clockwise to decrement it. The touch wheel can be configured to take a specified number of revolutions to go from the minimum value to the maximum value and can also be configured to roll over to the minimum or maximum when the limits of the range are reached.

Figure 32 - WIDGET_TOUCH_WHEEL hint

WIDGET_PROGRESS_BAR (17)

This hint specifies that the number shall be displayed as a horizontal progress bar. For a range-bounded parameter, the progress bar displays the specified range (similar to a slider). For an unbounded parameter, the progress bar displays from 0 to 100%.

Figure 33 - WIDGET_PROGRESS_BAR hint

WIDGET_SPINNER (5)

Spinner widgets provide a compact way to navigate a bounded integer or float parameter. Spinner widgets are the default widgets used for unbounded int16 parameters**. The spinner widget cannot be used with an unbounded floating point or int32 parameter.**

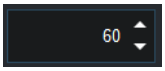


Figure 34 - WIDGET_SPINNER hint

Notes	*The range of the parameter: abs(max – min) x precision cannot exceed the maximum size of a signed integer for sliders and spinners.* *To aid in touch screen environments, clicking and dragging a spinner up/down will increase/decrease its value.*

Properties

Property	Type	Default	Description
w.keyboard	Integer		`disabled` – Disables the soft keyboard or number pad to enter characters when using a touchscreen.

WIDGET_TEXTBOX (6)

This hint specifies that a simple text entry field should be used for a number. The information entered into the text field is forced to conform to the constraints provided by the parameter. This is the default widget used for unbounded floating point parameters.

Figure 35 - WIDGET_TEXTBOX hint

WIDGET_IP_ADDRESS (14)

Displays an IPv4 Address format for a 32-bit integer. Only works with unbounded INT32 parameters.

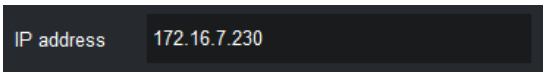


Figure 36 - WIDGET_IP_ADDRESS hint

WIDGET_AUDIO_METER (19)

This hint specifies that the number shall be displayed as a vertical audio meter. The number of red/yellow/green segments is fixed.

Figure 37 - WIDGET_AUDIO_METER hint

WIDGET_TIMER (21)

This hint applies only to integer parameters with RANGE_STEP_CONSTRAINT constraints. The parameter is displayed as a label and counts down if minVal < 0 or up if minVal >= 0. Negative numbers are not displayed. The step size is used to specify the number of ticks-per-second to use and must be a number between 1 and 1000.

When the maximum or minimum values are reached, the timer will stop counting.

To initialize the counter to a specific value but not have it start counting:

- If the minimum value is negative and the parameter value is positive, the timer will display absolute(min) –value but will not count.
- If the minimum value is positive and the parameter value is negative, the timer will display absolute(value) but will not count.

The timer can be reset or synchronized by sending a REPORT_PARAM message with the new parameter value (typically “1”).

Examples:

- min=-600, max=0, step=1 (count from 10:00 to 0:00 showing each second).
- min=0, max=600, step=1 (count from 0:00 to 10:00 showing each second)
- min=0, max=1000,step=1000 (count from 0:00:000 to 0:01:000 showing each millisecond)

Figure 29 shows an INT_32 parameter with a WIDGET_TIMER hint, a precision of 1000, and a value of 13794088.

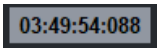


Figure 38 - WIDGET_TIMER hint

WIDGET_HEX_SPINNER (28)

Displays a spinner (entry field plus up/down arrows). Display the value in Base 16.

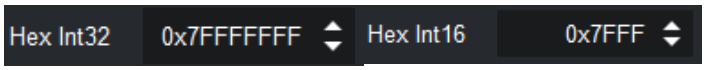


Figure 39 - WIDGET_HEX_SPINNER hint

Notes	*Due to the lack of unsigned data types in OGP, hex spinners do not function properly in the following circumstances:* *- An INT16 parameter with any values in the range of 0x8000 – 0xFFFF* *- An INT32 parameter with any values in the range of 0x80000000 – 0xFFFFFFFF* *- To allow a spinner to function in the range from 0x0000 – 0xFFFF, it is recommended that an INT32 parameter be used.*

WIDGET_ABSOLUTE_POSITIONER (29)

Position a value in 2-D space.

- When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate.
- When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate.

A crosshair in a box is dragged to the absolute position of the value in 2-D space. The ratio of width to height is the ratio of xmax-xmin to ymax-ymin with the assumption that the screen pixels are square. Values are updated and sent to the device as the crosshair is dragged.

Figure 40 - WIDGET_ABSOLUTE_POSITIONER hint

WIDGET_CROSSHAIR (30)

Position a value in 2-D space.

- When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate.
- When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate.

A crosshair that snaps to the center when released makes changes in +/- X, +/- Y relative to the offset from the center. The ratio of width to height is the ratio of xmax-xmin to ymax-ymin. Values are updated and sent to the device as the crosshair is dragged.

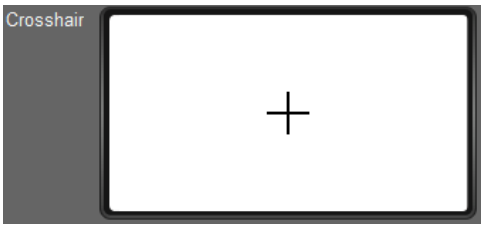


Figure 41 - WIDGET_CROSSHAIR hint

WIDGET_JOY_STICK(34)

Position a value in 2-D space.

- When used as an INT16, the 8 LSBs represent the X coordinate and the 8 MSBs represent the Y coordinate.
- When used as an INT32, the 16 LSBs represent the X coordinate and the 16 MSBs represent the Y coordinate.

Displays a joystick and modifies the X,Y values as the joystick is dragged north, south, east, and west of the center.

Figure 42 - WIDGET_JOY_STICK hint

WIDGET_COLOR_CHOOSER(23)

Display a color chooser as an element in the UI. Changes made to the color chooser are immediately sent to the device. Note that the color chooser provides control for Hue, Saturation, Lightness, but color values are INT32 values in ARGB format.

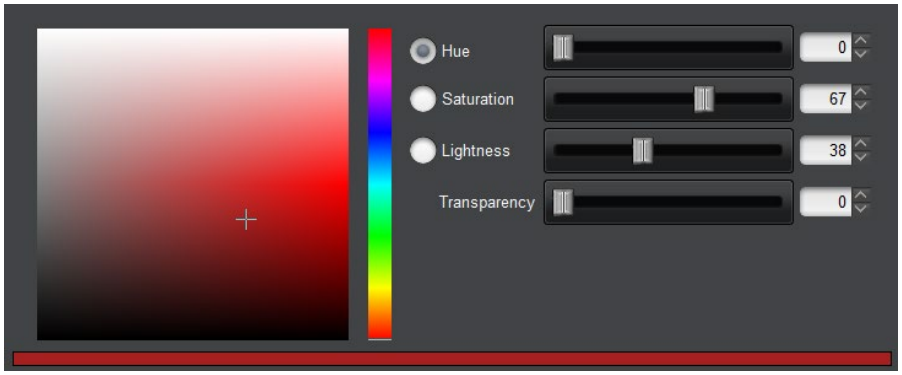


Figure 43 - WIDGET_COLOR_CHOOSER hint

WIDGET_COLOR_CHOOSER_POPUP(33)

Display a combo box control showing the 'current' color. On click, show the color chooser.

- If "Live" is togged on, update the parameter value immediately.
- If "Live" is toggled off, update the parameter value when the popup is closed.

Color values are INT32 values in ARGB format.

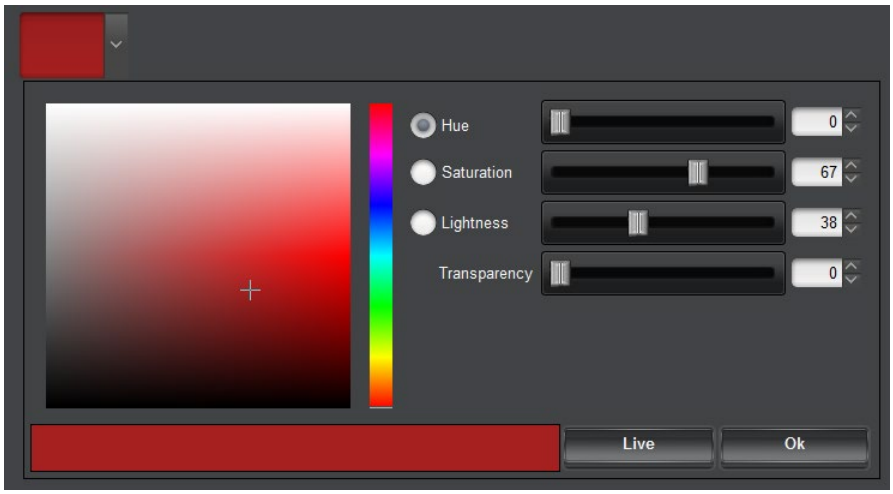


Figure 44 - WIDGET_COLOR_CHOOSER hint

WIDGET_GRAPH (256)

The graph widget provides a plot graph which tracks the value of a numeric parameter over time.

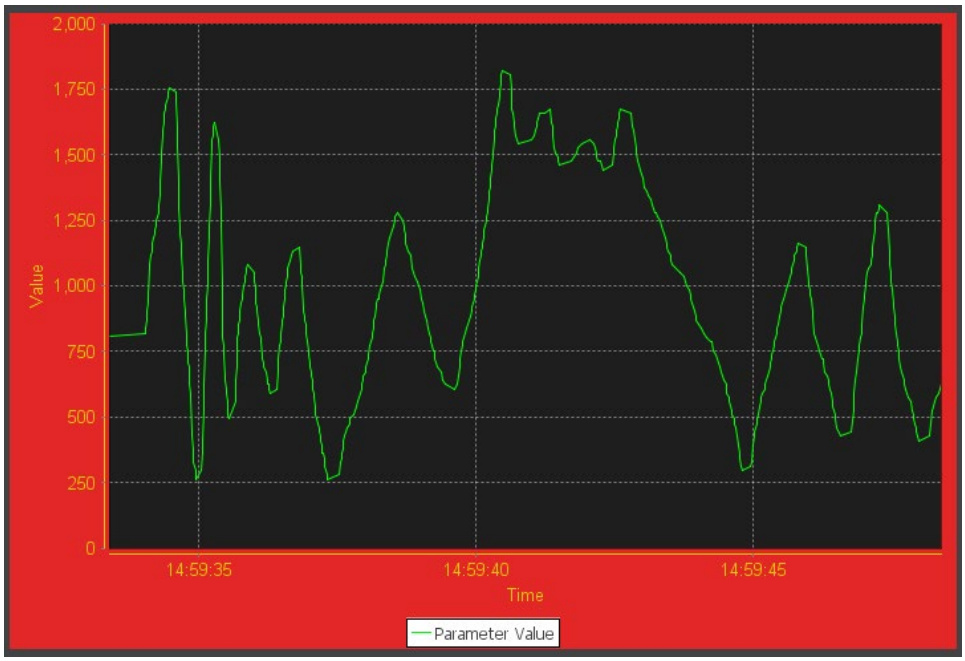


Figure 45 - WIDGET_GRAPH hint

A parameter utilizing a WIDGET_GRAPH widget may also specify additional configuration parameters in the config object of the parameter.

Properties

Property	Type	Default	Description
w.time	Integer		Sets the timescale of the plot. If set to 0, the timescale will adapt to display entire change history.
w.grid	String		Sets the color of the gridlines
w.plotfg	String		Sets the color of the plot foreground
w.plotbg	String		Sets the color of the plot background
w.hidelegend	Boolean		`true` – Legend is not shown `false` – Legend is shown
w.hidex	Boolean		`true` – X-axis scale is not shown `false` – X-axis scale is shown
w.hidey	Boolean		`true` – Y-axis scale is not shown `false` – Y-axis scale is shown
w.autoadvance	Booelan		`true` – graph will auto-update every 1 second `false` – graph will only update upon parameter change.

WIDGET_EQ_GRAPH (46)

The EQ graph widget provides a a visual representation of how bands effect frequencies across a given range. This advanced widget allows you to make an EQ graph, using parameters from any device that talks to DashBoard. The EQ graph creates a graphical representation of parametric equalization. For example, you can add a Ross Video Carbonite switcher to DashBoard as a device, and then measure bands from the Carbonite's parameters. The graphic below shows an EQ graph that is pulling parameters from a Carbonite switcher, and the equalizer settings have been mapped to slider controls to make adjustments from the DashBoard CustomPanel.

Each band has an associated frequency, range, and Q value, if required.

The filter that each band is applying can be specified in the configuration overrides. If the filter is not defined, then it will default to a peak filter.

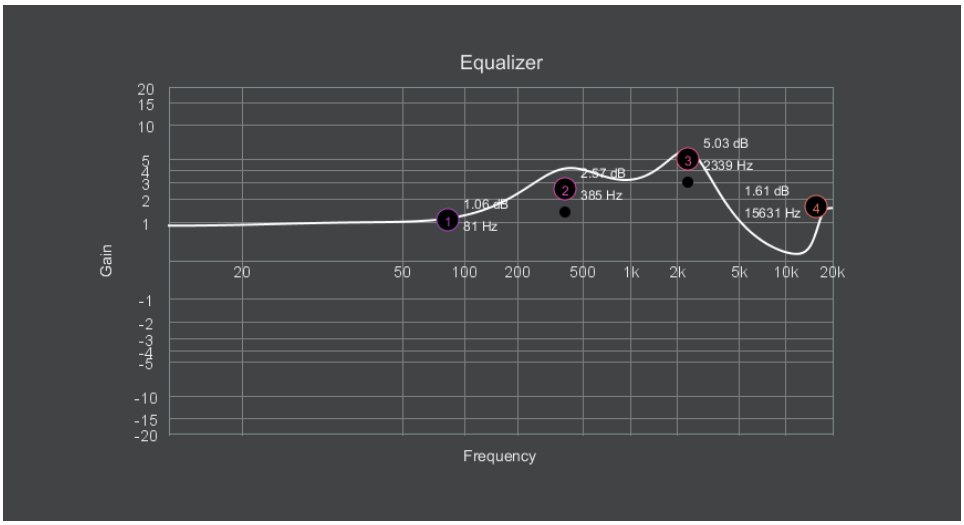


Figure 46 - WIDGET_EQ_GRAPH hint

A parameter utilizing a WIDGET_EQ_GRAPH widget must also specify additional configuration parameters in the config object of the parameter. For more details see the **DashBoard User Guide**.

Properties

Property	Type	Default	Description
w.linecolor	#[RGB Value]		Sets the color of the graph point to point line.
w.filters	[String Array]		Sets the filter for each band, where the possible values are lowshelf, peak or highshelf. One filter per point.
w. pointnames	[String Array]		Sets the name on each graph point, for example:
w.colorselected	#[RGB Value Array]		1, 2, 3, 4
w.colorunselected	#[RGB Value Array]		Sets the color for each point when it's selected. For example:
w.linethickness	[Integer]		#ffd966,#c27ba0,#6d9eeb,#93c47d
w.graphfontsize	[Integer]		Sets the color for each point when it's not selected. For example:

Property	Type	Default	Description
w.pointfontsize	[Integer]		#ffd966,#c27ba0,#6d9eeb,#93c47d
w.pointwidth	[Integer]		Sets the thickness of the graph point to point line.
w.pointheight	[Integer]		Sets the font size of the text used on the graph title, axis labels and axis entries.
w.xaxis	[String]		Sets the font size for the point names.
w.yaxis	[String]		Sets the width of the points.
w.xaxisentries	[String Array]		Sets the height of the points.
w.yaxisentries	[String Array]		Sets the x axis label.
w.pointamount	[Integer]		Sets the y axis label.
w.frequencyshift	[Integer]		Sets the line marks on the x axis. For example: 20,50,100,200,500,1000,2000,5000,10000
w.title	[String]		Sets the line marks on the y axis. For example:

Hints for String Parameters

The following widget hints may be used for String parameters (in addition to the separators and layout hints defined above). The last two hints apply only to a String parameter using the reserved objectID 0xFF01.

Widget Name	Value	Description
WIDGET_TEXT_ENTRY	3	Displays a normal text entry field. This is the default for editable String parameters.
WIDGET_PASSWORD	4	Displays an entry field for passwords (text entered in this field is obscured).
WIDGET_COMBO_ENTRY	11	Displays an entry field together with a dropdown list of selectable items. This is applicable only with the STRING_CHOICE constraint.
WIDGET_COLORED_DOT	12	Displays a colored icon. The icon color is specified using a tag in the text string.
WIDGET_RICH_LABEL	13	Displays a read-only multi-line text field with HTML formatting.
WIDGET_MULTILINE_TEXT_ENTRY	14	Displays a multi-line text editor.
WIDGET_NAME_OVERRIDE_APPEND	0	Special hint only for objectID 0xFF01 – causes this string to be appended to the displayed product name
WIDGET_NAME_OVERRIDE_REPLACE	1	Special hint only for objectID 0xFF01 – causes this string to replace the product name to be displayed

WIDGET_TEXT_ENTRY (3)

This is a text entry field used to enter String values. This is the default widget used with editable String parameters. It is very important to correctly set the length of the String with this widget as the length affects the width of the text field. In DashBoard the value is sent to the device when the user hits 'Enter' or changes focus to a different control on the screen.

Figure 47 - WIDGET_TEXT_ENTRY hint

Properties

Property	Type	Default	Description
w.keyboard	Integer		`disabled` – Disables the soft keyboard or number pad to enter characters when using a touchscreen.

WIDGET_PASSWORD (4)

This is a text entry field used to enter passwords. When the device receives a set message for a parameter using this hint, a device could send an empty string back to the device to clear the password field. Text in the password field is sent when it has changed from the value reported from the device and the user hits "Enter" or moves focus to another control.

Figure 48 - WIDGET_PASSWORD hint

WIDGET_COMBO_ENTRY (11)

This displays a text entry field along with a dropdown list. This option is available only for String parameters having a STRING_CHOICE constraint. The user may select an option from the dropdown list, or can type any value in the entry field. The text is sent to the device when a dropdown item is selected, when the user presses “Enter” or moves the focus after typing a value.

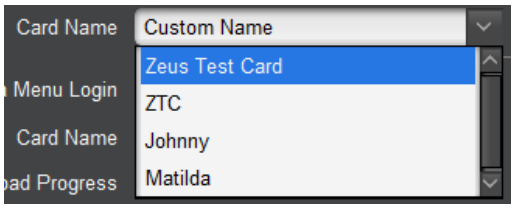


Figure 49 - WIDGET_COMBO_ENTRY hint, selecting from the dropdown list

WIDGET_COLORED_DOT (12)

This displays a colored icon. This should not be confused with Alarm parameters which have a similar appearance. The tag specifies the 24-bit RGB color index of the icon in hex, in the format <#RRGGBB>. If the string does not contain a valid color tag, the icon is drawn but not filled (i.e. background shows through).

Figure 50 - WIDGET_ICON_DISPLAY hint, and value “<#3F3FFF>”

WIDGET_RICH_LABEL (13)

This displays a read-only multi-line text field, and formats the text according to the HTML formatting tags embedded in the text. Total string length including tags is limited to 250 bytes. The display uses html support within the java display object, so the exact appearance of the label may vary depending on operating system and java version.

Figure 51 - WIDGET_RICH_LABEL hint

WIDGET_MULTILINE_TEXT_ENTRY (14)

This displays a multi-line text entry field. The amount of data a user can input into the field is limited by the maximum length specified by the parameter. The size of the field is the same regardless of the maximum number of bytes the user is allowed to enter. If the parameter’s value spans more lines than the number of rows represented by the text field, a vertical scrollbar is shown to allow the user to scroll. Text will be wrapped to avoid horizontal scrollbars.

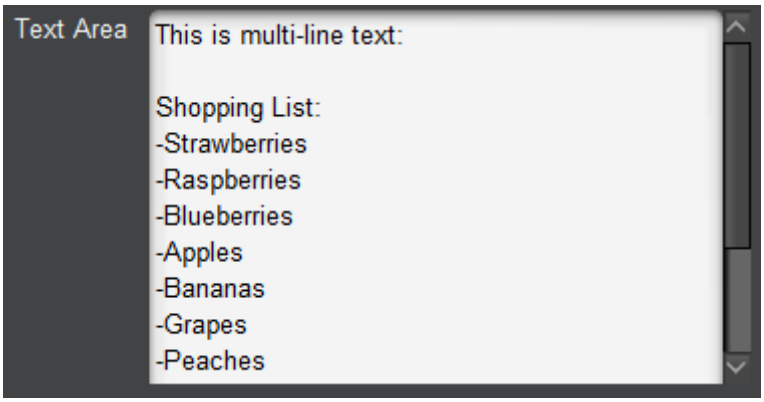


Figure 52 - WIDGET_MULTILINE_TEXT_ENTRY hint

WIDGET_NAME_OVERRIDE_APPEND (0)

This is a special hint ONLY FOR OID 255.1 (0xFF01). This causes the value of the String parameter with OID 255.1 to be appended to the end of the device name in DashBoard.

Figure 53 shows the result of setting parameter 255.1 to " (XPF)" with a **WIDGET_NAME_OVERRIDE_APPEND** hint.

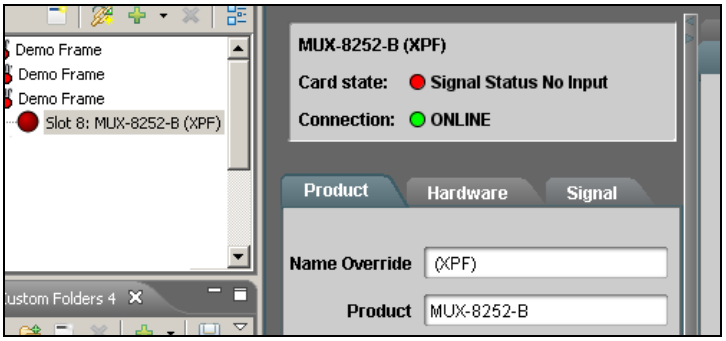


Figure 53 - WIDGET_NAME_OVERRIDE_APPEND hint

WIDGET_NAME_OVERRIDE_REPLACE (1)

This is a special hint ONLY FOR OID 255.1 (0xFF01). This causes the value of the String parameter with OID 255.1 to be displayed as the device name instead of the product name (OID 0x0105) in DashBoard. This is the only supported method for changing a product name dynamically. Devices should never modify their base product name (OID 0x0105); DashBoard, DataSafe, and User Rights all depend on the base product name remaining fixed. Change of the product name is assumed to mean that the user has physically removed a card, and has replaced it with a different type of card.

Figure 54 shows the result of setting parameter 255.1 to "My Device Name" with a **WIDGET_NAME_OVERRIDE_REPLACE** hint.

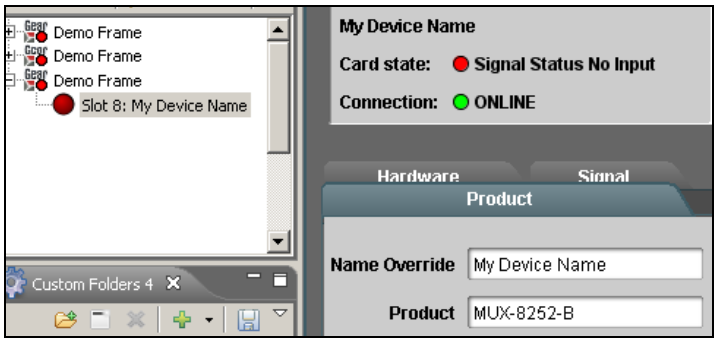


Figure 54 - WIDGET_NAME_OVERRIDE_REPLACE hint

Hints for STRUCT Types

Struct parameters may utilize the following widget types:

WIDGET_TABLE (36)

The table widget displays a line for each element in a STRUCT_ARRAY. Column headings are specified by the **name** property of each struct element. Each element of the struct is given a column in the table.

Clip Name	Director	Original Air Date	Author
Winter is Coming	Tim Van Patten	April 24, 2011	David Benoiff & D.B. Weiss
The Kingsroad	Brian Kirk	April 24, 2011	David Benoiff & D.B. Weiss
Lord Snow	Brian Kirk	May 1, 2011	David Benoiff & D.B. Weiss
A Golden Crown	Daniel Minahan	May 22, 2011	David Benioff & D. B. Weiss

Figure 55 - WIDGET_TABLE hint

A parameter using a WIDGET_TABLE widget may also specify additional configuration parameters in the config object of the parameter.

Properties

Property	Type	Default	Description
w.localselection	Boolean	`false`	`true` – edits in the table row do not update backing parameter; changes in the backing parameter do not update the selected row(s). `false` – backing parameter and table row changes track with each other.
w.scrollselection	Boolean	`true`	`true` – auto scroll to the selected row `false` – do not scroll to the selected row
w.reorder	Boolean	`false`	`true` – allow drag to reorder values `false` – do not allow drag to reorder values
w.rowstyleparam	String	*none*	OID of string array parameter providing style information (background, foreground, font, font size, etc.) for each row.
w.selectionparam	String	*none*	OID of integer parameter that will be populated with the index of a selected row.
w.rowaccessparam	String	*none*	OID of integer array parameter which determines read-only access for each row. (0 = read-only, 1 = read-write). If not specified, all rows are read-write.
w.rowheight	Number	*automatic*	Sets the row height. Specified in pixels
w.colwidth.*n*	Number	*automatic*	Sets the width of the n^{th} column. First column index is 0.
w.colminwidth.*n*	Number	*automatic*	Sets the minimum width of the n^{th} column. First column index is 0.
w.hscroll	Boolean	`false`	`true` – show horizontal scrollbar `false` – do not show horizontal scrollbar
w.alwaysscroll	Boolean	`false`	`true` – vertical scrollbar always shown `false` – vertical scrollbar only shown only when required
w.hgrid	Boolean	`true`	`true` – display horizontal grid lines `false` – do not display horizontal grid lines
w.vgrid	Boolean	`true`	`true` – display vertical grid lines `false` – do not display vertical grid lines

Data Types

OGP supports a number of parameter data types as summarized in the table below. For OGP messaging, the **OGP Type** value is a numerical index to indicate the parameter's data type. For JSON messaging, the **Data Type Name** is used to indicate the parameter type.

Data Type Name	OGP type	Data Size (bytes)	Description
INT16	2	2	16-bit signed integer (INT16)
INT32	4	4	32-bit signed integer (INT32)
FLOAT32	6	4	32-bit IEEE single-precision floating point number
STRING	7	variable	null-terminated UTF-8 string data_size = maximum permitted number of character data bytes
INT16_ARRAY	12	2 * len	array of 16-bit integers data_size = 2 * number of elements (total length of the array in bytes)
INT32_ARRAY	14	4 * len	array of 32-bit integers data_size = 4 * number of elements (total length of the array in bytes)
FLOAT32_ARRAY	16	4 * len	array of 32-bit floats data_size = 4 * number of elements (total length of the array in bytes)
STRING_ARRAY	17	variable	null-terminated UTF-8 strings precision = maximum string length for any element in the array data_size = maximum number of character data bytes
STRUCT	n/a	variable	User-defined data structure. (DashBoard 7.0+)
STRUCT_ARRAY	n/a	variable	Array of User-defined data structures. (DashBoard 7.0+)
BINARY_PARAM	18	variable	array of binary data of type unknown to DashBoard.

Endianness

All numeric data encoding within OGP is in Big Endian format. Therefore, highest order bytes of multi-byte numeric values are transmitted first.

Number Encoding

Signed integer data types are binary encoded 2's complement numbers. Valid ranges for integer types are:

Response	Min	Max
UINT8	0	255
UINT16	0	65535
INT16	-32,768	32,767
INT32	-2,147,483,648	2,147,483,647

Floating point data types are encoded as 32-bit IEEE (single-precision) floating point numbers. This encoding is broken down as:

- Sign: 1 bit
- Exponent: 8 bits; Range -126 to +127
- Base: 23 bits
- Data size is the number of bytes occupied by the value.

String Encoding

All string data encoding within OGP is in UTF-8 format. Strings are preceded by a length count byte, and are followed by a null terminating byte.

External Data Objects

To support more complex interaction with the device than is possible with parameters, DashBoard includes a set of general data objects called External Objects. Each object is identified by a 2-byte objectID (like parameters), and contains a type identifier and object-specific data. External object OIDs can overlap with parameter OIDs. The range of OIDs from 0xFE00 to 0xFFFF is reserved for future use.

External objects include an object type to indicate the type of data they encapsulate. The supported object types are:

objtype	Description
1	Constraint
2	File
3	Image
4	OGLML or XML document

Constraint

Parameter constraint information can be taken outside of the parameter descriptor and moved into an external object. This is useful, for example if there is a choice constraint with a large number of options, or a common constraint is to be applied to multiple parameters. The constraint field in the parameter descriptor simply refers to an external object ID.

Any constraint type can be externalized except the external constraint type itself. An external constraint object can be shared by multiple parameters (the external object will be requested only once for all parameters which share the constraint). The object type of the external constraint must be 1, and the object data must be encoded in the same format as used for an embedded constraint.

An external object that is not object type 0x0001 will be treated as a NULL constraint (unconstrained). Just like constraints declared in the parameter descriptor, external constraints must have the same data type as the referring parameter.

Arbitrary File

Arbitrary binary data can be sent from the device to DashBoard as a file download. These files are requested by supplying an integer parameter with a WIDGET_FILE_DOWNLOAD widget hint and a choice constraint. The numeric value of the parameter must match the OID of an external object containing the file data to download. The string value of the choice constraint is used to

supply a file name for the download. To upload the file data back to the device, the data must use the standard openGear file header information defined in the section

Image

Images may be encapsulated within an External Object to be displayed in the device editor (via OGLML) or to be used to override its icon in DashBoard. The icon may include a status indicator or DashBoard can overlay a status indicator over the provided icon. Icons can be provided either by a URL or embedded directly in the external object.

Images must be formatted as JPEG, GIF, or PNG. Icons must be 16x16.

OGLML Descriptor or Index XML

DashBoard includes powerful feature for defining the on-screen layout of a device's configuration page in DashBoard. These configurations are defined in an OGLML Document. These documents can be retrieved from a web server or sent to DashBoard in an external object.

OGLML Documents

This section includes the following topics:

- Containers
- Contexts
- OGLML Document Structure
- OGLML URLs
- OGLML Descriptor Format

Containers

All UI elements must be placed within a container. The container dictates how UI components are laid out within the DashBoard UI. There are several container types which provide different options on component layout. Layout containers may be nested.

By default, PanelBuilder will create a top-level **abs** container, and all elements (including nested containers) are placed within this top-level container.

Contexts

Contexts define scope within an OGLML document. PanelBuilder creates OGLML documents with a default context named "`opengear`". If multiple devices are linked to an OGLML document, each device has its own separate context. Therefore, elements defined within the context of one device are not visible within another device's context.

OGLML Document Structure

The basic structure of the OGLML document is as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<abs contexttype="opengear">
```

```
<api>
    Global code
</api>
<meta>
    Non-UI Tags here
    <api>
        Global ogScript code
    </api>
    <params>
        Parameter declarations
    </params>
    <menus>
        Menu declarations
    </menus>
    <widgets>
        Widget descriptors
    </widgets>
</meta>
<ui container>
    <ui elements/>
    <ui element>
        Local scope ogScript code
    </ui element>
    <nested ui container>
        <ui element/>
        . . .
    </nested ui containter>
    . . .
</ui container>
. . .
</abs>
```

Details about the individual tags are documented in the section OGLML Reference.

OGLML URLs

An OGLML URL can be a standard URL or an external object reference. The fragment ("#name") of the URL can optionally provide the ID of a child element inside of the OGLML document to reference. An external object reference URL has the form "eo://0x1234" where 0x1234 is the external object ID of the external object containing the OGLML descriptor.

OGLML URL examples:

- http://myhost/mydocument.xml
  (include the entire document at the given URL)
- https://10.0.100.1/document.xml#myid
  (include the element with id="myid" from the given URL)
- eo://0xAB12
  (load the OGLML descriptor from the external object with ID 0xAB12)

- eo://0xAB12#my-other-id
  (load the OGLML descriptor from the external object with ID 0xAB12 and select sub-element with id="my-other-id")

OGLML Descriptor Format

The first byte of an OGLML Descriptor defines the type of information to follow.

No OGLML Descriptor (0x00)

This is used when there is no OGLML document referenced by this descriptor.

Field	length	Description
desctype	1 (uint8)	0x00
content	0	No content is provided in this case

Descriptor provided by external object (0x01)

This is used when the descriptor is contained in an external object.

Field	length	Description
desctype	1 (uint8)	0x01
content	2 (uint16)	The external object ID of the object containing the OGLML Descriptor

OGLML Document provided by URL (0x02)

The OGLML document is hosted on a web server. The descriptor provides the URL of the OGLML Document.

Field	length	Description
desctype	1 (uint8)	0x02
urllen	1 (uint8)	The length of the URL to follow including the null terminator
url	urllen	The null-terminated URL of the external object. This must begin with "http://" or "https://". The content on the webserver can be uncompressed or follow web conventions for zip or deflate compression. file:// URLs may also be used but this should generally only be for development purposes and not actually on a released device.

Descriptor provides the OGLML Document in-line (0x03)

The OGLML File document immediately follows the descriptor type field.

Field	length	Description
desctype	1 (uint8)	0x03
content	*	OGLML XML File Content

Descriptor provides a GZipped OGLML Document in-line (0x04)

The OGLML document is provided immediately following the descriptor type field (document is compressed in GZip format).

Field	length	Description
desctype	1 (uint8)	0x04
content	*	GZipped OGLML XML File Content

Descriptor provides a Deflate OGLML Document in-line (0x05)

The OGLML document is provided immediately following the descriptor type field (document is compressed in Deflate format).

Field	length	Description
desctype	1 (uint8)	0x05
content	*	Deflate OGLML XML File Content

Custom Widgets

Custom widgets are user-defined controls within a DashBoard editor. These allow device designers and CustomPanel developers to reuse repeated elements within an OGLML document. Complex UI behaviour can be coded into the widget, which is hidden from the UI developer.

Custom Widgets allow the designer to design an element consisting of multiple controls, OGLML markup tags and ogScript. This element can then be instantiated multiple times within an OGLML document. Widgets may be defined within an OGLML document or made globally available in DashBoard.

Widgets allow configuration parameters exposed to tailor the look, feel and behaviour. These configuration parameters are also available through the PanelBuilder GUI, allowing simple customization of the widget.

Widgets are defined by creating a Widget Descriptor, which consists of a section OGLML/ogScript code that defines the controls. Additionally, a configuration block may be defined which creates a configuration page for the widget within PanelBuilder.

Creating Widgets

Widget Descriptor Structure

The widget descriptor has a structure as outlined below:

```
<widgetdescriptor id="widget-id">
   <config>
      <params>
         Configuration parameters here
      </params>
      <oglml>
         Optional OGLML markup for configuration editor
      </oglml>
   </config>
   <oglml>
      <meta>
         <params>
            Private parameter declarations
         </params>
         <api>
            Private ogScript functions
         </api>
```

```
        <meta>
        <layout-container>
            UI elements
        </layout-container>
    </oglml>
</widgetdescriptor>
```

OGLML Block

The OGLML section (encapsulated within an `<oglml>` tag) contains the OGLML document to create the widget. It may contain `<meta>`, `<ogscript>`, `<api>` and layout container tags in the same manner as a standard OGLML document. Note that all declarations within the `<oglml>` section are private to the widget.

Config Block

The config section (encapsulated within a `<config>` tag) contains OGLML document that creates a configuration page for the widget. The configuration page is displayed within the Edit Component dialog in PanelBuilder. By default, the default openGear layout will be used to present any parameters declared within a `<params>` tag in the config block:

```
<widgetdescriptor id="alarmgrid">
    <config>
        <params>
            <param access="1" name="String 1" oid="str1" type="STRING"
value="First"/>
            <param access="1" name="String 2" oid="str2" type="STRING"
value="Second"/>
            <param access="1" name="String 3" oid="str3" type="STRING"
value="Third"/>
            <param access="1" name="String 4" oid="str4" type="STRING"
value="Fourth"/>
            <param access="1" name="String 5" oid="str5" type="STRING"
value="Fifth"/>
            <param access="1" name="String 6" oid="str6" type="STRING"
value="Sixth"/>
        </params>
    </config>
    <oglml>
        <simplegrid cols="3" rows="2">
            <param height="40" oid="str1" widget="12" width="200"/>
            <param height="40" oid="str2" widget="12" width="200"/>
            <param height="40" oid="str3" widget="12" width="200"/>
            <param height="40" oid="str4" widget="12" width="200"/>
            <param height="40" oid="str5" widget="12" width="200"/>
            <param height="40" oid="str6" widget="12" width="200"/>
        </simplegrid>
    </oglml>
</widgetdescriptor>
```

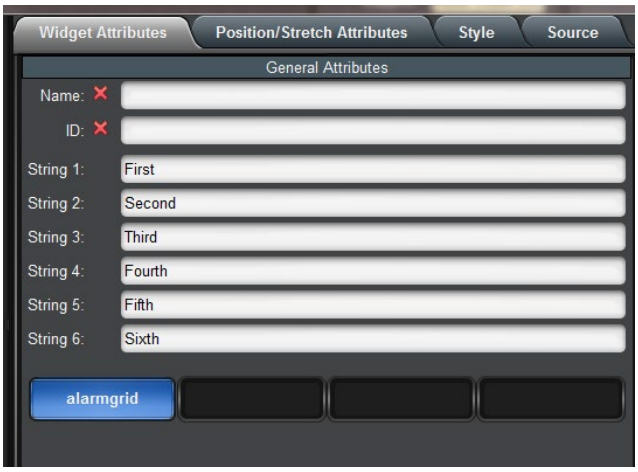


Figure 56 – Widget Configuration (Default Layout)

However, OGLML markup may be added by specifying it within an `<oglml>` block within the config block.

Note	*If an `<oglml>` block is specified within the `<config>` section, only parameters included in the `<oglml>` block will be displayed in the PanelBuilder "Edit Component" dialog.*

The following is an example of a widget descriptor incorporating an OGLML configuration markup. In the example, a `<simplegrid>` container is used to arrange configuration parameters into a 3x2 grid.

```
<widgetdescriptor id="alarmgrid-oglml">
   <config>
      <params>
         <param access="1" name="String 1" oid="str1" type="STRING"
value="First"/>
         <param access="1" name="String 2" oid="str2" type="STRING"
value="Second"/>
         <param access="1" name="String 3" oid="str3" type="STRING"
value="Third"/>
         <param access="1" name="String 4" oid="str4" type="STRING"
value="Fourth"/>
         <param access="1" name="String 5" oid="str5" type="STRING"
value="Fifth"/>
         <param access="1" name="String 6" oid="str6" type="STRING"
value="Sixth"/>
      </params>
      <oglml>
         <simplegrid cols="3" rows="2">
            <param height="40" oid="str1" widget="12" width="200"/>
            <param height="40" oid="str2" widget="12" width="200"/>
            <param height="40" oid="str3" widget="12" width="200"/>
            <param height="40" oid="str4" widget="12" width="200"/>
            <param height="40" oid="str5" widget="12" width="200"/>
            <param height="40" oid="str6" widget="12" width="200"/>
         </simplegrid>
      </oglml>
```

```
</widgetdescriptor>
```

Figure 57 – Widget Configuration (OGLML layout)

If the widget descriptor includes a **structtype** attribute, PanelBuilder will use this as a filter to only offer the widget for insertion if a struct parameter exists with matching **structtype** attribute.

Widget Samples

Numeric Keypad

This example creates a reusable control which presents a numeric keypad. The keypad accepts parameters to map it to a specific OID to update, as well as name and a default value.

Figure 58 - Keypad Custom Widget

The widget also defines a custom configuration panel, which is presented within PanelBuilder's "Edit Component" dialog.

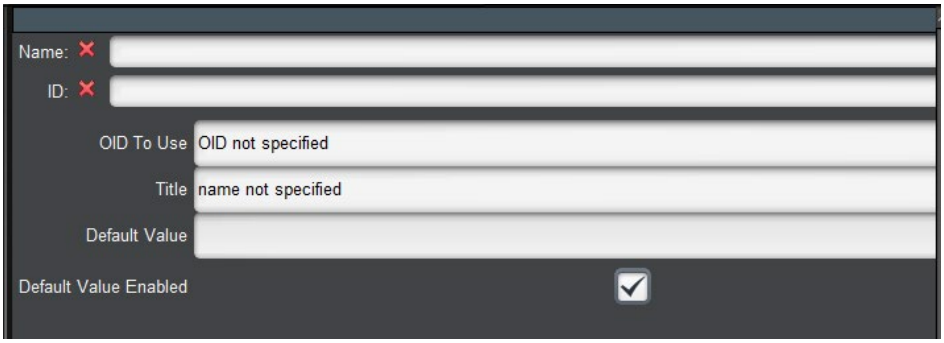


Figure 59 - Keypad Config Dialog

The widget descriptor to generate this widget is shown below. Comments have been added before various sections of the code to identify their functionality.

The `config` block defines four parameters:

- `Ext.Punch.Name` – OID whose value the punchpad will manipulate
- `Ext.Punch.DisplayName` – Name to display in the title bar of the widget
- `Ext.Punch.Default` – Value to set if the DFLT button is pressed
- `Ext.Punch.DefaultEnabled` – Enables/Disables the DFLT button

There is an `oglml` block within the `config` section to specify the layout of the configuration parameters in the Edit Component dialog.

This widget implements an ogScript function, `addDigit()`, to update the param value as the user types in the keypad.

The `oglml` section lays out the keypad using a table container, and hooks the `addDigit()` function to the buttonpress handler for each digit button.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<widgets>
    <widgetdescriptor id="com.rossvideo.widget.punchpad v3"
       icon="com.rossvideo.punchpad.png" inheritsrc="true" name="Punchpad
   v3">

         <!--Configuration section starts here-->
      <config>
          <!-- Variables that appear in the edit mode for the grid file
and that are part of the declaration for the widget -->

          <!—Config parameter declarations start here-->
          <params>
              <param access="1" maxlength="0" name="OID To use"
oid="Ext.Punch.Name"
                  type="STRING" value="OID not specified" widget="0"/>
              <param access="1" maxlength="0" name="OID To use"
                  oid="Ext.Punch.DisplayName" type="STRING"
                  value="name not specified" widget="0"/>
              <param access="1" constrainttype="INT_NULL"
name="Ext.Punch.Default"
                  oid="Ext.Punch.Default" precision="0" strvalue="0"
              type="INT16"
                  value="0" widget="0"/>
              <param access="1" constrainttype="INT_CHOICE" name="Default
Enabled"
                  oid="Ext.Punch.DefaultEnabled" precision="0"
              strvalue="On"
                  type="INT16" value="1" widget="8">
                  <constraint key="0">Off</constraint>
                  <constraint key="1">On</constraint>
              </param>
          </params>
          <!-- Definition for the UI that appears in edit mode -->
```

```
    <!—Config parameter layout starts here-->
    <oglml>
        <abs height="500" left="0" top="272" width="334">
            <table height="150" left="0" top="0" width="800">
                <tr>
                    <label anchor="east" fill="none" insets="0,0,0,5"
                        name="OID To Use" weightx="0.0"/>
                    <param anchor="west" element="0" fill="both"
                        oid="Ext.Punch.Name" showlabel="false"
                        weightx="1.0" weighty="1.0"/>
                </tr>
                <tr>
                    <label anchor="east" fill="none" insets="0,0,0,5"
                        name="Title" weightx="0.0"/>
                    <param anchor="west" element="0" fill="both"
                        oid="Ext.Punch.DisplayName" showlabel="false"
                        weightx="1.0" weighty="1.0"/>
                </tr>
                <tr>
                    <label anchor="east" fill="none" insets="0,0,0,5"
                        name="Default Value" weightx="0.0"/>
                    <param anchor="west" element="0" fill="both"
                        oid="Ext.Punch.Default" showlabel="false"
                    weightx="1.0"
                        weighty="1.0"/>
                </tr>
                <tr>
                    <label anchor="east" fill="none" insets="0,0,0,5"
                        name="Default Value Enabled" weightx="0.0"/>
                    <param anchor="west" element="0" fill="both"
                        oid="Ext.Punch.DefaultEnabled"
                    showlabel="false"
                        weightx="1.0" weighty="1.0"/>
                </tr>
            </table>
        </abs>
    </oglml>
</config>

<!-- Definition for the widget UI itself -->
<oglml>

    <!-- Temporary internal variables to the widget -->
    <!—Local parameter declarations start here-->
    <params>
        <param access="1" maxlength="0" name="Punch.Temp.Number"
            oid="Punch.Temp.Number" type="STRING" value=""
        widget="0"/>
    </params>
```

```
<!-- Global functions for the widget to use -->
<api id="addDigit" name="addDigit">
    function addDigit(digit)
        {
        var value=params.getValue('Punch.Temp.Number',0);
        var i;
        if (digit=='-')
            {
            if (value[0] != '-')
              value = '-' + value;
            else
              value = value.substring(1);
            }
        else if (digit == '.')
            {
            // is there a '.' already?
            for (i=0;i&lt;value.length;i++)
                {
                if (value[i]=='.')
                    return;
                }
            value +='.';
            }
        else if (value[0] != '0')
             value += digit;
        else // first digit is a 0
             value = digit;
        params.setValue('Punch.Temp.Number',0,value);
        }
</api>

<style id="TextStyle" name="TextStyle"
    value="size:20;font:bold;bg#000000;fg#FFFFFF;"/>
<abs height="317" left="641" style="bdr:etched;" top="355"
    virtualheight="317" virtualwidth="371" width="371">
    <abs left="147" top="174"/>

        <!—Title bar begins here-->
        <label height="23" id="Var.Name" left="25"
            name="%value['Ext.Punch.DisplayName'][0]%"
            style="size:16;font:bold;txt-align:west;" top="14"
            width="105"/>
        <param expand="true" height="32" oid="Punch.Temp.Number"
right="20"
            showlabel="false" top="10" width="200"/>

    <!—Table starts here-->
    <table bottom="10" left="20" right="20" top="49">
```

```
                    <!—Table row showing buttons 7, 8, 9, DFLT-->
                    <tr>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="7" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                           <task tasktype="ogscript">addDigit('7');</task>
                        </button>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="8" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                           <task tasktype="ogscript">addDigit('8');</task>
                        </button>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="9" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                           <task tasktype="ogscript">addDigit('9');</task>
                        </button>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="DFLT" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                            <task tasktype="ogscript">
                                var enabled =
                                    params.getValue('Ext.Punch.DefaultEnabled',
       0);
                                var value =
params.getValue('Ext.Punch.Default',0);
                                if (enabled == 0)
                                    return;
                                params.setValue('Punch.Temp.Number', 0,
                                                value.toString());
                             </task>
                        </button>
                    </tr>

                    <!—Table row showing buttons 4, 5, 6, CLR-->
                    <tr>
                         <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="4" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                            <task tasktype="ogscript">addDigit('4');</task>
                         </button>
                         <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="5" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                            <task tasktype="ogscript">addDigit('5');</task>
```

```
                    </button>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="6" rowspan="1" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task tasktype="ogscript">addDigit('6');</task>
                    </button>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="CLR" rowspan="1" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task asktype="ogscript">
                         params.setValue('Punch.Temp.Number', 0, '0');
                      </task>
                    </button>
                </tr>

                <!—Table row showing buttons 1, 2, 3, Enter-->
                <tr>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="1" rowspan="1" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task tasktype="ogscript">addDigit('1');</task>
                    </button>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="2" rowspan="1" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task tasktype="ogscript">addDigit('2');</task>
                    </button>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="3" rowspan="1" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task tasktype="ogscript">addDigit('3');;</task>
                    </button>
                    <button buttontype="push" colspan="1" fill="both"
height="43"
                      name="ENTR" rowspan="2" style="style:TextStyle;"
                      weightx="1.0" weighty="1.0" width="58">
                      <task tasktype="ogscript">
                         params.setValue('%value['Ext.Punch.Name'][0]%'
                         ,0,
                                    params.getValue('Punch.Temp.Numbe
                               r',0));
                         params.setValue('Punch.Temp.Number', 0, '0');

                      </task>
                    </button>
                </tr>
```

```
                    <!—Table row showing buttons +/- 0-->
                    <tr>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="+/-" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                            <task tasktype="ogscript">addDigit('-');</task>
                        </button>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="0" rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                           <task tasktype="ogscript">addDigit('0');;</task>
                        </button>
                        <button buttontype="push" colspan="1" fill="both"
height="43"
                            name="." rowspan="1" style="style:TextStyle;"
                            weightx="1.0" weighty="1.0" width="58">
                            <task tasktype="ogscript">addDigit('.');</task>
                        </button>
                    </tr>
                </table>
            </abs>
        </oglml>
    </widgetdescriptor>
</widgets>
```

Descriptor Location

Descriptors may be defined within an OGLML document, stored in an external file, or retrieved directly from a device.

Inline Widget Descriptors

Descriptors are defined within the `<meta>` block of an OGLML document. Descriptors may not be nested within other widget descriptors. All widget descriptors must be placed within a `<widgets>` block within the `<meta>` block.

External Widget Descriptor Files

The widget descriptor may be stored in an external file. External widget descriptor files have the extension `.widgetdescriptor`.

DashBoard searches for widget descriptors in the following locations:

- Within a `widgets` subfolder within the folder containing the OGLML document.
- Within the `widgets` folder inside the DashBoard installation directory.
- A file specified by use of the `baseurl` attribute of a `widgetdescriptor` tag.

Device-served Widget Descriptors

A device may specify a URL to retrieve widgets using reserved OID `0xFF14`. This mechanism

will retrieve a single file from the specified URL. This is the recommended approach for openGear device developers.

Parameter Mapping

Parameters declared within the config block are visible to the configuration editor and the widget itself. External ogScript functions may access these parameters via the **getConfigParams** function of the widget object.

Parameters declared within the oglml block are private to the widget, and not visible to the config block.

Global parameters are visible within the widget when referenced explicitly.

Widgets also support relative parameters. When a relative parameter is referenced, its name is concatenated to the string specified in the **baseOID** attribute of the widget instance.

Parameters within a widget are interpreted as relative if one of the following conditions is met:

- the OID begins with a `"."`
- the parameter has the attribute `relative` set to `true`.
- the parameter reference explicitly specifies the baseOID by explicitly prefixing `%baseoid%` to the OID of the parameter.

A parameter may force reference to a local parameter by prepending `%widget%` to the OID of the referenced parameter.

Example

The widgetdescriptor `keyer` references the parameters `.clip` and `.gain`. Global parameters are created called `keyer1.clip`, `keyer1.gain`, `keyer2.clip` and `keyer2.gain`. The widgets then may be instantiated as:

```
<widget id="key1" widgetid="keyer" baseoid="keyer1"/>
<widget id="key2" widgetid="keyer" baseoid="keyer2"/>
```

The key1 widget's parameters `.clip` and `.gain` are concatenated with the **baseoid** `keyer1` thus mapping them to the global parameters `keyer1.clip`, and `keyer1.gain`. In a similar manner, key2 widget's parameters map to the global parameters `keyer2.clip`, and `keyer2.gain`.

> The **baseoid** attribute may be queried and modified dynamically through the ogScript **getBaseOID** and **setBaseOID** member functions of the widget object.

Using DashBoard Prebuilt Custom Widgets

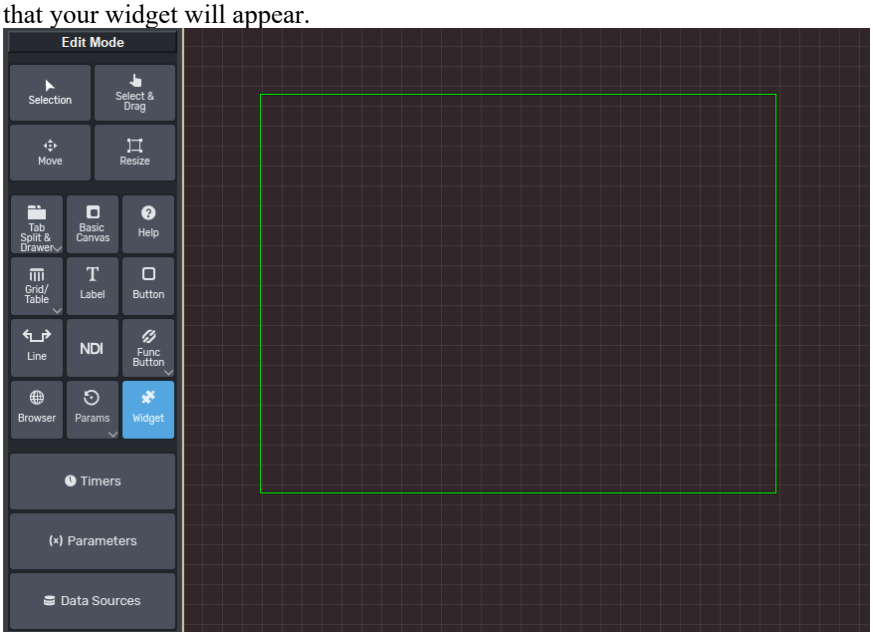
DashBoard provides several prebuilt custom widgets that can be customized for use in your CustomPanel. You can access these custom widgets from the **Widget** button on PanelBuilder **Edit Mode** toolbar. Check if you can leverage one of the existing widgets, by referring to the list of widgets below:

- **ogScript Macro Group** — This widget allows you to create scripts in the Visual Logic editor and presents as either a list of buttons or a playlist.
- **XPression Desktop Preview 1.0 —** This widget allows you to preview XPression playlists from a DashBoard CustomPanel.
- **XPression CountDown 1.0 —** This widget allows you to create an XPression Countdown timer.

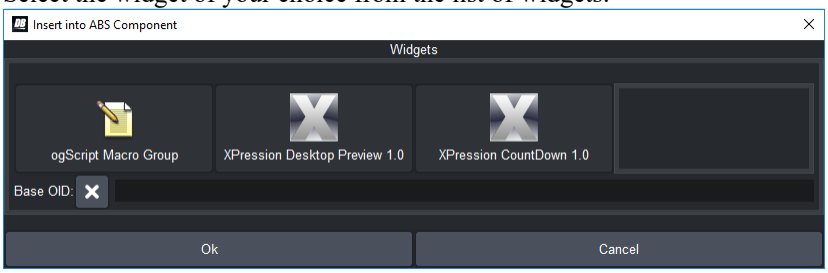
Simply follow the instructions below to add a custom widget to your DashBoard CustomPanel, and then refer to the additional implementation steps for the widget of your choice.

To Add a Custom Widget in DashBoard

1. Open DashBoard and select **PanelBuilder Edit Mode**.
   The Edit Mode toolbar appears.
2. Click the **Widget** button and click and drag your mouse on the canvas to determine the area that your widget will appear.



3. Select the widget of your choice from the list of widgets.



4. Click **Ok** and then refer to the additional instructions for that widget.

ogScript Macro Group Widget

This widget allows you to create scripts using the Visual Logic editor and can be displayed as either a list, a playlist, or buttons. The display types are shown below:

List (Default) Playlist Buttons

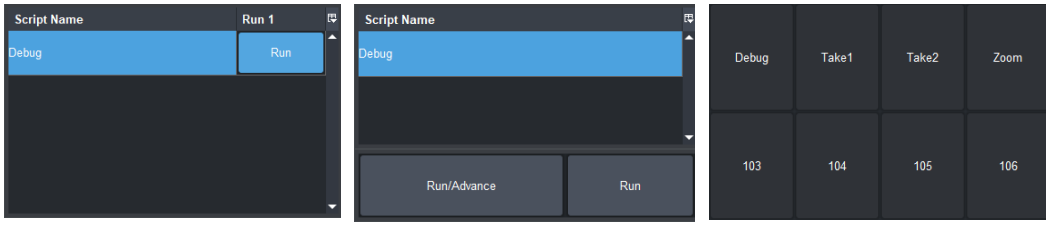


Figure 60: Display Types

Important: Unalike most DashBoard components and Device UIs, you can only add scripts to this widget when the panel is live. This is useful if you need to make live changes to scripts without switching to PanelBuilder Edit Mode. When the panel is live, you can edit this widget directly by right-clicking and selecting **Edit**.

The figure below illustrates the two areas that you must edit the widget:

Live Panel:
This opens the Script/Trigger Editor.

Edit Mode:
This opens the Component Editor.

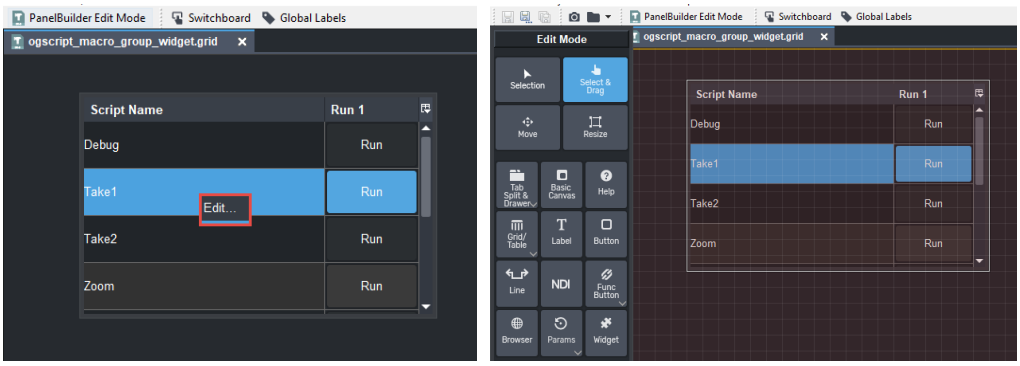


Figure 61: The live panel's edit button is shown on the left and the panel in PanelBuilder Edit Mode is shown on the right.

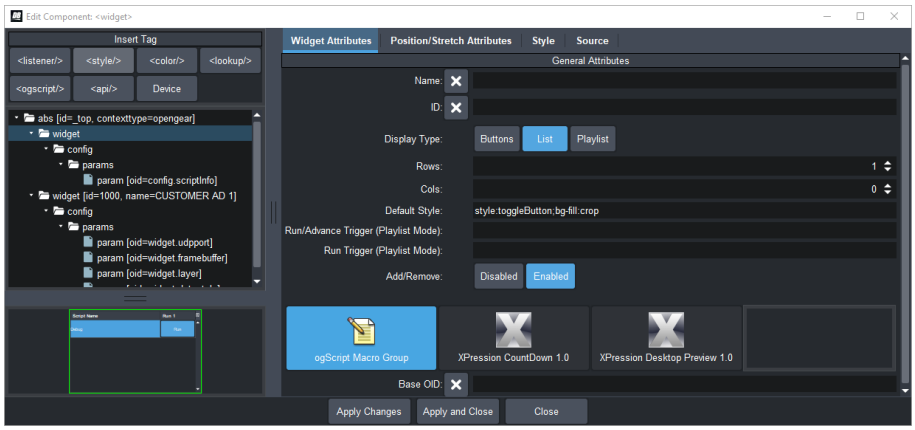
Additional resources can be found about Visual Logic in **the DashBoard User Guide**.

To Configure the ogScript Macro Group Widget

These instructions assume you have already added this widget to your DashBoard CustomPanel from the **Edit Mode** toolbar under **Widgets** and are ready to configure it.

- For more details see, To Add a Custom Widget in DashBoard.

1. To change the display type, double-click on the widget to open the **Component Editor**.



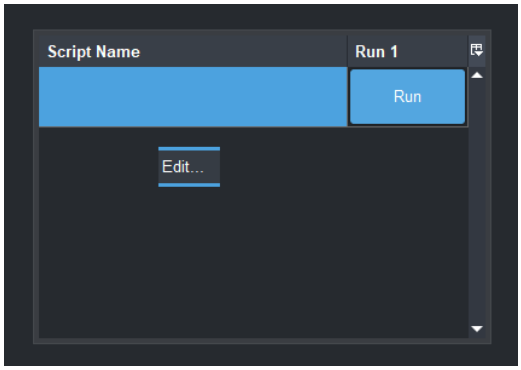
From the Widget Attributes tab under **Display Type**, select **Buttons**, **List** (default), or **Playlist** as your preferred display type and apply your changes.

2. To navigate to the live panel editor, on the top toolbar click **PanelBuilder Edit Mode** to exit Edit Mode.

   Tip: You can also press the keyboard shortkey to switch modes (**CTRL + G**).

   If your canvas does not have grid marks, then you are in live mode.

   To edit the live panel, right-click on the widget and click **Edit**.



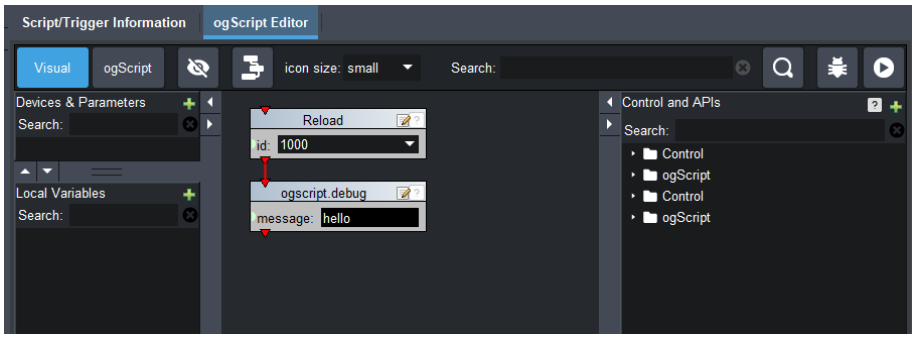
*The **Script/Trigger Information** Editor opens.*

Warning: Any changes you apply in this editor will occur immediately since the panel is live.

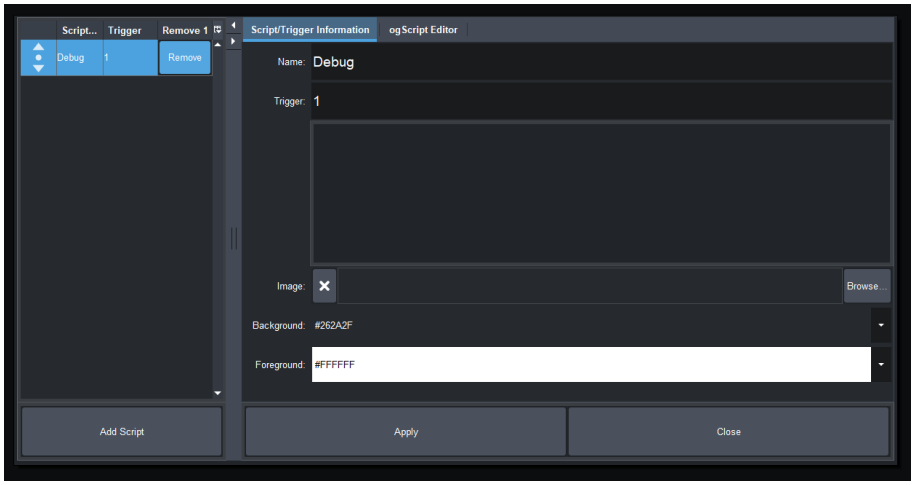
3. Once the **Script/Trigger Information** tab opens, enter the following:

   - **Name** — Enter the script name.
   - **Trigger** — Enter the trigger ID.
   - **Image** — Select an image from the file browser. (optional)
   - **Background** — Select a background color. (optional)

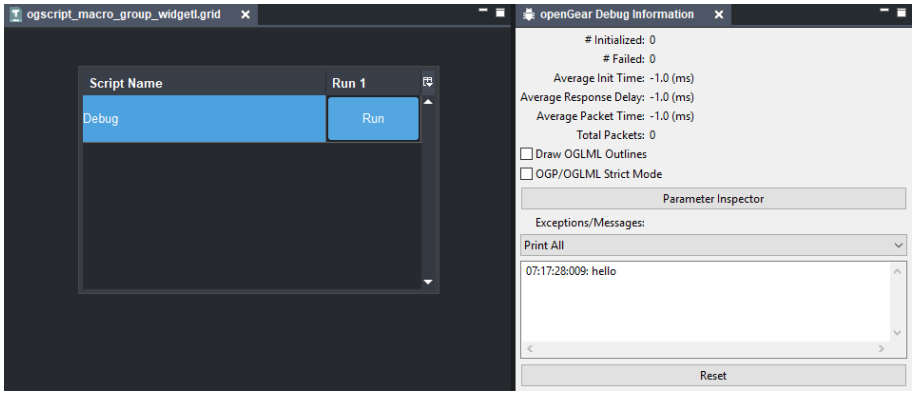
4. Open the **ogScript Editor** tab from the top menu, and create a script using the Visual Logic blocks or ogScript palette.



5. Go back to the **Script/Trigger Information** tab and click **Add Script** to add the script you created to the list on the right side.



6. Click **Apply** to apply your changes and then close the editor.
7. To verify that your script works as intended click **Run**.



XPression Desktop Preview 1.0

Setting up the XPression Desktop Preview in DashBoard

Before you begin, you must have already created your first DashBoard channel and completed the initial configuration to allow streaming through a global style.

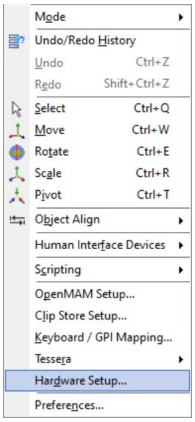
Note: XPression Version 10.0 or later is required.

To Configure the XPression Desktop Preview in DashBoard

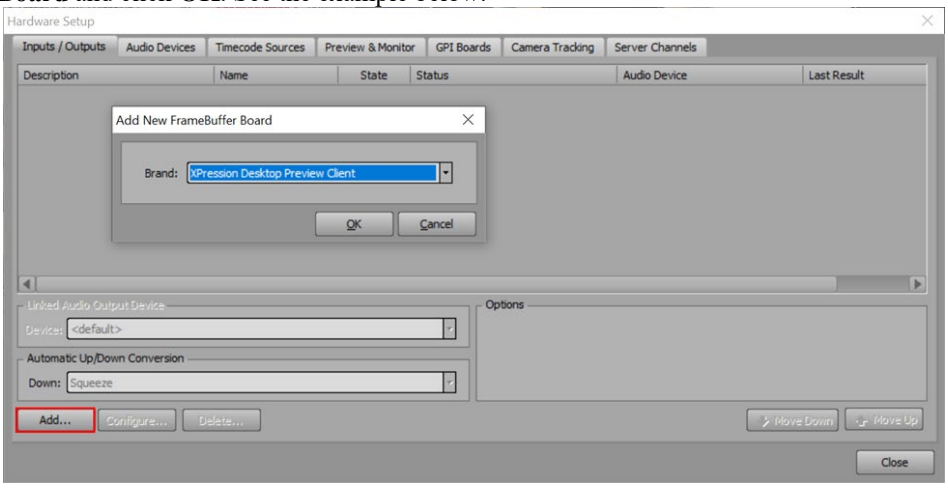
These instructions assume you have already added this widget to your DashBoard CustomPanel from the **Edit Mode** toolbar under **Widgets** and are ready to configure it.

- For more details see, To Add a Custom Widget in DashBoard.

1. To configure your XPression Desktop Preview Client, open XPression studio v.10.0 or later.
2. Click **Edit** from the top menu, select **Hardware Setup**.
   The hardware Setup Dialog box opens.

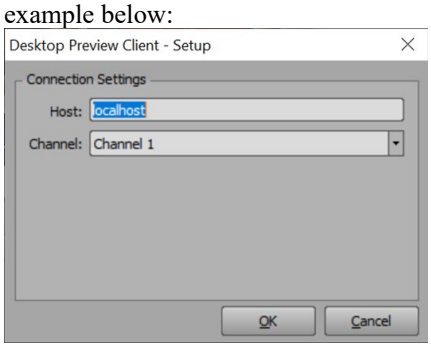


3. To add a new desktop preview, select the **Inputs/Outputs** tab and click **Add**.
4. Select a new **XPression Desktop Preview Client** from the **Add New FrameBuffer Board** and click **OK**. See the example below:

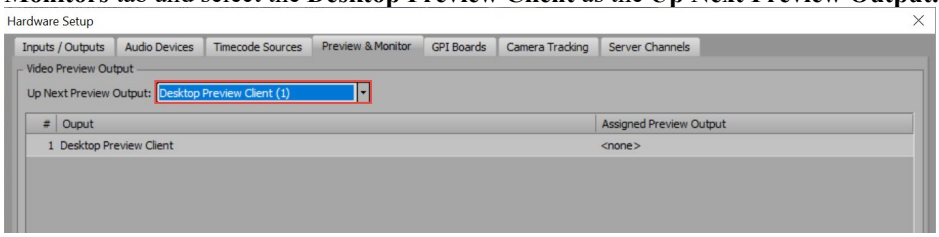


5. Set the **Host Address** to **localhost**, ensuring both DashBoard and XPression running on the same system.

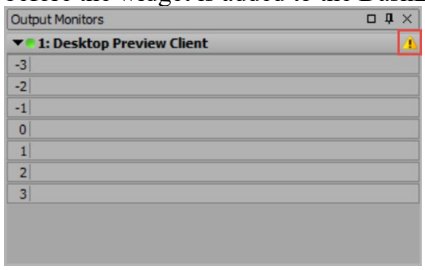
6. Select **Channel 1** (the channel of the **Desktop Preview Client**) and click **OK** to ensure that the channel option does not correlate with the **Output Monitors** channel. See the example below:



7. To assign the **Desktop Preview** Client to a Preview Output Monitor, open the **Preview Monitors** tab and select the **Desktop Preview Client** as the **Up Next Preview Output.**



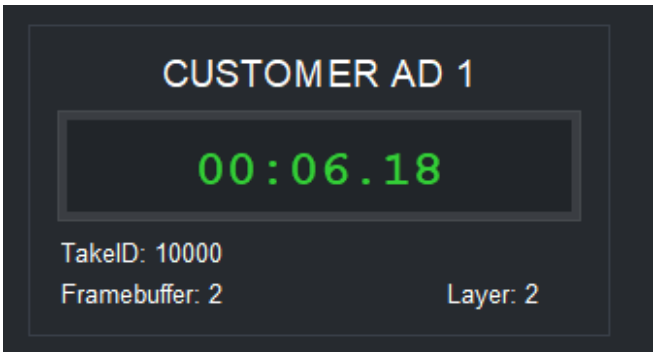
Note: In the Sequencer Playlist left column, select Output Monitors to view the status the status of the Desktop Preview Client. The status should show the status is *not connected* before the widget is added to the DashBoard panel.



Now that you have successfully generated a preview for the focused items in XPression, you can proceed to add the XPression Desktop Preview widget to your DashBoard CustomPanel.

XPression CountDown 1.0

The XPression Countdown widget allows you to monitor a specific framebuffer and layer of XPression to determine what take item is currently on that layer, and the amount of time left for that take item.



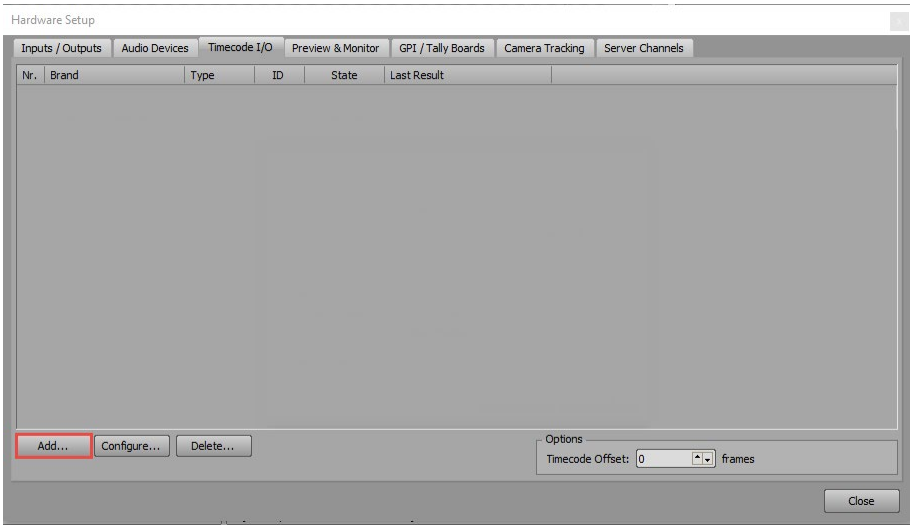
To Configure the XPression CountDown Widget

Before you begin, you must set up the CountDown Timer Broadcast on the XPression. Then you can add the widget to your DashBoard CustomPanel.

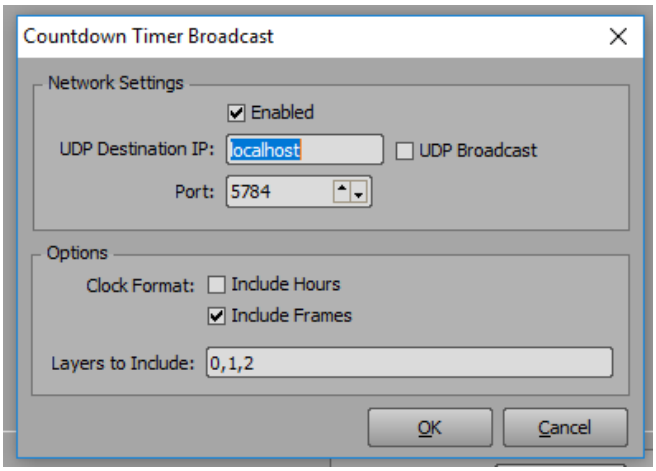
Note: XPression Version 10.0 or later is required.

Set up the CountDown Timer Broadcast on the XPression

1. To configure your XPression to broadcast the countdown data, open XPression Studio v.10.0 or later.
2. Click **Edit** from the top menu, select **Hardware Setup**.
   The hardware Setup Dialog box opens.
3. To add a countdown timer broadcast, select the **Timecode I/O** tab and click **Add**.

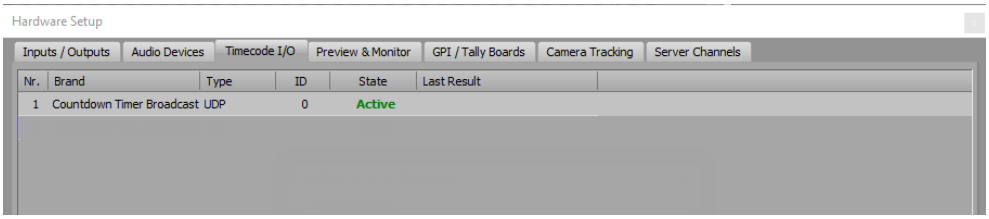


4. Select the **Countdown Timer Broadcast** and fill in the appropriate Network Settings and Options. Take note of the **UDP Port** because you need to enter the same port information in DashBoard later.



Click **OK**.

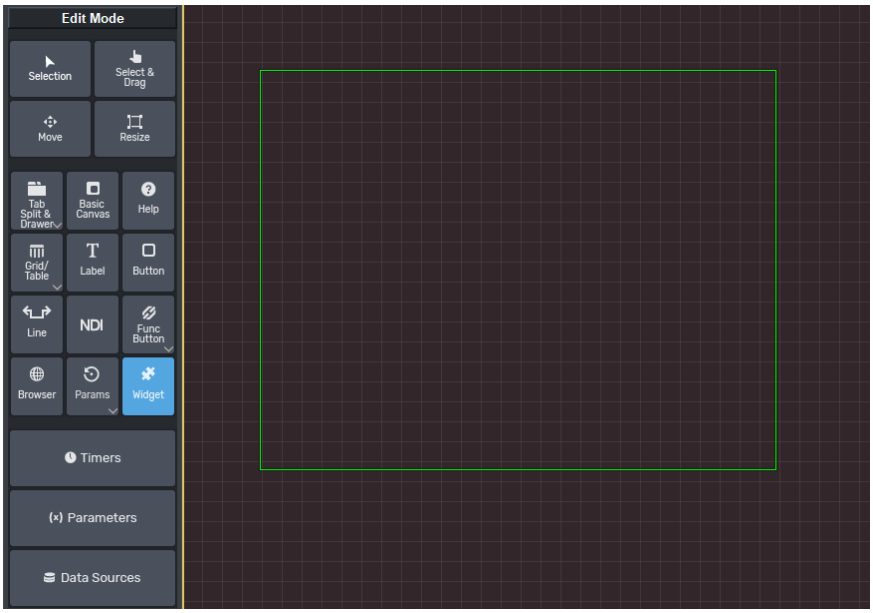
5. Verify that the state is **Active**.



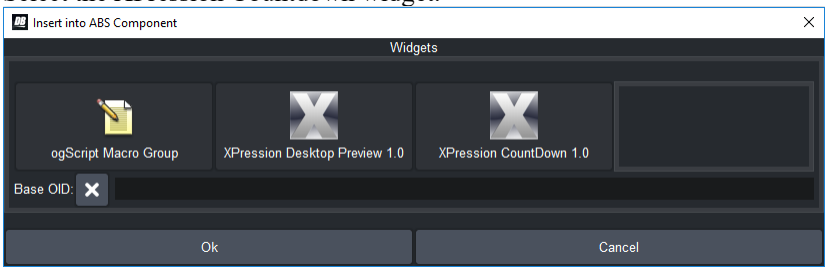
Now that you have successfully broadcast the Countdown Timer from XPression, you can proceed to add the XPression CountDown Timer widget to your DashBoard CustomPanel.

Add the Widget to the DashBoard CustomPanel

1. Open DashBoard and either create a CustomPanel or open an existing one. Select **PanelBuilder Edit Mode**.
   The Edit Mode toolbar appears.
2. Click the **Widget** button and click and drag your mouse on the CustomPanel canvas to determine the area that your widget will appear.

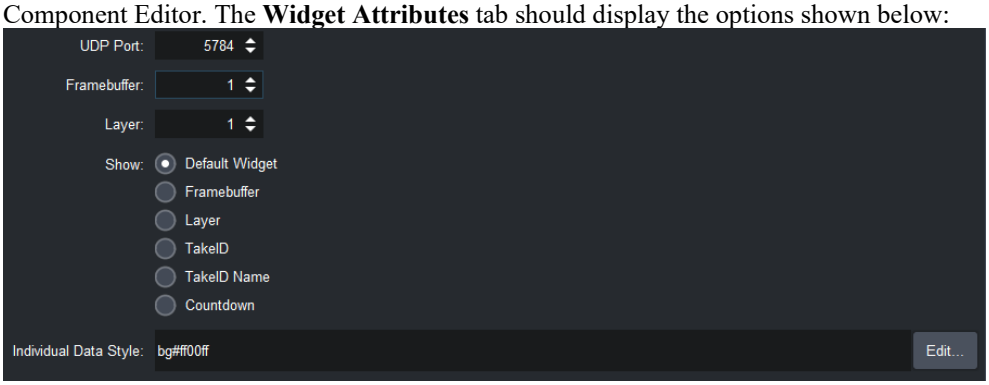


3. Select the XPression Countdown widget.



Click **Ok**.

4. After you have added the **XPression CountDown 1.0** widget, double-click to open the Component Editor. The **Widget Attributes** tab should display the options shown below:



Select from the following options:

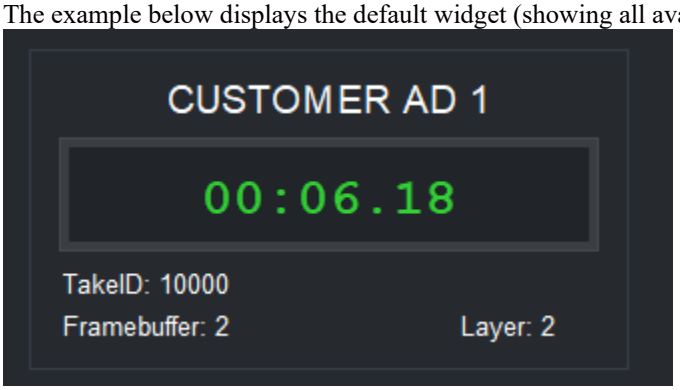
- **UDP Port** — Enter the number of the port that XPression sends the take information to.
- **Framebuffer** — Enter the number of the framebuffer that you want to listen for.
- **Layer** — Enter the layer ID that you want to listen for.
- **Show** —Select **Default Widget** to display information for all fields or select an individual field to display a single field.

- **Individual Data Style** — If you selected an individual field to be shown in the previous option, then click **Edit** to select a style for that field. This config option does not apply to the **Default Widget** option.

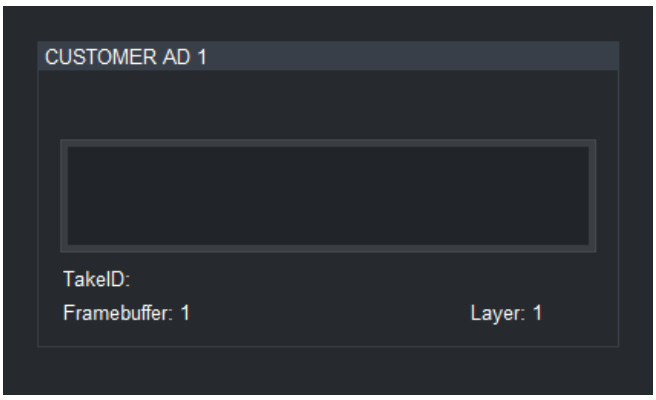
Apply your changes.

5. Verify that your widget displays the appropriate fields. If you decided to show only one individual data field, then only a label will appear.

   The example below displays the default widget (showing all available fields):



   Tip: If the countdown timer data is not displayed, as shown below, go back to troubleshoot the XPression CountDown Timer Broadcast and ensure that the XPression broadcast and the DashBoard widget are set to use the same UDP port.



Custom APIs Within CustomPanels

You can use OGLML's <api> tag to create a library of reusable ogScript code segments (APIs) within a CustomPanel.

You can also save ogScript code segments as JavaScript files (.js), and reference them from within <api> tags. This allows you to maintain an ogScript library that can be used by any of your CustomPanels.

The <api> tag provides a location for global ogScript code. Contents of the <api> tag are processed by the ogScript compiler directly. Elements within an <api> tag are scoped where they are declared in the XML; siblings and children of siblings have visibility to elements declared within the <api> tag.

The <api> tag should generally be placed within a <meta> tag for global ogScript code encapsulation. However, ogScript code intended to dynamically generate and modify the XML should be placed in a top-level <api> tag.

Syntax

```
<api>
   global-scope elements
</api>
```

Attributes

None.

This section includes information about about how to use the <api> tag effectively. It contains the following topics:

- Lexical Order and Loading Order
- Enabling Reuse by Keeping APIs in Separate Files
- Managing Scope

Lexical Order and Loading Order

<api> tags load in lexical order (the order in which they appear in the .grid file) unless the immediate attribute is set to true . When multiple <api> tags are set to load immediately, they load in lexical order relative to each other, but before any non-immediate <api> tags.

Interaction with On Load Handlers

DashBoard provides change handlers that are triggered by certain events. The loading of the panel is one such event. These are also triggered in lexical order, so, if an onload handler needs to use code that is defined in an <api> tag, one of these conditions must be met:

- The <api> tag being used by the onload handler must appear before the handler in the .grid file.
- The immediate="true" attribute of the <api> tag must be set, to load the API immediately.

Example to Demonstrate the Effects of Lexical Order and Loading Order

This section consists of a five-part example that illustrates the effects of lexical order and loading order.

Example – Part 1: Simple API Plus an onload Handler

The first part of the example has an <api> tag that defines a pretty printer function and prints the global namespace to the debug pane.

The <api> tag is followed by an ogscript element that handles the onload event for the enclosing top-level canvas.

Here is the code:

```
<abs contexttype="opengear" id="main-abs">
  <meta>

    <api id="api-pretty" name="Pretty Printer">
      // pretty printer
      function pretty (obj) {
        return JSON.stringify(obj, null, 2);
      }
      // print global namespace to debug pane
      ogscript.debug ('first api:\r\n' + pretty(this));
    </api>

    <ogscript handles="onload" id="main-abs-onload"
      name="Main onload handler" targetid="main-abs">
      ogscript.debug ('First onload handler');
    </ogscript>

  </meta>
</abs>
```

When the panel is loaded, the output appears as follows:

```
13:05:10:759: first api:
{}
13:05:10:759: First onload handler
```

```
Note that the global name space is reported as an empty object {}
because, although we defined the function pretty(), we didn't assign it
to a var.
```

Also note that the onload prints out after the API. In the next example, the lexical order of the onload handler and the <api> tag are reversed.

Example – Part 2: .grid File with <api/> Defined After <ogscript/> Element

In the second part of the example, the <api> tag appears after the ogScript onload handler:

```
<abs contexttype="opengear" id="main-abs">
  <meta>

    <ogscript handles="onload" id="main-abs-onload"
      name="Main onload handler" targetid="main-abs">
      ogscript.debug ('First onload handler');
    </ogscript>
```

```
    <api id="api-pretty" name="Pretty Printer">
      // pretty printer
      function pretty (obj) {
        return JSON.stringify(obj, null, 2);
      }
      // print global namespace to debug pane
      ogscript.debug ('first api:\r\n' + pretty(this));
    </api>

  </meta>
</abs>
```

When the panel is loaded, the output appears as follows:

```
13:11:35:480: First onload handler
13:11:35:491: first api:
{}
```

The output shows that the lexical order of onload handlers and APIs is significant.

The next part of the example adds another `<api>` tag to the CustomPanel, to put an object into the global namespace.

Example – Part 3: Putting an Object in the Global Namespace

The third part of the example is the same as the second part, except that it has an additional `<api>` tag that puts an object into the global namespace.

```
<abs contexttype="opengear" id="main-abs">
  <meta>

    // Code from Example Part 2: onload handler and first API
    <ogscript handles="onload" id="main-abs-onload"
      name="Main onload handler" targetid="main-abs">
      ogscript.debug ('First onload handler');
    </ogscript>

    <api id="api-pretty" name="Pretty Printer">
      // pretty printer
      function pretty (obj) {
        return JSON.stringify(obj, null, 2);
      }
      // print global namespace to debug pane
      ogscript.debug ('first api:\r\n' + pretty(this));
    </api>

    // Additional second API for Example Part 3:
    <api id="api-second" name="Second API">
      // define object in global namespace
      var animal = {
        type: 'tortoise'
```

```
      }
      // print global namespace to debug pane
      ogscript.debug ('second api:\r\n' + pretty(this));
    </api>

  </meta>
</abs>
```

When the panel is loaded, the output appears as follows:

```
13:20:56:437: First onload handler
13:20:56:447: first api:
{}
13:20:56:447: second api:
{
  "animal": {
    "type": "tortoise"
  }
}
```

The output shows that the `onload` handler and the APIs have loaded in their lexical order, and we've now added the `var` object named `animal` to the global namespace.

The next part of the example adds one more `<api>` tag that purposely conflicts with the animal defscription we just defined.

Example – Part 4: Adding an <api> Tag that Conflicts with a Previous <api> Tag

The third part of the example introduced an `<api>` tag that defined a `var` named `animal`, with a `type` value of `tortoise`. This `var` exists in the global namespace.

The fourth part of this example contains an additional `<api>` tag that also defines a `var` named `animal`, in conflict with the previous API.

The code for this part of the example is the same as before, except that a third `<api>` tag is added:

```
    // The following API will conflict with a previous API
    // Insert it after the second API.
    <api id="api-third" name="Third API">
      // define object in global namespace
      var animal = {
        type: 'hare'
      }
      // write out global namespace.
      // Note that this uses pretty() from in another api
      ogscript.debug ('third api:\r\n' + pretty(this));
    </api>
```

When the panel is loaded, the output appears as follows:

```
13:33:24:091: First onload handler
```

```
13:33:24:098: first api:
{}
13:33:24:098: second api:
{
  "animal": {
    "type": "tortoise"
  }
}
13:33:24:099: third api:
{
  "animal": {
    "type": "hare"
  }
}
```

The output shows that the animal's `type` has been redefined as a `hare` instead of a `tortoise`. The next part of the example sets the `immediate=true` attribute for the third (`hare`) API.

Example – Part 5: API Definition with immediate="true"

The fifth part of the example demonstrates the importance of order. The third API has the attribute `immediate = 'true'`, which means it is to be loaded before others that do not have their `immediate` attributes set to `true`, regardless of the order they appear in the code.

The third API uses the `pretty()` function. Because the third `<api>` tag is processed before `pretty()` is defined, an error results.

The code is the same as before, except the following line:

```
<api id="api-third" name="Third API">
```

Is replaced with this:

```
<api id="api-third" immediate="true" name="Third API">
```

Because the `<api>` tag that appears last in the code has its `immediate` attributes set to `true`, it is loaded before all others, and before the `onload` handler.

When the panel is loaded, the output appears as follows:

```
EXCEPTION:
ReferenceError: "pretty" is not defined. (Element API: api-third#7)
```

Because this `<api>` tag was invoked before all the other ones, the `pretty()` function it uses from another API isn't yet defined, so we get an error.

You can fix this problem in one of two ways:

1. Removing the `immediate="true"` attribute, unless it is required. Removing it and ensuring the API was loaded after APIs upon which it depends would fix the problem.

2. Set the `immediate="true"` attribute for the `<api>` tag that provides `pretty()`. As long as it's lexically in front of the third API, we'll get the behaviour we want or expect, as shown here:

```
13:47:33:833: first api:
{}
13:47:33:834: third api:
{
  "animal": {
    "type": "hare"
  }
}
13:47:33:864: First onload handler
13:47:33:865: second api:
{
  "animal": {
    "type": "tortoise"
  }
}
```

The final output shows:

- those <api> tags set to load immediately did so before both the onload handler, and the second <api> tag that didn't have immediate="true" set.
- the immediately loaded <api> tags loaded in their relative lexical order, so the third <api> tag could use the pretty() function defined in the first.
- the tortoise beat the hare because two <api> tags defined the same global variable, and the last one to do so "won" the race.

The final .grid file for this example is as follows:

```
< abs contexttype="opengear" id="main-abs">
  <meta>

    <ogscript handles="onload" id="main-abs-onload"
      name="Main onload handler" targetid="main-abs">
      ogscript.debug ('First onload handler');
    </ogscript>

    <api id="api-pretty" immediate="true" name="Pretty Printer">
      // pretty printer for objects
      function pretty (obj) {
        return JSON.stringify(obj, null, 2);
      }

      // print global namespace to debug pane
      ogscript.debug ('first api:\r\n' + pretty(this));
    </api>

    <api id="api-second" name="Second API">
      // define object in global namespace
      var animal = {
        type: 'tortoise'
```

```
      }

      // print global namespace
      ogscript.debug('second api:\r\n' + pretty(this));
    </api>

    <api id="api-third" immediate="true" name="Third API">
      // overwrite the existing definition of animal
      var animal = {
        type: 'hare'
      }

      // write out global namespace
      ogscript.debug('third api:\r\n' + pretty(this));
    </api>

  </meta>
</abs>
```

Loading order with Minimal Mode and Subscriptions Protocols

If a device developer has implemented support for Minimal Mode and Subscriptions protocols on an openGear protocol (OGP) device and the DashBoard CustomPanel that interacts with it also supports these protocols, then when the panel loads it will only load parameters from the device's indicated minimal set of parameters and its' subscriptions list of OIDs. This also applies to Ross products that support these protocols, like Ultrix and Ultritouch.

Tip: You can check whether a DashBoard CustomPanel supports subscription by double-clicking on the empty canvas to open the Component Editor, and navigating to the topmost level of the panel's source code to verify whether a `subscription="true"` tag is present in the top level attributes. Alternatively, if multiple device contexts are used, the subscription tag may only appear in the <context/> tag. In the DashBoard user interface, you may also notice additional options are available for devices that support subscriptions, and these selection choices remain inactive if they do not apply to a selected device.

Enabling Reuse by Keeping APIs in Separate Files

Although all the examples in this section show the API code inline with the OGLML, it's good practice to keep them in separate JavasScript files.

This allows you to reference the API code from any CustomPanel, and to effectively update all uses of the API code by editing a single JavaScript file.

We recommend using a naming convention such as the following:

- ***myfile*.js** for 'pure' javascript files that do not contain ogScript or params objects specific to ogScript, and therefore could be used in DashBoard or anywhere else.
- ***myfile*.grid.js** for APIs that rely on using `ogscript` and/or `params` objects.

DashBoard's GUI provides a convenient way to navigate to JavaScript files you wish to include. The impact on the OGLML is to set the `src` attribute like this:

To reference a JavaScript file from within the tag, set the src attribute to the filepath, as in the following example:

```
<api src="file:/path/to/file/myfile.grid.js"/>
```

Benefits of using separate files for APIs include the following:

- You can easily share APIs between different custom panels.
- For ‘pure’ JavaScript files, you can quality assurance tools such as JSHint to weed out programming.

Managing Scope

Everything defined within an `<api>` tag has global scope. This means that naming clashes are likely to occur if you include `<api>` tags from multiple authors.

Consider the following two APIs, each of which contains a function named `initialize()`:

```
// Transcendental Vector Engine API
function initialize (arg1, arg2) {
  // do stuff in <api/> 1
}


// Pressurized Water Reactor API
function initialize (arg1) {
  // do stuff in <api/> 2
  return someValue;
}
```

Whichever of these two APIs appears later in the .grid file overwrites the previous API's `initialize()` function, almost certainly producing undesirable results.

To minimize and ideally eliminate such problems, we strongly recommend the use of JavaScript's module pattern because it minimizes use of the global namespace.

An Internet search for 'Javascript Module Pattern' provides plenty of educational material at some depth. The following section provides a concise summary.

The Module Pattern

The main idea of the module pattern is to keep almost everything private to the module, which is implemented as an immediately invoked function expression, as demonstrated in the following example:

```
var myModuleName = (function() {

  // every object I define here is kept private and cannot be
  // accessed from outside the module because they are contained
  // by a 'closure' which is the space between the outermost
  // curly brackets {…} in this example.

  function initialize () {
    // do stuff
  }

  // The objects I wish to publish are referenced in this JSON object
  // that allows precise control over what is revealed to client apps

  return {
    initialize: initialize
  }
}());
```

Usage from elsewhere in the CustomPanel is simple, and which particular initialize function you you use is clear:

```
TVE.initialize(5, 7); // initialize transcendental vectors
var isSafe = PWR.initialize('Reactor B'); // initializes PWR reactor B
```

The code inside the included JavaScript files might look like this:

Transcendental Vectors Engine <api/>

```
// Code in the file, transcendentalVectorsEngine.grid.js:
// This puts an object called 'TVE' in the global namespace
var TVE = (function() {
  function initialize (arg1, arg2) {
    // do stuff
  }

  // publish API
  return {
    initialize: initialize
  }
}());
```

Pressurized Water Reactor <api/>

```
// Code in the file, pressurizedWaterReactor.grid.js:
// This puts an object called 'PWR' in the global namespace
var PWR = (function () {
  // private attribute - cannot be accessed from within the Custom Panel
  var temp;

  function initialize (arg1) {
    // do stuff
    return temp < 200;
  }
}());
```

OGLML Reference

In This Section

This section describes the OGLML tags.

The following topics are discussed:

- General Attributes
- Style Hints
- Layout/Container Tags
- Widget Tags
- Non-UI Tags
- Device Resource Declarations
- Device Resource Tags
- Macro Expansion

General Attributes

An OGLML document consists of a series of (nested) tags, described in detail in the following sections. Each tag can take optional attributes. The following chart lists attributes that can be used with all tags.

Note that there are also tag-specific attributes; these are discussed in the Tags section.

You can also find more information about

- Using OGP Devices that Support Subscriptions Protocol
    - subscriptions
    - Examples

Syntax

```
<component attribute="value" attribute="value" ... >
```

General Attributes

Attribute	Values	Restrictions	Description
containertype	bottom		Adjust the border and shading of the

Attribute	Values	Restrictions	Description
	inset etched raised lowered tabpage		component. See the examples below.
scroll		Should not be used with “browser” tag. Nesting within another “scrolls” element is not recommended Scrolling for the “menu” tag will always be true.	Indicates that the component created by the tag should be enclosed in a scrollable container. If the display is too small to display the component, horizontal or vertical scrollbars are added.
	true		Provides both horizontal and vertical scrollbars
	false		Provides no scrollbars
	horizontal		Provides only horizontal scrolling
	vertical		Provides only vertical scrolling
	always		Forces horizontal and vertical scrollbars to always be visible.
contexttype	opengear …		A device context is a data structure that contains information about the attributes of a device. The contexttype indicates the type of device or data source in the values.
objectid			**node-id** of the source of parameters within container. The **objectid** is passed onto child elements and containers.
objecttype			Type of device when communicating with an openGear or DashBoard Connect device.
id	*String*	Must be unique within all OGLML files displayed by a device. IDs must only use A-Z, a-z, 0-9, “-“, and “_” characters.	Used to uniquely identify/reference an element in the OGLML file.
subscriptions	*Boolean*	Required for devices that support subscriptions protocol. This attribute can only be added to Layout/Container tags. If you are using more than one	When set to `true`, this flag indicates support for openGear Protocol (OGP) JSON devices that support the Subscriptions Protocol. These protocols significantly improve the handling of OGP JSON device communication by eliminating unnecessary parameter updates.

Attribute	Values	Restrictions	Description
		OGP device with subscriptions support as a data source for your CustomPanel, you can use context (or device context) tags.	
width	*Positive integer*	Required for browser tags	Specifies the preferred with (in pixels) for a component. May be ignored by DashBoard depending on the component.
height	*Positive integer*	Required for browser tags	Specifies the preferred height (in pixels) for a component. May be ignored by DashBoard depending on the component.
style	*style hints*	If a parameter already provides style hints as part of its constraint, style hints should not be overridden with this style tag – results are unpredictable.	openGear Style Hints are used to specify the background color, foreground color, icon, and border for certain components. Refer to Style Hints.

Using OGP Devices that Support Subscriptions Protocol

Using OGP JSON devices that support subscriptions is recommended to:

- Optimize memory usage and communication
- Increase panel efficiency
- Load device panels faster

Overview of Requirements to Support Subscriptions on the CustomPanel Side
DashBoard CustomPanels that have a data source where the OGP device supports the subscription protocol will require the following components to take advantage of subscriptions:

1. The `subscriptions="true"` attribute must be added to either context (device context) tags or a Layout/Container Tags. The following other General Attributes are required: `contexttype="opengear"`, `objectid="my ID Here"`, `objecttype="my DeviceType Here"`.

   Tip: It is recommended that you use the DashBoard **Component Editor** to add the data source, and when you add the device that supports subscriptions you can select the auto-subscribe checkbox to add this automatically.

2. The `subscription oids="oid1,oid2,oid3*"` list must be specified in the panel. This comma separated list supports wildcards and must be added to indicate which device OID updates the DashBoard CustomPanel will always receive. See subscription

   Note: If you followed the tip in step 1 and added a device context, you will still need to add the oids manually to the template provided.

3. **Optional**: You can make use of the new subscribe and unsubscribe functions to modify params Objects.

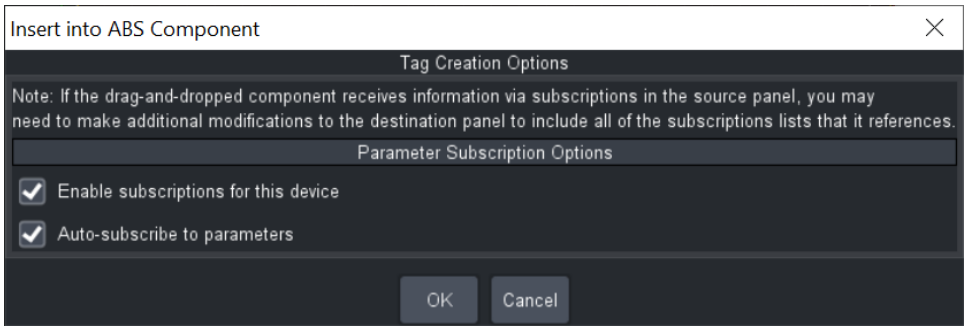
   Tip: You can use the **ogScript Editor > Script Palette** to add these functions.

For related resources, see: context (device context), subscriptions, meta, subscription

subscriptions

When set to **`true`** in a Container or Layout Tag, this attribute indicates support for openGear Protocol (OGP) JSON devices that support the Subscriptions Protocol. These protocols significantly improve the handling of OGP JSON device communication by eliminating unnecessary parameter updates. Panels must also indicate a list of subscription OIDs to receive in addition to the minimal set.

Note: When you drag and drop components from a DashBoard Connect or OGP device panel into another DashBoard panel, you will see a prompt that allows you to automatically enable subscriptions for the device (which adds the `subscriptions="true"` tag to your panel), and/or auto-subscribe to parameters (which adds the list of subscription OIDs).



Tip: If the DashBoard Connect or OGP device does not support subscriptions, then the Parameter Subscription Options will be grayed out throughout the DashBoard UI.

How device communication with the DashBoard Client has changed:

Instead of always receiving a ***full*** set of all the parameter updates from an OGP device, now panels can get a ***minimal*** *set* of parameter updates that is sent by OGP devices. With subscriptions, panels can indicate a list of subscription OIDs to receive in addition to the minimal set.

Note: It is necessary to indicate a list of subscription oids that the panel will always receive parameter updates for from the OGP JSON device. See, subscription and add the list using the oids attribute.

You can see an example of the syntax below for a top level openGear context. In this case an <abs/> absolute container is used, but any layout/container tag is valid.

Syntax of a Subscriptions Panel with Multiple Elements from a Device

```
<abs contexttype="opengear" id="_top" keepalive="false"
objectid="MyUltritouch..." subscriptions="true">
   <meta>
      <subscription oids="oid1, oid2, oid3*"/>
   </meta>
</abs>
```

Example of a Subscriptions Panel with Multiple Elements from a Device

```
<abs contexttype="opengear" id="_top" keepalive="false"
objectid="MyUltritouch..." subscriptions="true">
   <meta>
       <subscription oids="db.touch*,deviceoptions.speakervolume"/>
   </meta>
</abs>
```

Note: To add multiple device sources for the panel, add the **`subscriptions="true"`** attribute to each device context tag, see: context (device context)

Syntax of a Subscriptions Panel with a Device Context Tag

```
<context contexttype="opengear" objectid="DeviceID..."
```

```
subscriptions="true">
   <meta>
      <subscription oids="oid1, oid2, oid3*"/>
   </meta>
</context>
```

Example of a Subscriptions Panel with Two Device Contexts

```
<abs contexttype="opengear" id="_top" keepalive="false"
objectid="MyUltritouch..." objecttype="Ultritouch Device">
   <context contexttype="opengear" objectid="Kyles_Ultritouch..."
subscriptions="true">
      <meta>
         <subscription oids="db.touch*,deviceoptions.speakervolume"/>
      </meta>
    </context>
    <context contexttype="opengear" objectid="Daves_Ultritouch..."
subscriptions="true">
      <meta>
         <subscription oids ="devices*, deviceoptions.lcdbrightness"/>
      </meta>
    </context>
</abs>
```

For related resources, see: context (device context), subscriptions, meta, subscription , Using OGP Devices that Support Subscriptions Protocol

Examples

The following image illustrates the available **containertype** values:

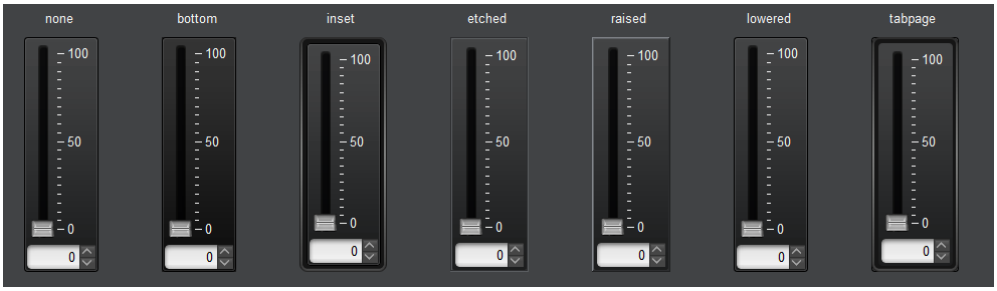


Figure 62 – containertype examples

openGear Style Hints

openGear Style Hints provide something similar to an inline style CSS attribute in HTML. For certain components, they can be used to specify a background color, foreground color, border, and icon. The hints can be provided inside OGLML tags or via parameter choice constraint values.

Syntax

To specify a style hint inside an OGLML tag, the style attribute is used:

```
<component style="style-hint;style-hint;...;" component attributes>
```

To specify a style hint within a parameter choice constraint, the style tag is inserted at the end of the constraint value, enclosed in angle brackets (< >). In order to represent the angle brackets in the OGLML document, they must use standard XML escape sequences (`< >`). Specifying hints within the constraint value allows different styles to be applied to each choice.

```
<constraint key="key">value&lt;style-tag;style-tag;... &gt;</constraint>
```

Note	*Style hints may be specified in **either** the OGLML style attribute **or** within the constraint value, **but not both**.*

For clarity, this document will provide examples using the OGLML style attribute only, however the style hints may be utilized within constraints unless specifically mentioned.

Style Hint Reference

The following style hints are supported:

Tag	Description
#color-value	Sets the component background color.
bdr:*border-style*	Sets the component border style.
bdr#*color-value*	Sets the component border color.
bg#*color-value*	Sets the component background color.
bg-align:*value*	Sets the alignment of a background image.
bg-fill:*value*	Controls how a background image is sized.
b-u:*image-url*	Sets a container background to image located at a specified URL.
di:none	Removes a component drag icon.
di-eo:*external-oid*	Sets a component drag icon to image encapsulated in an external OID.
di-u:*image-URL*	Sets a component drag icon to image located at a specified URL.
f:*style-hint*	Style modifier when button value is false.
fg#color-value	Sets the component foreground color.
font:*font-type*	Sets the font type.
grid#color-value	Sets the table gridline color.
hi:none	Removes a component hover icon.
hi-eo:*external-oid*	Sets a component hover icon to image encapsulated in an external OID.
hi-u: *image-URL*	Sets a component hover icon to image located at a specified URL.
i:none	Removes a component icon.
i-eo:*external-oid*	Sets a component icon to image encapsulated in an external OID.

Tag	Description
i-u: *image-URL*	Sets a component icon to image located at a specified URL.
m:*t,l,b,r*	Sets insets around the label of a button.
o#*color-value*	Sets the text outline color.
size:font-size	Sets the text font size.
style:*style-id*	Applies style hints defined within a **style** tag.
t:*style-hint*	Style modifier when button value is true.
tt:*tool-tip-string*	Sets a tooltip for a label or button.
txt-align:alignment	Sets the alignment of text.

style Style Hint

User styles may be created within an OGLML document to allow standardized formatting to be applied to multiple components. Styles are defined using the `style` tag. A predefined style may be referenced by a component as part of its `style` attribute. Additional style hints may be included in the same style attribute string. If the style string explicitly specifies a hint which contradicts a hint in the predfined style, the explicitly added hint shall supersede.

Style Hint	Values	Restrictions	Description
style:*style-id*	*String*	The style with the provided ID must be defined in an OGLML document at a higher scope than where it is referenced.	Apply the style hints of the style defined in a different set of style hints. See **style** tag documentation for more information.

Examples

The following example applies button style hints as defined in the predefined style `CommandButtonStyle`. Note that the "Stop" button has an additional hint applied (`size:big`), and overrides the background color (`bg#ff0000`).

```
<style id="ButtonStyle" value="bg#808000;bdr:etched;"/>
<button name="Start" style="style:ButtonStyle;"/>
<button name="Stop" style="style:ButtonStyle;size:big;bg#ff0000;"/>
<button name="Reset" style="style:ButtonStyle;"/>
```

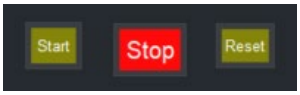


Figure 63 - Style Tag Example

Component Color

The foreground, background and border colors of components may be specified. It is often a good idea to override the background and foreground as a pair to avoid the possibility of the background and foreground being the same (or similar) colors in the UI.

Style Hint	Values	Restrictions	Description
#color-value or bg*#color-value*	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*		Set the background color of the component. Colors may be specified as RGB, RGBA or one of the pre-defined color constants. R, G, B and A are specified as 2-digit hex values.
fg*#color-value*	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*		Set the foreground color of the component. Colors may be specified as RGB, RGBA or one of the pre-defined color constants. R, G, B and A are specified as 2-digit hex values.
bdr*#color-value*	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*		Create a line border around the component with the specified color. Colors may be specified as RGB, RGBA or one of the pre-defined color constants. R, G, B and A are specified as 2-digit hex values.
o*#color-value*	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*		Create an outline around the text within a component with the specified color. Colors may be specified as RGB, RGBA or one of the pre-defined color constants. R, G, B and A are specified as 2-digit hex values.
grid*#color-value*	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*	Applies to table container only.	Specifies the color of the gridlines for a table container. Colors may be specified as RGB, RGBA or one of the pre-defined color constants. R, G, B and A are specified as 2-digit hex values.

Example

The following style tag creates a label using the predefined background `#panelbg` and the foreground (text) in orange.

```
<label name="Label" style="bg#panelbg;fg#FFC000;"/>
```

Predefined Colors

DashBoard defines color constants, which make up the standard color scheme. Color constants are the default colors that are used when you build CustomPanels in DashBoard. These colors are used in the standard controls within DashBoard, but may be applied to the background, foreground or border color style tag of any component. Custom color constants may be defined within an OGLML document using the color tag.

Color constants can be used anywhere in your code in the place of actual values. You can add them in the GUI using the drop-down color palettes available in the **Style** tab of the **Component Editor**.

The following image illustrates the pre-defined color constants in the color palette:

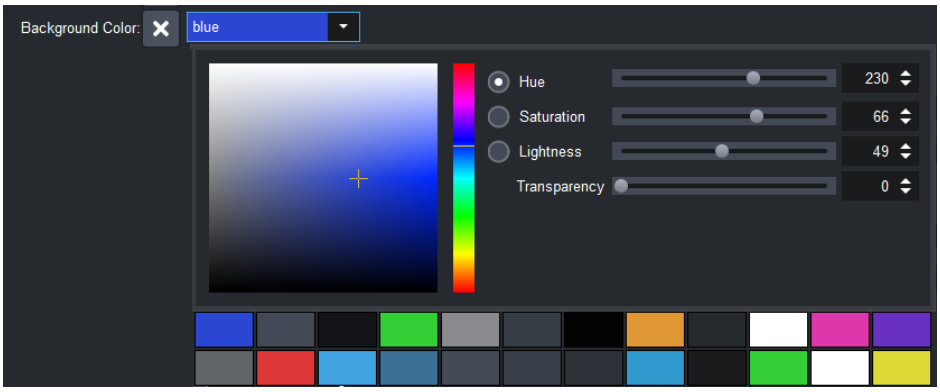


Figure 64 - Predefined color constants

Simply hover your mouse over a color palette color to see the intended standard control usage for the color constant. In the following image you can see the **selectedmuted** color constant identified below:

Using the recommended DashBoard color scheme ensures that

- You are applying colors consistently throughout your UI.
- You are saving time, because you won't need to customize the style of each standard control that you add.
- Your panels will stay current with any new DashBoard color constant changes.

DashBoard defines the following color constants:

Color Constant	Color Sample	Description
#panelbg		Panel background color.
#panelfg		Panel foreground color for a basic control or button.
#selectbg		Background color for a toggle button that is selected.
#selectedmuted		Color for a mute button that is selected.
#buttonbg		Background color for a button.
#tableheader		Color for a table header.
#tablezebra		Secondary color for table rows.
#readonlyborder		Color for a read-only border.
#listbg		Background color for a list.
#tabbg		Background color for a selected tab.
#textbg		Color for background for text.
#lightdivider		Color of a light divider.
#darkdivider		Color of a dark divider.
#modaloverlay		Color for a modal overlay.
#timerfg		Foreground color for a timer.
#red		Red.
#orange		Orange.
#yellow		Yellow.
#green		Green.
#teal		Teal.
#blue		Blue.
#purple		Purple.

Color Constant	Color Sample	Description
#pink		Pink.
#transparent		No fill; the element will be transparent.
#user-defined-color		Color defined by the user using the color tag.

Border Styles

The style of a component border may be specified with the bdr hint. If the border hint is not specified, a simple line will be drawn for the control border.

Note: *The containertype attribute, if specified for a component, will override the bdr style hint.*

Style Hint	Values	Restrictions	Description
bdr:*border-style*	none		Removes the border from the component.
	etched		Create an etched border around the component.
	shadow		Creates a drop shadow under the component.
bdr#*color-value*	Sets border color; see Component Color Section.		
grid#*color-value*	Sets grid color in a table; see Component Color Section.		

Examples

The following image illustrates the border style hint:

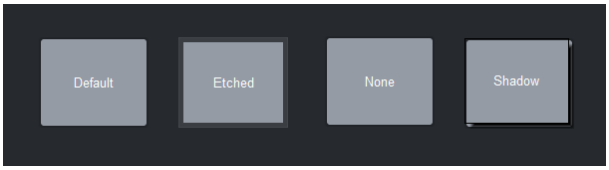


Figure 65 - Border style

Text/Font Styles

The following style hints modify the rendering of text in a component.

Style Hint	Values	Restrictions	Description
size:*font-size*	*Integer size* smaller small **normal** big bigger biggest		Set the font size for the component. Number specifies a font size in points (1/72”). **smaller** corresponds to 2/3 normal size. **biggest** corresponds to 4x normal size. See examples below.

Style Hint	Values	Restrictions	Description
font:*font-type*	**default** bold mono		Set the control font to the default font, a bold font, or a mono-spaced font.
txt-align:*alignment*	**center** north northeast east southeast south southwest west northwest		Controls the position of text within a button or label control. 
fg#*color-value*	Sets text foreground color; see Component Color Section.		
o#*color-value*	Sets text outline color; see Component Color Section.		

Examples

The following image illustrates the size style

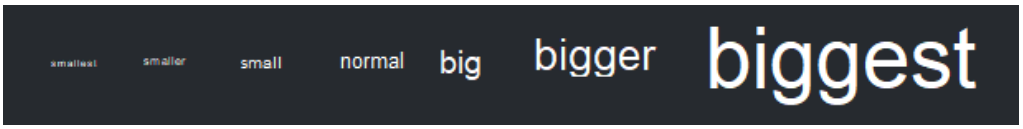


Figure 66 - size style attribute

The following image illustrates the font style:

Figure 67 - font style attribute

Icon Styles

Icon styles may be applied to label and button components. DashBoard allows separate icons to be defined for the default icon, the icon when a mouse hovers over the control, and the icon when the control is dragged (if dragging is enabled on the component).

Note *Icon Styles have no effect on buttons with the* ***flat*** *attribute.*

Style Hint	Values	Restrictions	Description
i-eo:*external-oid*	*External OID*	External OID specified must be type 0x03.	Set the icon for the component (applies to labels and buttons).
i-u: *image-URL*	*URL String*	Full qualified URL to PNG, GIF or JPG image.	Set the icon for the component. (applies to labels and buttons)
i:none			Remove the icon for the component.
di-eo:*external-oid*	*External OID*	External OID specified must be type 0x03.	Set the drag icon for the component (only applies if “dragvalue” attribute is used).

Style Hint	Values	Restrictions	Description
di-u:*image-URL*	*URL String*	Full qualified URL to PNG, GIF or JPG image.	Set the drag icon for the component (only applies if “dragvalue” attribute is used)
di:none			Remove the drag icon for the component.
hi-eo:*external-oid*	*External OID*	External OID specified must be type 0x03.	Set the hover icon for the component (applies to buttons)
hi-u: *image-URL*	*URL String*	Full qualified URL to PNG, GIF or JPG image.	Set the hover icon for the component (applies to buttons)
hi:none			Remove the hover icon for the component.

Example

```
<button buttontype="push" style="i-u:http://my-server/RossLogo.jpg;hi-u:http://my-server/DashBoardLogo.jpg;" />
```

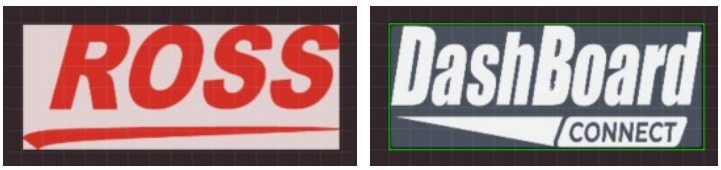


Figure 68 – Background and Hover Icon

Tooltip Style

Tooltip may be added to components. Balloon help text will be displayed when the mouse hovers over the component.

Style Hint	Values	Restrictions	Description
tt:*tool-tip-string*	*String*	May only be applied to label and buttons.	Set the tooltip of the component to the specified String. A “;” can be inserted into the string by inserting the escape sequence “\;”.

Example

```
<button name="Tooltip" style="tt:This is the tooltip text" />
```

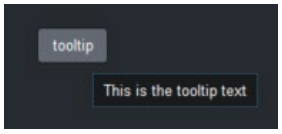


Figure 69 – Tooltip Style

Inset Style

Insets provide a margin from the edge of a component to the text or icon content.

Style Hint	Values	Restrictions	Description
m:*top*,*left*,*bottom*, *right*	*4 Integers*	This hint can only be applied to button widgets.	Sets the margins around the label of the button. The margins are specified in pixels.

Background Styles

Background styles allow images to be placed in the background of container components.

Style Hint	Values	Restrictions	Description
b-u:*image-url*	*URL String*	Must be a fully qualified URL.	Set the background image of the component.
bg-fill:*value*	none		Do not scale the image.
	both		Stretch the image to the width/height of the control.
	horizontal		Scale the image to the width of the control (maintain aspect ratio).
	vertical		Scale the image to the height of the control (maintain aspect ratio).
	fit		Scale the image to the largest size that will fit inside of the control (maintain aspect ratio).
	crop		Scale the image to fill the control maintaining the aspect ratio. Crop the image to remove the parts that don't fit.
	tile		Tile the image (starting at the upper left) to fill the background of the control.
	paint9		Divide the image into 9 areas (defined with Background Insets) to define fixed corners, vertically or horizontally stretched sides, and a stretched center.
bg-align:*value*	**center** north northeast east southeast south southwest west northwest		If the fill is set to anything other than "both" or "tile", this controls where the background is positioned in the component. 

Button Style Modifiers

All style options can be overridden for toggle and radio buttons, such that the style of the widget is determined by the value of the backing parameter. For toggle buttons, the style can be specified for the true state (button toggled down) and false state (button toggled up). For radio buttons, the style can be specified for each choice for the true state (choice selected) and false state (choice not selected).

Syntax

```
<component style="t:true-style-hint;f:false-style-hint;...;" >
```

Hint Modifier	Values	Restrictions	Description
t:*true-style-hint*	Valid style hint		Applies the style hint only when the choice is true.
f:*false-style-hint*	Valid style hint		Applies the style hint only when the choice is false.

Examples

The following example creates a toggle button whose color is green when true (toggled down) and red when false (toggled up):

```
<param oid="0x7" style="t:bg#00ff00;f:bg#ff0000;" widget="13"/>
```

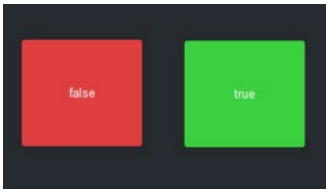


Figure 70 - Toggle Button Style Modifier

The following example changes the font size to `big` for the selected radio button:

```
<param oid="0x6" style="t:size:big;f:size:normal;" widget="9"/>
```

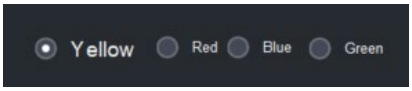


Figure 71 – Radio Button Style Modifier

Layout/Container Tags

Container tags define regions of the layout which contain other elements. Containers control how the child elements are presented within DashBoard. Container tags accept attributes which impact the container as a whole, and may also specify additional attributes which may be applied to child elements; these define how the elements are displayed within the container. Containers may be nested.

The following containers are supported:

Tag	Description
abs	Allows elements to be placed in absolute positions
borderlayout	Creates a border layout that maintains proportions of components anchored to the border edges or center when resized, and offers the option to set one component to grow in relation to the other components
flow	Aligns elements in a horizontal row
pager	Creates a pager control component that is customizable using script
popup	Presents child elements in a popup window
simplegrid	Creates a grid of fixed-size rows and columns

Tag	Description
split	Creates a draggable split screen with 2 components
tab	Creates a tabbed page
table	Creates a grid of rows and columns

abs

Use absolute positioning and sizing for components inside of the **abs** tag. The sizing and positioning of child components must be specified as attributes of those child components.

Syntax

```
<abs container attributes>
      <component child component attributes> </component>
      <component child component attributes> </component>
      . . .
</abs>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified to the <abs> tag:

Attribute	Values	Restrictions	Description
virtualwidth	Integer		Defines a virtual width and height to use for all coordinates inside of the container. All offsets and dimensions inside of the container are scaled based on current width/height vs. virtualwidth/virtualheight. When these attributes are used, the UI will scale as the container ize changes.
virtualheight	Integer		
subscriptions	String		When set to subscriptions="true", this flag indicates support for openGear Protocol (OGP) JSON devices that have implemented both minimal mode and subscription protocol. The minimal mode protocol provides the foundation for the subscription protocol. See the subscriptions entry for more details.

Child Component Attributes

In addition to General Attributes, the following attributes may be specified to child components:

Attribute	Values	Restrictions	Description
left	Integer		Defines the distance between the left edge of the **abs** and the component. When combined with **right** it will force the component to fill the available area.

Attribute	Values	Restrictions	Description
right	Integer		Defines the distance between the right edge of the **abs** and the control. When combined with **left**, it will force the component to fill the available area.
top	Integer		Defines the distance between the top edge of the **abs** and the control. When combined with **bottom**, it will force the component to fill the available area.
bottom	Integer		Defines the distance between the bottom edge of the **abs** and the control. When combined with **top**, it will force the component to fill the available area.
width	Integer	Ignored if both **left** and **right** are specified.	Defines the width of the control. If undefined, the control's calculated preferred size will be used.
height	Integer	Ignored if both **top** and **bottom** are specified.	Defines the height of the control. If undefined, the control's calculated preferred size will be used.

virtualwidth and virtualheight

If the **virtualheight** and **virtualwidth** attributes are not set, components within the abs container will be displayed in their specified size. Resizing the abs container will not scale the child components. If the abs area does not encompass the area required for the specified components, the components will be cropped.

If virtualheight and virtualwidth attributes are set, component size and position within the abs container are scaled according to:

$$component\ display\ width = component\ width \times \left(\frac{abs\ width}{abs\ virtualwidth}\right)$$

$$component\ display\ height = component\ height \times \left(\frac{abs\ height}{abs\ virtualheight}\right)$$

Examples

The following example creates an abs container with 4 buttons placed in a 2x2 grid. The buttons will not scale if the abs is resized (they will be cropped):

```
<abs left="16" top="16" width="250" height="250">
   <button left="5" top="5" width="100" height="100" name="1"/>
   <button left="110" top="5" width="100" height="100" name="2"/>
   <button left="5" top="110" width="100" height="100" name="3"/>
   <button left="110" top="110" width="100" height="100" name="4"/>
</abs>
```

The following example creates an abs container with 4 buttons placed in a 2x2 grid. The buttons will scale if the abs is resized:

```
<abs left="16" top="16" width="250" height="250" virtualwidth="250"
virtualheight="250">
   <button left="5" top="5" width="100" height="100" name="1"/>
   <button left="110" top="5" width="100" height="100" name="2"/>
```

```
    <button left="5" top="110" width="100" height="100" name="3"/>
    <button left="110" top="110" width="100" height="100" name="4"/>
</abs>
```

In the following example, the 2x2 grid will be scaled to half its original size. All buttons will appear 50x50 pixels in size:

```
<abs left="16" top="16" width="125" height="125" virtualwidth="250"
virtualheight="250">
    <button left="5" top="5" width="100" height="100" name="1"/>
    <button left="110" top="5" width="100" height="100" name="2"/>
    <button left="5" top="110" width="100" height="100" name="3"/>
    <button left="110" top="110" width="100" height="100" name="4"/>
</abs>
```

borderlayout

You can use the border layout tool to create an area on a CustomPanel that you can anchor components to and later resize to maintain your intended layout. You can use a border layout to anchor components against any of the four borders of the container and in the center. It is useful for adding menus along the border edge of a CustomPanel, or to group components within a CustomPanel. A border layout must have more than one component, because it is designed to responsively resize multiple objects contained within its borders. Typically, you can have a component anchored to each side, and then a fifth central component. Any component could also be a basic canvas containing other components.

If you want one of the anchored components to grow when the container is resized, you can set the border layout's Growth Quadrant to match the component area you'd like to grow (top, right, bottom, left, or center). You can only set a single growth quadrant. The areas that aren't in the growth quadrant will be adjusted when you resize the border layout container. The components anchored to the top or bottom will keep the same height, while the width expands or minimizes to match the container size. The components anchored to the right or left will keep the same width, while the height expands or minimizes to match the container size.

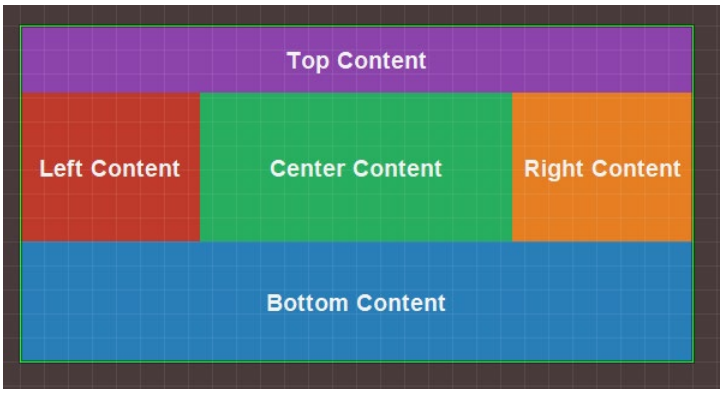
If a Growth Quadrant is not specified in the GUI, the **[default]** border layout will maintain certain proportions of the side components, and the central component will grow when resized. For the top and bottom sides, the height is maintained, and the width will fill the container as it is resized. For the left or right sides, the width is maintained, and the height will fill the container as it is resized.

Use absolute positioning and sizing for components inside of the **borderlayout** tag. The sizing and positioning of child components must be specified as attributes of those child components. Child components are resized based on the specified growth quadrant.

Note: In the child component attributes you must include an **anchor** that is set to **Top**, **Bottom**, **Right**, **Left** or **Center** to specify the side that the component is anchored to. In the source code the anchors are **north**, **south**, **east**, **west** or **center**.

For more details, see the ***DashBoard User Guide***.

You can see an example of a border layout with labels used as the anchored components below:



Syntax

```
<borderlayout container attributes>
      <component child component attributes> </component>
      <component child component attributes> </component>
      . . .
</abs>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified to the `<borderlayout>` tag:

Attribute	Values	Restrictions	Description
grow	String	In the GUI, the growth quadrant must be set to Value must be set to **[default]**, **Center**, **Top**, **Bottom**, **Left** or **Right**. In the source code, the value must be set to **north**, **south**, **east**, **west** or **center**.	This attribute determines which of the anchored components will grow when the border layout is resized responsively. This attribute impacts how the width and height of the child components will behave when the border layout is resized responsively. **Note:** If this attribute is not defined, then by default the **center** component is the only one that will grow when resized. Any components on the border will responsively resize as follows: • **north** or **south** - For the top and bottom sides, the height is maintained, and the width will fill the container as it is resized. • **east** or **west** - For the left or right sides, the width is maintained, and the height will fill the container as it is resized. Note: If set to **[default]** in the GUI, then the **grow** attribute will not appear in the source code, but by default the behavior is the same as **grow="center".**

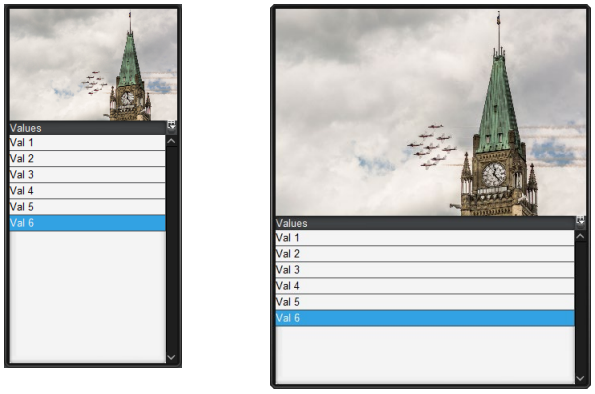
Child Component Attributes

In addition to General Attributes, the following attributes may be specified to child components:

Attribute	Values	Restrictions	Description
anchor	String	In the GUI, the value must be set to **Top**, **Bottom**, **Left** and **Right**. In the source code, the values are shown as **north**, **south**, **east**, **west** or **center**.	Defines the border side which the component will be anchored to.
width	Integer	**Note:** For certain components, you must include this attribute for the component to appear. With labels or buttons, you don't need to include a defined width as the text will determine the width. With <abs/> containers the width is required for the component to appear on the canvas.	Defines the width of the component. This is impacted by the growth attribute.
height	Integer	**Note:** For certain components, you must include this attribute for the component to appear. With labels or buttons, you don't need to include a defined height as the text will determine the height. With <abs/> containers the height is required for the component to appear on the canvas.	Defines the height of the component. This is impacted by the growth attribute.

Example

The following example creates a border layout that is set to grow='north', the label image is set to anchor='north', and the table is set to anchor='center'. The figure below shows the border layout before and after being resized. You can see that when the border layout is resized, the label image grows north, and that the table remains centered, and became shorter to accommodate the label image's growth.




```
<borderlayout grow="north" height="480" style="bdr:etched;" width="220">
   <label anchor="north" height="40" style="bg#dark;bg-u:cd-3.jpg;bg-
   fill:fit;" width="6"/>
   <param anchor="center" expand="true" height="70" oid="params.table"
   showlabel="false" width="250"/>
</borderlayout>
```

For more information and an expanded example, see the ***DashBoard User Guide***.

flow

Arrange controls horizontally across the page. Wrap the controls vertically if there is not enough space to show all controls on a single row.

Syntax

```
<flow container attributes>
   <component component attributes> </component>
   <component component attributes> </component>
   <component component attributes> </component>
   . . .
</flow>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified:

Attribute	Values	Restrictions	Description
anchor	**center** east west		Defines the alignment of the controls 

Default values shown in **bold**.

Child Component Attributes

See General Attributes. There are no additional attributes for child components.

Example

The following example places 6 buttons in a horizontal row, aligned to the left edge of the flow container.

```
<flow height="200" left="16" top="16" width="1000">
    <button buttontype="push" height="126" name="1" width="126"/>
    <button buttontype="push" height="126" name="2" width="126"/>
    <button buttontype="push" height="126" name="3" width="126"/>
    <button buttontype="push" height="126" name="4" width="126"/>
    <button buttontype="push" height="126" name="5" width="126"/>
    <button buttontype="push" height="126" name="6" width="126"/>
</flow>
```

popup

Creates a button that, when clicked, displays a balloon dialog containing the component. Popup groups may be defined. Only one popup from each group is displayed at a time, however popups from different groups may be displayed simultaneously.

Note: *It is an error to put more than 1 component tag under a popup tag.*

Syntax

```
<popup container attributes>
       <component child component attributes> </component>
</popup>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified to the <popup> tag:

Attribute	Values	Restrictions	Description
name	String		The name to display on the button to trigger the popup.
group	String		The group attribute is used to define different groups that can be open at the same time. If this attribute is not defined, the popup is not a part of any group.

Child Component Attributes

In addition to General Attributes, the following attributes must be specified to child components:

Attribute	Values	Restrictions	Description
width	Integer	Required	Width of the container inside the popup
height	Integer	Required	Height of the container inside the popup

Example

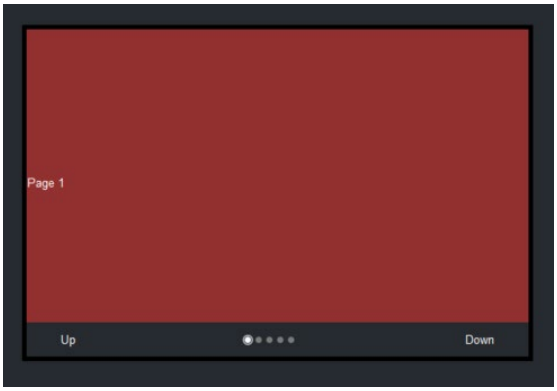
The following example creates a popup triggered by a button labelled “Selector”. The popup contains an abs container with 4 buttons placed in a 2x2 grid.

```
<popup name="Selector">
    <abs height="100" width="100">
        <button left="0" top="0" width="50" height="50" name="1"/>
        <button left="50" top="0" width="50" height="50" name="2"/>
        <button left="0" top="50" width="50" height="50" name="3"/>
        <button left="50" top="50" width="50" height="50" name="4"/>
    </abs>
</popup>
```

Figure 72 – Popup

pager

Creates a pager control component that is customizable using script. It is not currently available in the GUI. The pager control is built into an <abs> absolute container, and the abs container attributes can be used.



Syntax

For more information on creating a pager control using scripting, see the ***DashBoard User Guide***.

Container Attributes

In addition to General Attributes, the following attributes may be specified to the `<pagercontrol>` tag:

Attribute	Values	Restrictions	Description
virtualwidth	Integer		Defines a virtual width and height to use for all coordinates inside of the container. All offsets and dimensions inside of the container are scaled based on current width/height vs. virtualwidth/virtualheight. When these attributes are used, the UI will scale as the container size changes.
virtualheight	Integer		

Example

The following example creates a horizontal pager control.

```
<abs contexttype="opengear" id="_top" keepalive="true">
   <pager height="224" left="13"
style="look:round;bg#923030;bdr:thick;bdr#000000;" top="13" width="567">
      <config key="w.orientation">horizontal</config>
      <config key="w.model">var model = {
   currentPage: 1,

   getNumPages: function()
   {
      return 5;
   },
   getCurrentPage: function()
   {
      return this.currentPage;
   },
   scrollToPage: function(pageNum)
   {
      this.currentPage = pageNum;
      ogscript.reveal('page-' + pageNum);
   }

}
;
model</config>
      <tab tabposition="none">
         <abs id="page-0"/>
         <abs id="page-1">
            <label height="58" left="143" name="asdasdfasdf" style="txt-
align:west" top="58" width="161"/>
         </abs>
         <abs id="page-2">
            <button buttontype="push" height="65" left="185"
name="asdfasdfas" top="50" width="182"/>
         </abs>
         <abs id="page-3"/>
         <abs id="page-4"/>
         <abs id="page-5"/>
      </tab>
```

```
    </pager>
</abs>
```

simplegrid

Creates a grid of fixed-sized cells. All cells in a **simplegrid** control are of the same size. Child components are laid out left-to-right, top-down and are sized to fill the cell. If more control over layout is required, the table container should be used instead.

Syntax

```
<simplegrid container attributes>
    <component component attributes> </component>
    <component component attributes> </component>
    <component component attributes> </component>
    . . .
</simplegrid>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified:

Attribute	Values	Restrictions	Description
rows	Integer		Specifies the number of rows in the grid
cols	Integer		Specified the number of columns in the grid

Child Component Attributes

See General Attributes. There are no additional attributes for child components.

Example

The following example creates a 2 row x 3 column grid, with buttons 1, 2, 3 on the top row and buttons 4, 5, 6 on the bottom row. Each cell is 100x100 pixels.

```
<simplegrid left="16" top="16" height="200" width="300" rows="2"
cols="3">
    <button buttontype="push" name="1"/>
    <button buttontype="push" name="2"/>
    <button buttontype="push" name="3"/>
    <button buttontype="push" name="4"/>
    <button buttontype="push" name="5"/>
    <button buttontype="push" name="6"/>
</simplegrid>
```

split

Creates a split screen with exactly two components. The split is either horizontal (with a left component and a right component, separated by a vertical split bar) or vertical (with a top component and a bottom component separated by a horizontal split bar). If only one component is defined under the split tag, the split is removed and the single component is returned.

Note: *It is an error to put more than 2 component tags under a split tag.*

Syntax

```
<split container attributes>
       <component child component attributes> </component>
       <component child component attributes> </component>
</split>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified to the `<split>` tag:

Attribute	Values	Restrictions	Description
orientation	**horizontal**		The first component will be on the left and the second component will be on the right.
	vertical		The first component will be on the top and the second component will be on the bottom.

Default values shown in **bold**.

Child Component Attributes

In addition to General Attributes, the following attributes may be specified to child components:

Attribute	Values	Restrictions	Description
weight	Double value between +0.0 and 1.0		Specifies how much of the screen should be devoted to each side of the split. If the weight is defined for both components, the split is determined by weight / total weight.
minw	Positive integer		The minimum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
minh	Positive integer		The minimum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
maxw	Positive integer		The maximum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
maxh	Positive integer		The maximum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.

Example

The following example creates a split container with a horizontal split:

- Left side contains an abs container with 4 buttons placed in a 2x2 grid.
- Right side contains an abs container with 4 buttons placed in a 2x2 grid.

```
<split height="150" width="300" orientation="horizontal">
   <abs weight="0.5" height="100" width="100">
      <button left="5" top="5" width="25" height="25" name="1"/>
      <button left="30" top="5" width="25" height="25" name="2"/>
      <button left="5" top="30" width="25" height="25" name="3"/>
      <button left="30" top="30" width="25" height="25" name="4"/>
   </abs>
   <abs weight="0.5" height="100" width="100">
      <button left="5" top="5" width="25" height="25" name="5"/>
      <button left="30" top="5" width="25" height="25" name="6"/>
      <button left="5" top="30" width="25" height="25" name="7"/>
      <button left="30" top="30" width="25" height="25" name="8"/>
```

```
    </abs>
</split>
```

tab

Creates a tab component where each child component within the **tab** tag represents a separate tab page inside of the tab component. Note that the **height** and **width** attributes of a tab component include the space occupied by the tab labels, not just the size of child components.

Syntax

```
<tab container attributes>
      <component for tab 1 child component attributes> </component>
      <component for tab 2 child component attributes> </component>
      . . .
</tab>
```

Container Attributes

In addition to General Attributes, the following attributes may be specified to the <tab> tag:

Attribute	Values	Restrictions	Description
tabposition	**north** east south west none	How the tabs are rendered within their quadrant is determined by the look and feel.	Specifies the placement of the tabs. 
	none		Tabs will be hidden and the visible component must be controlled through the “reveal” tag or the OGP REVEAL_ELEMENT trap.
tablayout	scroll **stack**		If there are more tabs than can fit in the horizontal space available, this controls whether there are multiple rows of tabs (“stack”) or if additional tabs are on the same row and accessible via scrolling (“scroll”).
tabheight	Integer	Tab will not resize below minimum size to render tab label text	Specifies the height of the tab label, in pixels. Note that the width of the tab is determined by the length of the tab label names.
onchange	ogScript String		The provided snippet of ogScript is triggered when the selected tab changes. Current tab index is: **this.getSelectedIndex()** Current tab name is: **this.getTitleAt(this.getSelectedIndex())**

Default values shown in **bold**.

Child Component Attributes

In addition to General Attributes, the following attributes may be specified to child components:

Attribute	Values	Restrictions	Description
name	String	This attribute is used in elements contained within a tab tag.	Specifies (or overrides) the name to display in the **tab** for a component. If the component provides its own name (e.g. an OGP Menu), that name will be used in the absence of this attribute.
selected	default forced **none**	This attribute is used in elements contained within a tab tag.	default = this tab will be selected by default when the UI is loaded. forced = this tab will be selected by default when the UI is loaded and, if the UI is refreshed, this tab will be selected again. none = when the UI is loaded, the first tab in the tab group is selected. If the UI is refreshed, DashBoard should attempt to maintain the current tab selection.

Default values shown in **bold**.

Example

The following example creates a tab container with three tabs:

- First tab contains an abs container with 4 buttons placed in a 2x2 grid.
- Second tab contains an abs container with 4 buttons placed in a 2x2 grid.
- Third tab contains a single button

```
<tab width="250" height="300" left="1" tabposition="north" top="1">
    <abs height="300" width="250" name="First Tab">
        <button left="5" top="5" width="25" height="25" name="1"/>
        <button left="30" top="5" width="25" height="25" name="2"/>
        <button left="5" top="30" width="25" height="25" name="3"/>
        <button left="30" top="30" width="25" height="25" name="4"/>
    </abs>
    <abs height="250" width="250" name="Second Tab">
        <button left="5" top="5" width="25" height="25" name="5"/>
        <button left="30" top="5" width="25" height="25" name="6"/>
        <button left="5" top="30" width="25" height="25" name="7"/>
        <button left="30" top="30" width="25" height="25" name="8"/>
    </abs>
    <button name="Go"/>
</tab>
```

table

A table is a grid of rows and columns. A cell in the table can span any number of rows or columns. Each cell in a table contains a component defined in a child tag. Similar to HTML, each row of cells in a table must be encapsulated in a `tr` tag. Each element inside of a `tr` tag defines a component to be placed inside a cell. For simple grids with fixed-sized cells, the simplegrid container may be used instead.

Syntax

```
<table container attributes>
    <tr>
        <component child component attributes> </component>
        <component child component attributes> </component>
        . . .
    </tr>
    <tr>
        <component child component attributes> </component>
        <component child component attributes> </component>
        . . .
    </tr>
    . . .
</table>
```

Child Tags

Tag	Values	Restrictions	Description
<tr>			Encapsulates a row.
<*component*>	Any valid component tag		Defines the component for a table cell. Must be a child of a **tr** tag.

Container Attributes

See General Attributes.

Child Component Attributes

The following set of attributes controls the layout of cells and components. To control the appearance of the table contents, these additional attributes should be defined in the child tags that define the content of the table cells.

Attribute	Values	Restrictions	Description
fill			Controls how the component inside of a table cell fills the cell itself.
	none		Uses the component's natural width and height and floats it inside of the cell.
	Horizontal		Uses the component's natural height but fills the horizontal space.
	Vertical		Uses the component's natural width but fills the vertical space.
	both		Ignores the component's natural width and fills the entire cell.

Attribute	Values	Restrictions	Description
anchor	**center** north northeast east southeast south southwest west northwest		If the **fill** is set to anything other than `both`, this controls where the component is attached to the cell. 
rowspan	Positive integer	Cells must not collide	The number of rows spanned by a cell.
colspan	Positive integer	Cells must not collide	The number of columns spanned by a cell.
insets	4 positive integers separated by commas. e.g. “5,5,5,5”		Specifies padding around the component. The 4 numbers represent the top, left, bottom, and right padding. The insets are specified in pixels.
weightx	Double value between +0.0 and 1.0		Specifies how to distribute extra horizontal space. The table calculates the weight of a column to be the maximum **weightx** of all the components in a column. If the resulting layout is smaller horizontally than the area it needs to fill, the extra space is distributed to each column in proportion to its weight. A column that has a weight of zero receives no extra space. If all the weights are zero, all the extra space appears between the grids of the cell and the left and right edges.
weighty	Double value between +0.0 and 1.0		Specifies how to distribute extra vertical space. The table calculates the weight of a row to be the maximum **weighty** of all the components in a row. If the resulting layout is smaller vertically than the area it needs to fill, the extra space is distributed to each row in proportion to its weight. A row that has a weight of zero receives no extra space. If all the weights are zero, all the extra space appears between the grids of the cell and the top and bottom edges.

Attribute	Values	Restrictions	Description
orientation	**horizontal** vertical	Only applies to element tags that return multiple components.	If a tag returns multiple components (e.g. a param tag for an array parameter), this specifies whether the returned components should be in the same row (horizontal) or in the same column (vertical).
minw	Positive integer		The minimum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
minh	Positive integer		The minimum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
maxw	Positive integer		The maximum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
maxh	Positive integer		The maximum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.
placeholders	Positive integer **Default 0**		This tag specifies the minimum number of elements which are expected to be returned by a tag. If a tag returns fewer than the specified number of elements, placeholder elements are created and added to the layout in their place. A value of `0` means that the tag is ignored if no elements were returned (or the tag is undefined).
maxperrow	Positive integer **Default -1**		A value > 0 defines the maximum number of elements in a row. Additional elements will be placed on the next row.

Default values shown in **bold**.

Note: *DashBoard uses a Java Swing GridBagLayout internally. For more information about GridBagLayout, please see* *http://docs.oracle.com/javase/8/docs/api/java/awt/GridBagLayout.html*

Example

The following sample utilizes a table to create a numeric keypad.

```
<table height="300" left="16" top="16" width="300">
    <tr>
        <button buttontype="push" fill="both" name="1"> </button>
        <button buttontype="push" fill="both" name="2"> </button>
        <button buttontype="push" fill="both" name="3"> </button>
    </tr>
    <tr>
        <button buttontype="push" fill="both" name="4"> </button>
        <button buttontype="push" fill="both" name="5"> </button>
```

```
        <button buttontype="push" fill="both" name="6"> </button>
    </tr>
    <tr>
        <button buttontype="push" fill="both" name="7"> </button>
        <button buttontype="push" fill="both" name="8"> </button>
        <button buttontype="push" fill="both" name="9"> </button>
    </tr>
    <tr>
        <button buttontype="push" fill="both" name="*"> </button>
        <button buttontype="push" fill="both" name="0"> </button>
        <button buttontype="push" fill="both" name="#"> </button>
    </tr>
</table>
```

Top Level Attributes

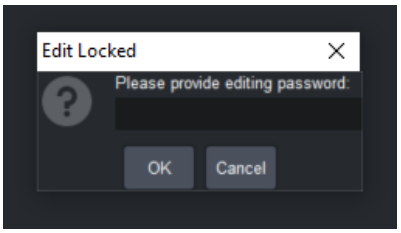
Top level attributes can be added to any of the container tags listed in the Layout/Container Tags, but only if that container is the uppermost container in the source code. The source code can be found when you enter **PanelBuilder Edit Mode**, and double-click on an empty spot of the canvas. This opens the **Component Editor**, with the uppermost container selected in the tree view (typically an **<abs>** container in a new CustomPanel file). You can add the top level attributes in the uppermost container.

The following top level attributes are supported:

Attribute	Description
editlock	Allows a panel to be protected with a user-defined defined password. When a user tries to enter Panel Builder Edit Mode, the password prompt will appear requesting the credentials.
encrypt	Encrypts a panel to protect the source code.
gridsize	Allows you to snap components to the grid backdrop when in PanelBuilder Edit Mode. You can organize and automatically line up components on the screen along the provided horizontal and vertical grid lines.
keepalive	When set to true, this flag prevents panels from being unloaded by the memory manager. When set to **false**, if this panel is inactive and DashBoard runs low on memory it can be unloaded.

editlock

Defines the password that will protect a panel from tampering. The editlock value can be any user-defined string. When editlock is set, a password popup will appear when a user attempts to enter PanelBuilder **Edit Mode**.



To set an **editlock** password, enter PanelBuilder Edit mode, and double-click on an empty area of the canvas. The **Component Editor** will open with the uppermost <abs> selected in the tree view. Click the Source tab, and edit the top line of code to include editlock = "<password>".

Warning: Anyone can open the panel .grid file in any text editor and view the password, unless your panel is encrypted. For more details on how to encryp your panel see, encrypt.

You can see an example below with the password set to `RossVideo12345`.

Syntax

```
<abs contexttype="opengear" id="_top" editlock="RossVideo12345" style="">
</abs>
```

encrypt

This allows you to encrypt a panel so that the source code cannot be viewed. It is recommended to encrypt passwords when using the **editlock** attribute. To encrypt a panel so that the source code cannot be viewed, add encrypt= "SimpleEncrypt" to the uppermost <abs>. The **editlock** tag *must* be set to "SimpleEncrypt".

Warning: make sure to use the correct capitalization, as setting encrypt to any other value may break your panel!

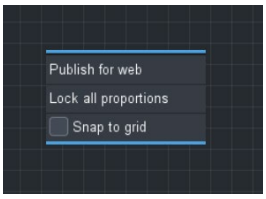
You can see an example below using the **encrypt** tag.

Syntax

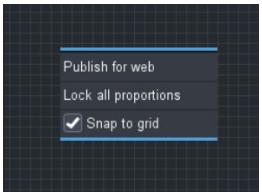
```
<abs contexttype="opengear" id="_top" encrypt="SimpleEncrypt" style="">
</abs>
```

gridsize

This **Snap to Grid** feature allows you to snap components to the grid backdrop when in PanelBuilder Edit Mode. You can organize and automatically snap components to the nearest horizontal and vertical grid lines. With the CustomPanel open, enter PanelBuilder Edit Mode. When adding a new component to the canvas, or resizing an existing component, it will auto-fill to encompass the closest grid space. To enable **Snap to Grid**, right-click on the blank canvas and select **Snap to grid**.



The default grid size is set to 20. You can adjust the size of the grid in the source code to make the grid larger or smaller, as shown below:



You can see an example below using the **gridsize** tag.

Syntax

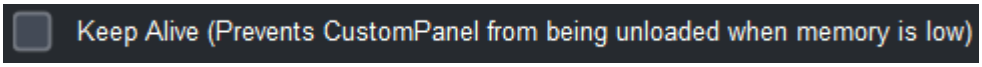
```
<abs contexttype="opengear" id="_top" gridsize="10" style="">
</abs>
```

keepalive

When DashBoard runs low on memory, it may unload panels that are not active, in order to free up memory. If you have a panel that runs tasks in the background (listeners, gpi triggers, timers, etc), you may not want DashBoard to unload your panel. You can use the **keepalive** flag in the top-level container>, to indicate that this panel should not be unloaded. For more details, see the Memory Manager feature in the ***Dashboard User Guide***.

Note: Panels without this flag cannot be unloaded.

Tip: From the PanelBuilder Component Editor, in the **Abs Attributes** tab, you can select the **Keep Alive** checkbox to ensure panel is not unloaded.



Syntax

```
<abs contexttype="opengear" id="_top" keepalive="true" style="">
</abs>
```

Widget Tags

Widget tags are components that can be added to an OGLML page. In contrast with the Container tags described previously, widget tags do not contain other components.

The following tags are provided:

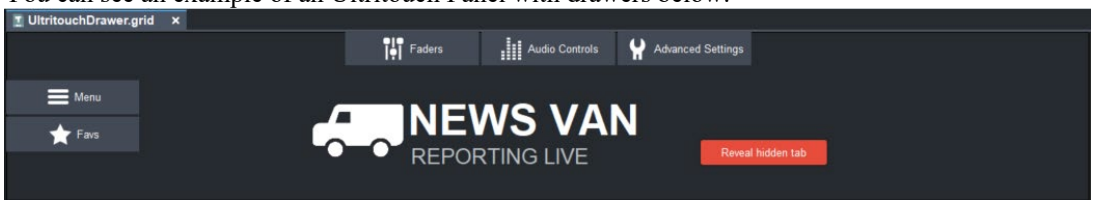
Tag	Description
reveal	Brings hidden tab pages to the front
drawer	Creates a container that allows you to add drawer tabs to maximize panel space, by organizing additional content in hidden drawer tabs.
ext	Opens the editor for a specified node in the DashBoard Tree.
exit	Creates an exit button that, when clicked, causes DashBoard to close the current panel, window, or application.
help	Creates a help pop-up button which can display a custom help title and

Tag	Description
	message when selected.
image	Displays a static image
label	Creates a static text label
button	Creates a button
browser	Creates a web browser window
blank	Placeholder, used to leave a blank cell in simplegrid, table and flow containers
lock	Allows DashBoard client screen to be locked
memory	The memory manager widget allows you to add a memory status indicator bar to monitor the current memory usage of the DashBoard application.
widget	Creates an instance of a custom widget
wizard	You can create a basic wizard, that is also customizable using script.

drawer

If space is limited on your custom panel, you can now create drawers to make additional space for content. This is ideal for smaller panels with restricted space, such as the Ultritouch custom panel, or any panel that is crowded with too many components. It can help to organize your content, compartmentalize standalone functions, or to minimize certain parts of the custom panel when it is not in use.

You can see an example of an Ultritouch Panel with drawers below:



It is recommended that you create the drawer using the **Tab Split & Drawer** button that can be found in Panel Builder **Edit Mode** toolbar.

For more information, see the ***DashBoard User Guide > Panel Builder > Adding basic Components > Drawers***.

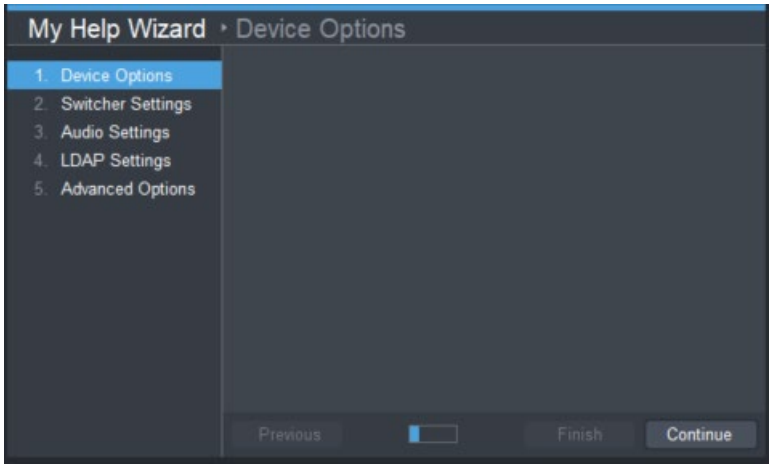
Syntax

```
<drawer name="drawer-name", targetid="element1,element2,...">
   <abs> name= <abs>
<drawer height="380" left="163" tabfill="both" top="141" width="538">
      <abs anchor="drawer-tab-name" height="48" id="north1" name="north1"
width="249"/>
   </drawer>
<drawer height="380" left="163" tabfill="none" top="141" width="538">
   <abs anchor="east" height="48" id="north1" name="north1" width="249"/>
</drawer>
```

wizard

You can create wizards that contain a title, a page navigation pane, and a progress bar. The wizard allows you to automate complex tasks and break them into a series of steps that walk users through the process from start to finish.

In addition to determining which features you would like to be visible, you can also choose how many pages appear in the wizard. Pages are shown as Page 1, Page 2, and so on. It's easy to change the default page name to be more descriptive, since the navigation pane already provides automatic numbering on each tab. For example, "Page 1" could be renamed "Device Options" and that tab will display "1. Device Options" in the navigation pane, as shown below.



reveal

Creates a button that, when clicked, causes elements within tab pages to become visible in the UI. When the button is pressed, DashBoard finds all components with the provided target ID(s) and checks to see if they are contained within a tab component or **menugroup**. If component is found, its tab page is brought to the foreground (made the active tab). If the specified component is buried deep within the UI (e.g. a tab within a tab), the device must supply multiple IDs to 'reveal' the desired component and the component's parents.

Syntax

```
<reveal name="button-name", targetid="element1,element2,...">
</reveal>
```

Attributes

In addition to General Attributes, the following attributes may be specified to the <reveal> tag:

Attribute	Values	Restrictions	Description
name	String		The name to display on the button.
targetid	list of Strings separated by commas or semicolons.	Each string in the value must refer to the **id** of another component.	Specifies the element ids to show.

Examples

The following example creates a button that reveals the menu with id “Key1Panel”

```
<reveal name="Key 1", targetid="Key1Panel"/>
```

The following example creates a button that reveals the menu “Key1Panel” and the tab “Key1ChromaTab”.

```
<reveal name="Chroma Key 1", targetid="Key1Panel,Key1ChromaTab"/>
```

ext

Creates a button that, when clicked, causes DashBoard to open an editor tab for a device in the DashBoard tree view. When the button is pressed, DashBoard searches its tree view for a node with the provided ID. If a node is found and the node contains an editor, its editor is opened and/or brought to the foreground (made the active tab).

If a component is buried deep within the UI (e.g. a tab within a tab), the card can supply multiple IDs to ‘reveal’ not only the desired component, but also the component’s parents.

Syntax

```
<ext name="button-name" objectid="node-id" buttontype="type" general attributes/>
<ext name="button-name" objectid="FileNavigator,file-path,file-name" buttontype="type" general attributes/>
```

Attributes

In addition to General Attributes, the following attributes may be specified to the <ext> tag:

Attribute	Values	Restrictions	Description
name	String		The name to display on the button.
objectid	String	The value must refer to the node ID an element in DashBoard’s tree view.	Specifies the id of the components to show. DashBoard provides a few shortcuts to reference elements under the device node in the tree or a sibling device in the same frame: %frame% will be replaced with the frame’s primary identifier. %device% will be replaced with the device’s primary identifier. %slot 1 (or 2, or 3, etc.)% will be replaced with the primary identifier of the device in the referenced slot in the same

Attribute	Values	Restrictions	Description
			frame. If the String starts with "`FileNavigator,`" the objectid specifies a path and filename of a resource in the DashBoard file navigator, rather than the tree view.
buttontype	**button** label	Optional	"button" = display the link as a button "label" = display the link as a label

Default values shown in **bold**.

exit

Creates an exit button that, when clicked, causes DashBoard to close the current panel, window, or application. If a message prompt is defined, then a **Yes** or **No** message prompt pop-up will appear when the button is pressed. An example of an exit button with a prompt set would be: prompt="Do you wish to exit this panel?".

If you set the exit button to close a panel when pressed, additional options are available to set DashBoard to jump to another device user interface from the tree view, or Custom Panel file. When the button is pressed, DashBoard searches its tree view for a node with the provided ID. If a node is found and the node contains an editor, its editor is opened and/or brought to the foreground (made the active tab).

Syntax

```
<exit name="button-name" level= "panel|window|application" openobjectid=
"node-id" prompt="Exit-prompt-message" general attributes/>
<exit name="button-name" level= "panel" openobjectid="FileNavigator,file-
path,file-name" buttontype="type" prompt="Exit-prompt-message" general
attributes/>
```

Attributes

In addition to General Attributes, the following attributes may be specified to the `<ext>` tag:

Attribute	Values	Restrictions	Description
name	String		The name to display on the button.
level	String		The level can be set to one of the following: `panel`, `window` or `application`. Where: • `panel` closes the current panel. • `window` closes the current window. • `Application` closes the DashBoard application. Note: Setting `level` to `window` can also result in exiting the DashBoard application if only one window is open when the button is pressed.
objectid	String	The value must refer to the node ID an element in DashBoard's tree	Specifies the id of the components to show. DashBoard provides a few shortcuts to reference elements under the device

Attribute	Values	Restrictions	Description
		view.	node in the tree or a sibling device in the same frame: %frame% will be replaced with the frame's primary identifier. %device% will be replaced with the device's primary identifier. %slot 1 (or 2, or 3, etc.)% will be replaced with the primary identifier of the device in the referenced slot in the same frame. If the String starts with "`FileNavigator,`" the objectid specifies a path and filename of a resource in the DashBoard file navigator, rather than the tree view.
prompt			If you want a message prompt to appear to confirm whether to exit the panel, application or window, add the following: `prompt="your-message-text"` When defined, the message you have entered appears in a **Yes** or **No** pop-up dialog.
buttontype	**button** label	Optional	"button" = display as a button "label" = display as a clickable label

Default values shown in **bold**.

help

Creates a help pop-up button which can display a custom help title and message when selected.

Syntax

```
<help control attributes>
      <![CDATA[ <html>Html text</html>]]>
</help>
```

Control Attributes

In addition to General Attributes, the following attributes may be specified to the `<help>` tag:

Attribute	Values	Restrictions	Description
popupwidth	Integer		Specifies the width of the popup content, in pixels.
popupheight	Integer		Specifies the height of the popup content, in pixels. This does not include the title.
Title	String		The title to display in the popup.
Message	String	Can be plain text or html.	The message to display in the popup.

Default values shown in **bold**.

Examples

The example code below creates a 40 by 40 pixel help pop-up, as shown in *Figure 70*.

```
<help height="40" left="25" top="25" width="40" popupheight="200"
popupwidth="500" style="bg#ff0000;" title="Example Help">
<![CDATA[<html><left><b><u>Html formatted heading</u></b><br><font
color=#ffffdd>Take me Home</font><br><a
href="https://www.rossvideo.com/">Ross Video</a><br>
The latest software release for Carbonite Black Solo unlocks a powerful
USB Media Player functionality and is available to customers at no
additional cost. This new media player provides the functionality of a
single-channel clip player, for playout of compressed MPEG-4 AVC media
directly from a connected USB-media drive. There is no other production
switcher in the world with this level of built-in media playback.
</html>]]>
</help>
```

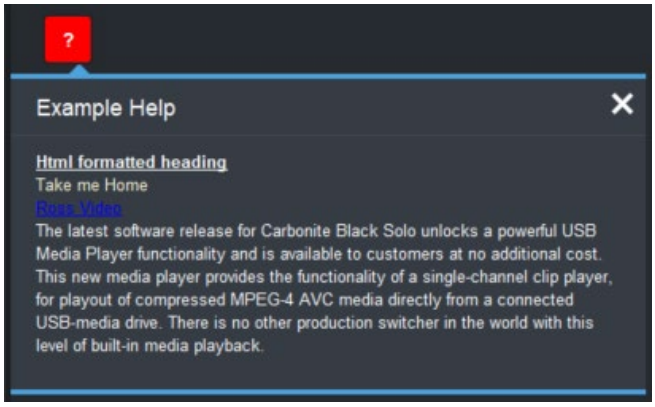


Figure 73 – Example Help Dialog

image

Fetch an image from the provided URL and display it.

Syntax

```
<image src="URL-String" attributes> </image>
```

Attributes

Attribute	Values	Restrictions	Description
src	*URL String*	Required. Must be a fully qualified URL.	Set the background image of the component.
height top bottom	*Integers*		It **top** and **bottom** are both specified, or **height** is specified, the image will be stretched to the height specified. Otherwise, the image's native height is used.

Attribute	Values	Restrictions	Description
width left right	*Integers*		It **left** and **right** are both specified, or **width** is specified, the image will be stretched to the width specified. Otherwise, the image's native width is used.

Examples

The following example places an image at its native size:

```
<image src="http://whatever.com/logo.jpg" top="50" left="50"/>
```

The following example places an image and scales it to 200x100 pixels in size.

```
<image src=" http://whatever.com/logo.jpg" top="50" left="50"
height="100" width="200"/>
```

label

Display a label. If the name is not defined, the text content of the label is used to provide the content. One or more ogScript tasks can be attached to a label to be fired when the label is clicked.

Syntax

```
<label name="label-name" attributes> </label>
```

Attributes

Tag	Values	Restrictions	Description
name	String		The text to display in the label.
align	left right center		The horizontal alignment of the text within the label.
header	true **false**		Format the label as a header element (apply a standard header background, foreground, and border).
html	true **false**		The text is actually a snippet of HTML (you do not need to provide the <html></html> tags).

Default values shown in **bold**.

Examples

```
<label name="This is a label" />
<label html="true" name="This is an &lt;i&gt;HTML&lt;/i&gt; label"/>
<label header="true" name="Label with the header attribute" />
```

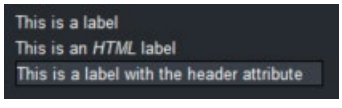


Figure 74 – Label Examples

button

Display a button. One or more ogScript tasks can be attached to a button to be fired when the button is pressed or toggled.

Syntax

```
<button name="label-name" attributes> </button>
```

Attributes

Attribute	Values	Restrictions	Description
name	*String*	Required	The text to display in the label.
buttontype	**push** toggle checkbox radio		The type of button to create. Push buttons are stateless. Toggle, checkbox, and radio are all 2-sate “on” and “off”.
toggled	true **false**		The initial state of the button.
flat	true **false**	Only applicable to **push** or **toggle buttontype**	Request a ‘flat’ look for the button (or toggle button) in the UI. Note that icons styles may not be applied to flat buttons.

Default values shown in **bold**.

Examples

This example displays a series of simple pushbuttons as illustrated in Figure 73:

```
<button buttontype="push" name="push" top="25" width="80"/>
<button buttontype="toggle" name="toggle" toggled="true" top="25"
width="80"/>
<button buttontype="push" flat="true" left="400" name="flat" top="25"
width="80"/>
<button buttontype="radio" left="500" name="radio" top="25"/>
<button buttontype="checkbox" left="600" name="checkbox" top="25"/>
```

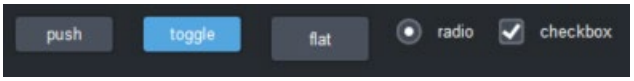


Figure 75 – Button Examples

browser

Embed a web browser component in the page and point it at the specified URL.

Note *The browser plug-in is a heavy widget, and should therefore be used sparingly.*

Syntax

```
<browser url="URL-String" height="height" width="width" attributes >
</browser>
```

Attributes

Attribute	Values	Restrictions	Description
url	URL String	Required. Must be a fully qualified URL.	The URL to use for the provided browser.
width	Positive integer	Required	The width (in pixels) of the browser
height	Positive integer	Required	The height (in pixels) of the browser

Notes

This is a native browser component provided by the OS.

- On Windows it will generally be Internet Explorer.
- On Linux it will either be WebKit GTK or XUL Runner.
- The browser is currently disabled on OS X pending a bug fix within the Eclipse platform.

The web sites pointed to by the browser must NOT contain Java Applets.

Not all plug-ins will be available on all browsers. It is recommended that developers test their web pages inside of DashBoard on multiple platforms.

The browser is a heavyweight component and must not be used inside of a scrolling component.

The browser will cause rendering issues if it is clipped by other components.

Example

```
<browser url="http://www.rossvideo.com" height="400" width="1200"/>
```

blank

Creates a blank placeholder component. This can be used to fill space where necessary.

Syntax

```
<blank attributes />
```

Attributes

See General Attributes.

Example

```
<table left="25" top="25" width="400">
    <tr>
        <label name="This" width="100"/>
        <label name="is" width="100"/>
        <label name="a" width="100"/>
        <label name="table" width="100"/>
    </tr>
    <tr>
        <label name="with" width="100"/>
        <blank/>
        <label name="blank" width="100"/>
        <label name="tags" width="100"/>
    </tr>
</table>
```

Figure 76 – Blank Tag Example

lock

Creates a button that, when pressed, will turn on DashBoard's screen lock. The lock button will display a lock icon by default but this icon can be overridden by a card developer.

Syntax

```
<lock name="button-name" attributes />
```

Attributes

See General Attributes.

Attributes	Values	Restrictions	Description
name	*String*		Text to display on the button. Text will be rendered beside the lock icon.

Example

```
<lock name="Lock Screen" left="25" top="25"/>
```

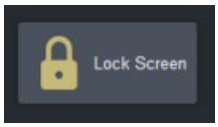


Figure 77 – Lock Button

When locked, the DashBoard UI will be darkened, with an unlock widget.

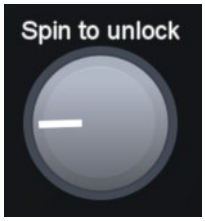


Figure 78 – Lock Screen Widget

memory

The memory manager widget allows you to add a memory status indicator bar to monitor the current memory usage of the DashBoard application. This performs the same function as the memory manager indicator that is available in the top right DashBoard toolbar. The memory manager widget allows you to continue to monitor the memory usage of the status indicator while a panel is in full screen mode. You can add a memory manager widget directly to your panel and customize its size and position. By default the **<memory/>** tag is 60 pixels in width by 20 pixels in height, and it is located in the top left corner.

Syntax

```
<abs contexttype="opengear" id="_top" keepalive="false" style="">
   <memory height="50" left="1500" top="50" width="200"/>
</abs>
```

A memory manager widget appears in the specified area.

Figure 79 – Lock Screen Widget

Attributes	Values	Restrictions	Description
id	*String*		Widget identifier.
height	*String*		Height of the memory manager widget.
width	*String*		Width of the memory manager widget.
left	*String*		Offsets the memory manager widget a select number of pixels from the left side margins of the panel.
right	*String*		Offsets the memory manager widget a select number of pixels from the right side margins of the panel.
top	*String*		Offsets the memory manager widget a select number of pixels from the top margins of the panel.
bottom	*String*		Offsets the memory manager widget a select number of pixels from the bottom margins of the panel.

widget

Creates an instance of a custom widget. The widget must be defined through a `widgetdescriptor` tag. Parameters declared within the `widgetdescriptor`'s `config block` may be overridden through `param` tags within a `config` block.

Syntax

```
<widget widgetid="widget-id" baseOID="base-oid">
   <config>
      <params>
         <param/>
         <param/>
         . . .
      <params/>
   </config>
</widget>
```

Attributes

In addition to General Attributes, the following attributes may be specified to the `<widget>` tag:

Attributes	Values	Restrictions	Description
widgetid	*String*	Must match the **id** of a declared widgetdescriptor.	Widget identifier.
baseOID	*String*		Specifies the base OID string for relative parameter access. Relative parameter access within the widget will be prefixed with the value of the **baseOID** attribute string.

See Also

- widgetdescriptor
- config
- param

Examples

The following example displays a custom widget with id `alarmgrid`:

```
<widget widgetid="alarmgrid" top="100" left="100"/>
```

The following example displays a custom widget with id `alarmgrid`, overriding the value of parameter `str2` with the value "New String Value":

```
<widget left="100" top="300" widgetid="alarmgrid">
    <config>
        <params>
            <param oid="str2" value="New String Value"/>
        </params>
    </config>
</widget>
```

Non-UI Tags

The following tags do not provide any UI elements themselves. They contribute new parameters, script snippets, constraints, etc. for use elsewhere in the OGLML document.

The following tags are provided.

Tag	Description
api	Provides a location for global ogScript code.
context (device context)	A device context is a data structure that contains information about the attributes of a device data source.
meta	This is a convenient parent tag for all non-UI tags.
subscription	This tag indicates the list of subscription oids that the panel wishes to receive parameter updates from the OGP JSON device. **Note**: The device source must support subscriptions protocol.
widgets	This is a container for user-defined widget descriptors.
widgetdesriptor	Defines a custom widget.
lookup	A lookup defines constants to be substituted inside of other tag attributes or used in ogScript blocks.
style	To provide something similar to Cascading Style Sheets (CSS) available in HTML, styles can be defined in a tag and referenced in the style attribute of widget tags.
color	Defines or overrides a color constant for use within style hints.
ogscript	Defines an ogScript code snippet to handle an event on a UI element or parameter.
constraint	Defines the constraint of a parameter.

Tag	Description
params	The parent container for parameters defined within the OGLML document.
timer	The timer tag fires events at regular intervals.
listener	The listener tag allows an OGLML page to process network communications using protocols not already available.
task	Defines a block of ogScript to be run when an event happens in the system.
timertask	Defines a block of ogScript to be run when a timer goes off.
include	This tag allows an OGLML document to be assembled from several individual XML files or fragments.

api

Provides a location for global ogScript code. Contents of the `<api>` tag are processed by the ogScript compiler directly. Elements within an `api` tag are scoped where they are declared in the XML; siblings and children of siblings have visibility to elements declared within the `api` tag.

You can use the `<api>` tag to create a library of reusable ogScript code segments. For more information and best practices, see Custom APIs Within CustomPanels.

The `api` tag should generally be placed within a `<meta>` tag for global ogScript code encapsulation. However, ogScript code intended to dynamically generate and modify the XML should be placed in a top-level `api` tag.

Syntax

```
<api>
    global-scope elements
</api>
```

Attributes

None.

context (device context)

A device context is a data structure that contains information about the attributes of a device. It provides a means to organize the OGLML document structure of the DashBoard CustomPanel. Typically this tag is used if a CustomPanel (also called a device panel elsewhere) is used to add more than one data source to the panel.

Basic Syntax

```
<context contexttype="opengear" objectid="Daves_Ultritouch...">
</context>
```

Syntax for Panels that Support Subscriptions

```
<context contexttype="opengear" objectid="DeviceID..."
subscriptions="true">
```

```
    <meta>
       <subscription oids="oid1, oid2, oid3*"/>
    </meta>
</context>
```

Example of a Subscriptions Panel with Two Device Contexts

```
<abs contexttype="opengear" id="_top" keepalive="false" objectid="MyUltritouch..."
objecttype="Ultritouch Device">
   <context contexttype="opengear" objectid="Kyles_Ultritouch..."
subscriptions="true">
      <meta>
         <subscription oids="db.touch*,deviceoptions.speakervolume"/>
      </meta>
    </context>
    <context contexttype="opengear" objectid="Daves_Ultritouch..."
subscriptions="true">
      <meta>
         <subscription oids ="devices*, deviceoptions.lcdbrightness"/>
      </meta>
    </context>
</abs>
```

In this example, you can see two separate device contexts, which point to two different Ross Ultritouch devices that support subscriptions protocol. The topmost container for the panel, in this case an <abs/> does not need to be modified to add device contexts.

Attributes

Attribute	Values	Restrictions	Description
contexttype	string		Typically set to **opengear** for openGear or DashBoard Connect devices.
objectid	string		Object ID provided by DashBoard.
subscriptions	String set to "true" or "false".	*This attribute must be set to "true" to support OGP devices that support the subscription protocol.	This flag is required to indicate support for subscriptions devices that are used as a data source in this panel. **Note**: The panel must also provide a list of **subscription OIDs** to determine which device parameters the panel will always receive updates for.

subscription

This tag indicates the list of subscription oids that the panel wishes to receive parameter updates from the OGP JSON device.

Note: The device source must support subscriptions protocol. This tag only works when used in conjunction with the **`subscriptions="true"`** attribute.

Syntax

```
<context contexttype="opengear" objectid="DeviceID..."
subscriptions="true">
   <meta>
      <subscription oids="oid1, oid2, oid3*"/>
   </meta>
</context>
```

Attributes

Attribute	Values	Restrictions	Description
oids	String of comma separated OIDs	*Required to support OGP devices that support the subscription protocol.	The list of OID parameters for the openGear device source must be listed here, otherwise the panel will only get updates for the minimal set of OIDs. **Note**: This tag can only be added to a CustomPanel that indicates support for **`subscriptions="true"`** in the context or top level attributes of the panel. It is recommended to nest the subscription oid list within a meta tag. See the details below for more information wildcards.

Note: You can use wildcard asterisks to include multiple OIDs simultaneously that have the same starting prefix in the name. The wildcard should be added after this prefix. These wildcards are useful when you don't want to type out a whole list of similar OIDs manually. Instead you can add a subset of OIDs by including a wildcard. If wildcards are used, your list of subscriptions are optimized by DashBoard to use the wildcard that includes the most items.

About Using Wildcards

Adding a wildcard asterisk to a list of parameter OIDs in a DashBoard device panel, will allow you to quickly add multiple sets of parameter OIDs that start with the same prefix. You can only add an asterisk to the end of an oid prefix name. The asterisk means that you will subscribe to all parameters that start with the prefix you entered.

For example, if you wanted to add three OIDs, types.audiomixer, types.audiomixerpartition and types.audiosound, you could use the following wildcards: ty*, types.audio*, or types.au*. If you use more than one wildcard that applies to the same parameters, DashBoard will choose the most efficient wildcard to optimize. In the example above, ty* would be used. You cannot add a wildcard before the prefix or have text after the wildcard. For example, *ypes. and ty*p are not valid.

For related content, see: context (device context), subscriptions, meta

meta

This is a convenient parent tag for all non-UI tags. The meta tag does not deepen the scope,

therefore children of the meta are considered at the same scope as the meta tag itself, and therefore siblings of other top-level tags.

Syntax

```
<meta>
    non-ui-tags
</meta>
```

Attributes

None.

widgets

This is a container for user-defined widget descriptors.

Syntax

```
<widgets>
    <widgetdescriptor/>
    <widgetdescriptor/>
    . . .
</widgets>
```

Attributes

None.

widgetdescriptor

Defines a custom widget. The widget descriptor contains two blocks denoted by `<config>` and `<oglml>` tags. The `config` section includes content to render the widget's configuration page within PanelBuilder's **Edit Component** dialog. The `oglml` block contains the content to create the widget itself.

The `widgetdescriptor` tag may be contained within a widgets block of an oglml document, in an external file or be served up via URL

Syntax

```
<widgetdescriptor id="widget-id" baseurl="URL-string"
structtype="structtype">
    <config/>
    <oglml/>
</widgetdescriptor>
```

Attributes

Attributes	Values	Restrictions	Description
id	*String*	Must be unique	Widget identifier.
structtype	*String*		Specifies a dependency of the widget upon a global struct parameter with matching **structtype**. Currently this type checking is restricted only to PanelBuilder UI; a custom widget will only be available in PanelBuilder if a parameter exists with matching **structtype**.
baseurl	*String*	Must be a valid, fully qualified URL.	When specified, the widget descriptor will be fetched from a document specified by the URL, rather than inline.

See Also

widget
config
param

Examples

The following creates a custom widget which displays four alarm dots in a 2x2 grid. The strings that sit beside each dot are configurable parameters of the widget.

```
<widgetdescriptor id="alarmgrid">
    <config>
        <params>
            <param access="1" type="STRING" oid="str1" name="String 1"/>
            <param access="1" type="STRING" oid="str2" name="String 2"/>
            <param access="1" type="STRING" oid="str3" name="String 3"/>
            <param access="1" type="STRING" oid="str4" name="String 4"/>
        </params>
    </config>
    <oglml>
        <simplegrid cols="2" rows="2">
            <param oid="str1" widget="12" width="200" height="40"/>
            <param oid="str2" widget="12" width="200" height="40"/>
            <param oid="str3" widget="12" width="200" height="40"/>
            <param oid="str4" widget="12" width="200" height="40"/>
        </simplegrid>
    </oglml>
</widgetdescriptor>
```

The following retrieves a widget descriptor from a web server:

```
<widgetdescriptor
baseurl="http://mydevice/files/widgets.widgetdescriptor"/>
```

The widget is then displayed with the following:

```
<widget widgetid="alarmgrid" top="100" left="100"/>
```

The following example displays the widget, overriding the value of parameter `str2` with the value "New String Value":

```
<widget left="100" top="300" widgetid="alarmgrid">
    <config>
        <params>
            <param oid="str2" value="New String Value"/>
        </params>
    </config>
</widget>
```

lookup

A lookup defines constants to be substituted inside of other tag attributes or used in ogScript blocks. Lookups contain "entry" tags to define key/value pairs. Constants defined in a parent context can be referenced in a child context. If a key from the parent context is re-defined in a child context, the re-defined value will take precedence in the child's scope.

Global `lookup` tags should usually be placed within an `api` tag.

Syntax

```
<lookup id="id-string" scope="scope">
    <entry key="key">value</entry>
    <entry key="key">value</entry>
    . . .
</lookup>
```

Attributes

Attribute	Values	Restrictions	Description
scope	private **public** window	If "private", the lookup must define the **id** attribute.	By default, all key/value pairs are added to a general lookup table. The lookup table in any context is the concatenation of all parent lookup tables and sibling lookup tables. If the scope is set to "private", the key/value pairs can only be referenced using the lookup table's ID.
id	string		If defined, key/value pairs for this lookup can be referenced in ogScript using "ogscript.getPrivateString('[id]', '[key]');" Or substitute inside of other attributes with %const['id']['key']%
code	true **false**		Must be set true if the lookup **value** contains executable script.
multiline	true **false**		Must be set true if lookup **value** contains multi-line strings.

Default values shown in **bold**.

Example

The following tag creates a public lookup

```
<lookup>
   <entry key="breakfast">Bacon and Eggs</entry>
   <entry key="lunch">BLT</entry>
   <entry key="dinner">Bacon explosion</entry>
   <entry key="snack">Bacon-maple donut</entry>
</lookup>
```

The following code returns the string `BLT`.

```
var currentMeal = ogscript.getString('lunch');
```

The following tag creates a private scope lookup

```
<lookup id="family" scope="private">
   <entry key="father">Homer Simpson</entry>
   <entry key="son">Bart Simpson</entry>
   <entry key="mother">Marge Bouvier-Simpson</entry>
   <entry key="daughter">Lisa</entry>
   <entry key="baby">Magaggie</entry>
</lookup>
```

The following code would return the string `Homer Simpson`.

```
var name = ogscript.getPrivateString('family', 'father');
```

The following tag creates a block of code lookup:

```
<lookup code="true" id="GlobalScripts" multiline="true">
   <entry key="UpdateTimer">
   if (params.getValue('Update_Automatically', 0) == 1)
   {

   ogscript.getTimerManager().getTimer('UpdateTimer').startTimer(false);
   }
   else
   {
      ogscript.getTimerManager().getTimer('UpdateTimer').stopTimer(false
   );
   }
   </entry>
</lookup>
```

The following is an example of instancing the code defined in the above lookup:

```
<ogscript
handles="onload">%const['GlobalScripts']['UpdateTimer']%</ogscript>
```

style

To provide something similar to Cascading Style Sheets (CSS) available in HTML, styles can be defined in a tag and referenced in the **style** attribute of widget tags.

Syntax

```
<style id="style-name" value="value-string"/>
```

Attributes

Attribute	Values	Restrictions	Description
id	string	Must not contain a semicolon	The ID to use when referencing the style.
value	string	Must not contain any circular references to itself.	Value contains a style hint string following the same format used in the **style** attribute of other tags.

Examples

The following example applies button style hints as defined in the predefined style `CommandButtonStyle`. Note that the "Stop" button has an additional hint applied (`size:big`), and overrides the background color (`bg#ff0000`).

```
<style id="ButtonStyle" value="bg#808000;bdr:etched;"/>
<button name="Start" style="style:ButtonStyle;"/>
<button name="Stop" style="style:ButtonStyle;size:big;bg#ff0000;"/>
<button name="Reset" style="style:ButtonStyle;"/>
```

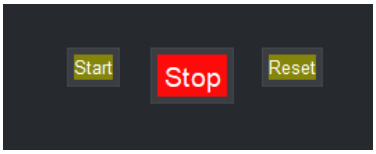


Figure 80 - Style Tag Example

color

Defines or overrides a color constant for use within style hints.

Syntax

```
<color id="color-name" value="color-value"/>
```

Attributes

Attribute	Values	Restrictions	Description
id	string		The ID to use when referencing the color.
value	*#RRGGBB* or *#color-constant* or *#RRGGBBAA*		Value contains a style hint string following the same format used in the color style attribute.

Example

The following example defines a color constant `ColorBlue` and applies it to the background of a button widget.

```
<color id="VibrantBlue" value="#0000FF"/>
<button name="Blue Button" style="bg#VibrantBlue"/>
```

ogscript

Defines an ogScript code snippet to handle an event on a UI element or parameter.

Syntax

```
<ogscript handles="eventType">
ogScript code
</ogscript>
```

Attributes

Attribute	Values	Restrictions	Description
use	online		Script will only run on a real device
	offline		Script will only run on a file-based device
	both		Script will run always
targetid	string		The ID of the UI element to target.
handles		Multiple “handles” arguments can be supplied, separated by commas.	The type of event that triggers the script.

	attributechange		Can be used to trigger scripts when selected NK device is changed or monitor status of FTP download/upload:
			
			<ogscript attribute="com.rossvideo.ftp.event" handles="attributechange">var progressEvent = event.getNewValue();
			if (progressEvent == null)
			{
			ogscript.debug('No progress');
			}
			else
			{
			ogscript.rename('label.bytes', (progressEvent.getTotalBytesTransferred() / 1024) + 'kb');
			}</ogscript>

	dragvalue		Must specify something to return (generally a string or a number) when the element is dragged.
	onaction		Triggered when a button is pressed.

Attribute	Values	Restrictions	Description
	onchange	Only supported by tabs and parameters.	Triggers script when parameter or tab is changed.
	onclick		Triggers script when element is clicked.
	onclose		Triggered when the panel has been closed (can be used for cleaning-up).
	oncontextmenu		Triggers script when the element is right-clicked, or tapped and held. To create a context menu, define an array of menu options, each associated with a segment of ogScript. For more information, see Example of a Context menu on page 146.
	ondrag		Triggers script when the element is dragged
	ondrop		Triggers script when another component is dropped on the component.
	onkeypress		Triggers script when the component has focus and a keyboard key is pressed.
	onkeyrelease		Triggers script when the component has focus and a keyboard key is released.
	onlassoout		Triggers script when a lassostart operation has started and the component with the selected ID is no longer inside of its bounding rectangle.
	onlassoover		Triggers script when a lassostart operation has started and the component with the selected ID is inside of its bounding rectangle.
	onlassostart		Triggers script when the user clicks and starts to drag a 'lasso' rectangle.
	onlassostop		Triggers script when a lasso rectangle that is being dragged stops (see the Ultrix UI with the physical view of the frame for an example).
	onload		Triggered when the panel has finished loading or is reloaded
	onmousedown		Triggers script at onmouse click down event.
	onmouseenter		Triggers script when the pointer moves over the component.
	onmouseexit		Triggers script when the pointer leaves the component.
	onmouseup		Triggers script on mouse click up event.
	onmousemove		Triggers script when mouse moves over component
	onmouseup		Triggers script when the mouse is released after having been pressed while pointing to the component.

Attribute	Values	Restrictions	Description
	onresize		Triggers script when the component is resized.
oid	Positive integer	Must be a defined OID. Only applies to “onchange”	The OID of the parameter to target.
element	List of array indices separated by commas	All array elements referenced must exist in the parameter value. Only applies to “onchange”	By default, all elements of an array parameter are targeted. This attribute can be used to return a subset of the array. If a list is provided, only the elements at the provided indices are returned (note- you can specify the elements in any order). This value must be “0” for a non-array parameter.
script	ogScript	Can also be the text content of the <ogscript> tag.	The script to run when triggered by any of the events listed in “handles”.

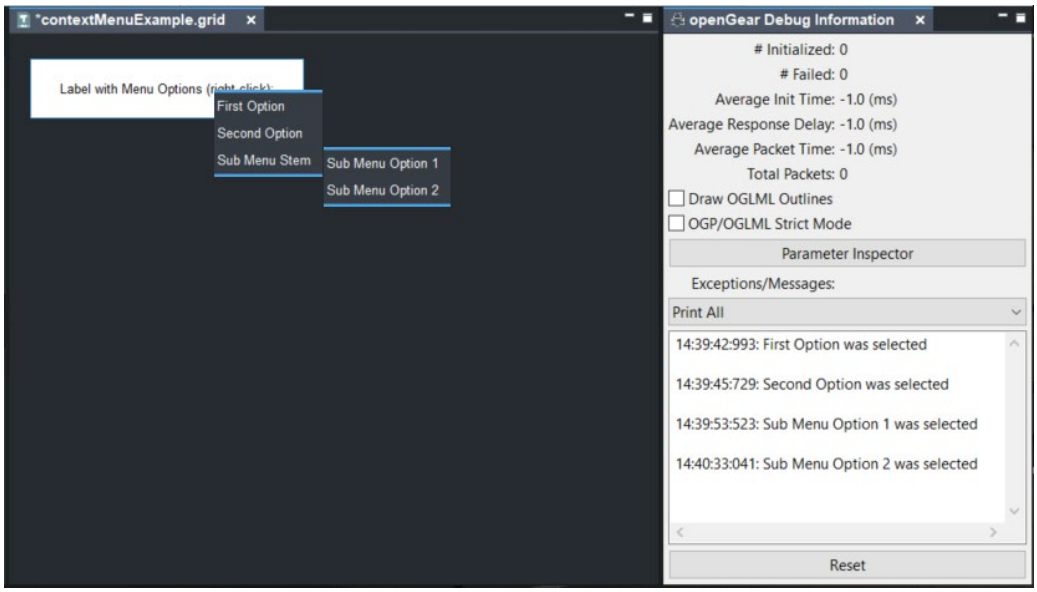
Note: Default values are shown in **bold**.

Example of a Context Menu

In this example, an `<ogscript>` tag uses the `oncontextmenu` event handler to present a menu of options to the user. The menu also includes submenu options. The target is a label with `id="myMenuLabel"`.

When the user right-clicks or taps and holds the label, the menu options appear. When the user clicks or taps a menu option, the function associated with that option is called. In this example, the functions output messages to the openGear debug console.

The following figure shows the context menu fully expanded, and the messages that appear in the openGear debug console when the user selects each menu option.



The following code produces the context menu shown above:

```
<abs contexttype="opengear" style="fg#foreground;">
   <meta>
      <ogscript handles="oncontextmenu" targetid="myMenuLabel">var
myContextMenu = {};
myContextMenu["First Option"] = function()
{
   ogscript.debug("First Option was selected");
};

myContextMenu["Second Option"] = function()
{
   ogscript.debug("Second Option was selected");
};

myContextMenu["Sub Menu Stem"] = {};
myContextMenu["Sub Menu Stem"]["Sub Menu Option 1"] = function()
{
   ogscript.debug("Sub Menu Option 1 was selected");
};

myContextMenu["Sub Menu Stem"]["Sub Menu Option 2"] = function()
{
   ogscript.debug("Sub Menu Option 2 was selected");
};

return myContextMenu;</ogscript>
   </meta>
   <label height="60" id="myMenuLabel" left="21" name="Label with Menu
Options (right-click):" style="txt-
align:center;bdr:line;bdr#selectbg;bg#listbg;fg#foreground;" top="25"
width="275"/>
</abs>
```

constraint

Defines the constraint of a parameter. The structure of this object depends upon the **constrainttype** of the parameter. Range constraints are specified as an attribute of a param tag; Choice, Alarm Table, and Struct constraints are specified using constraint tags as children to the param object.

Constraints may be defined within a `param` declaration, or defined globally and referenced by specific parameters.

Syntax

Constraints with inline constraint values:

```
<constraint constrainttype="ctype" constraint="cvalue" id="constraint-id"
/>
```

Constraints defined within a `param` tag with inline constraint values:

```
<param constrainttype="ctype" constraint="cvalue" param-attributes />
```

Choice and Alarm Constraints:

```
<param constrainttype="ctype" param-attributes>
   <constraint key-attributes>cvalue</constraint>
   <constraint key-attributes>cvalue</constraint>
   . . .
</param>
```

Choice and Alarm Constraints defined within a `param` tag:

```
<param constrainttype="ctype" param-attributes />
   <constraint key-attributes>cvalue</constraint>
   <constraint key-attributes>cvalue</constraint>
   . . .
</param>
```

See sections below for examples and syntax for each constraint type.

Constraint Types

Constraint	Constraint Type	Param Type
Unconstrained	INT_NULL	INT16_PARAM INT16_ARRAY INT32_PARAM INT32_ARRAY
	FLOAT_NULL	FLOAT_PARAM FLOAT_ARRAY
Range Constraint	INT_RANGE INT_STEP_RANGE	INT16_PARAM INT16_ARRAY INT32_PARAM INT32_ARRAY
	FLOAT_RANGE FLOAT_STEP_RANGE	FLOAT_PARAM FLOAT_ARRAY
Integer Choice Constraint	INT_CHOICE	INT16_PARAM INT16_ARRAY INT32_PARAM INT32_ARRAY
String Choice Constraint	STRING_CHOICE	STRING_PARAM STRING_ARRAY
Alarm Table	ALARM_TABLE	INT16_PARAM INT32_PARAM
Constraint Reference	ID_REFERENCE	*All*
Structure	STRUCT	STRUCT_PARAM STRUCT_ARRAY

Note *If no constraint is specified for a parameter, it will be unconstrained by default.*

Refer to the appropriate section below for definition of the constraint object for each constraint type.

constraint (Unconstrained)

Specifies that a parameter is unconstrained. All parameters are considered unconstrained by default if no constraint is applied.

Syntax

```
<param constrainttype="constraint-type" attributes />
```

Attributes

Attribute	Values	Restrictions	Description
constrainttype	INT_NULL	**param type**: INT16_PARAM INT16_ARRAY INT32_PARAM INT32_ARRAY	Parameter is unconstrained
	FLOAT_NULL	**param type**: FLOAT_PARAM FLOAT_ARRAY	

Examples

The following constraint specifies an integer to be unconstrained:

```
<param constrainttype="INT_NULL" name="Delay" oid="0x500"
type="INT16_PARAM"/>
```

constraint (Constraint Reference)

References a globally-defined constraint. A constraint may be specified globally in the `<meta>` block. These globally-defined constraints may then by referenced by specific parameters.

Syntax

```
<constraint id="constraint-id" constrainttype="constraint-type">

<param constrainttype="ID_REFERENCE" constraint="constraint-id"
attributes />
```

Attributes

Attribute	Values	Restrictions	Description
id	*String*		Unique identifier for this constraint
constrainttype	*Any valid constraint type*	Param type must be compatible with the referenced constraint.	See Constraint Types for valid constraint types.

Examples

The following example creates a global constraint `VideoFormat`. Params `0x501`, `0x502` and `0x503` are all constrained using this constraint definition.

```
<constraint constrainttype="INT_CHOICE" id="VideoFormat">
  <constraint key="0">480i-59.94</constraint>
  <constraint key="1">576i-50</constraint>
  <constraint key="2">1080i-29.97</constraint>
  <constraint key="3">1080i-25</constraint>
  <constraint key="4">720p-59.94</constraint>
  <constraint key="5">720p-50</constraint>
  <constraint key="6">1080p-59.94</constraint>
  <constraint key="7">1080p-50</constraint>
</constraint>

<param constrainttype="ID_REFERENCE" constraint="VideoFormat" name="Vid1"
oid="0x501" type="INT16_PARAM"/>

<param constrainttype="ID_REFERENCE" constraint="VideoFormat" name="Vid2"
oid="0x502" type="INT16_PARAM"/>

<param constrainttype="ID_REFERENCE" constraint="VideoFormat" name="Vid3"
oid="0x503" type="INT16_PARAM"/>
```

constraint (Range Constraints)

Constrains a numeric parameter type to a specific range. Minimum and maximum values effect the parameter's valid range. Display minimum and maximum values scale the parameter value to a different range for display purposes. Finally a step value can be set to constrain the minimum step size a value may be changed by.

Syntax

Min / Max Constraint:

```
<param constraint="min;max;" constrainttype="constraint-type" attributes
/>
```

Min / Max Constraint with Display-Min and Display-Max:

```
<param    constraint="min;max;display-min;display-max;"
      constrainttype="constraint-type" attributes />
```

Min / Max Step Constraint:

```
<param constraint="min;max;step" constrainttype="constraint-type"
attributes />
```

Min / Max Step Constraint with Display-Min and Display-Max:

```
<param    constraint="min;max;display-min;display-max;step"
      constrainttype="constraint-type" attributes />
```

Attributes

Attribute	Values	Restrictions	Description
constrainttype	INT_RANGE INT_STEP_RANGE	**param type**: INT16_PARAM INT16_ARRAY INT32_PARAM INT32_ARRAY	Type of constraint
	FLOAT_RANGE FLOAT_STEP_RANGE	**param type**: FLOAT_PARAM FLOAT_ARRAY	
constraint	*min*	Required	Minimum value to which a parameter can be set
	max	Required	Minimum value to which a parameter can be set
	display-min	Optional; must be used with **display_max.**	The displayed value of the parameter when the parameter has a value of *min*. The default value is *min*.
	display-max	Optional; must be used with **display_min.**	The displayed value of the parameter when the parameter has a value of *max*. The default value is *max*.
	step	*xxx*_STEP_RANGE constraints only	Smallest increment a value may be changed by. Spinner widgets will increment a parameter by the **step** value. Note that the step increment is applied to the parameter value, not the display value.

Examples

The following example constrains a FLOAT_PARAM to [0,100]:

```
<param constraint="0.0;100.0;" constrainttype="FLOAT_RANGE" name="Delay"
oid="audio.delay" type="FLOAT_PARAM"/>
```

The following example constrains an integer to [0, 255] mapping it to a display range of [0, 100], and the value increments by steps of 2:

```
<param constraint="0;255;0;100;2" constrainttype="INT_STEP_RANGE"
name="Gain" oid="key1.gain" type="INT16_PARAM"/>
```

constraint (Integer Choice Constraints)

Choice constraints provide a list of possible values for a parameter, based upon a text selection. For integer parameters, the parameter may only be assigned a value specified in the constraint.

Syntax

```
<param constrainttype="INT_CHOICE" type="param-type" attributes >
   <constraint key="choice1-key">choice1-value</constraint>
   <constraint key="choice2-key">choice2-value</constraint>
   . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
type	INT16_PARAM INT32_PARAM INT16_ARRAY INT32_ARRAY		Parameter must be integer type.
key	*Integer*		Numeric assignment of current enumerated choice.
value	*String*		Text name for the current enumerated choice

Examples

The following constraint provides an enumerated choice:

```
<param constrainttype="INT_CHOICE" name="Channel" oid="0x503"
type="INT16_PARAM">
  <constraint key="0">Channel 01</constraint>
  <constraint key="1">Channel 02</constraint>
  <constraint key="2">Channel 03</constraint>
  <constraint key="3">Channel 04</constraint>
</param>
```

constraint (String Choice Constraints)

Choice constraints provide a list of possible values for a parameter, based upon a text selection. For String parameters, the constraint provides a set of defaults, but the user may arbitrarily enter any other value for the parameter.

Syntax

```
<param constrainttype="STRING_CHOICE" type="param-type" attributes >
   <constraint>value</constraint>
   <constraint>value</constraint>
   . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
type	STRING_PARAM STRING_ARRAY		Parameter must be string type.
value	*String*		Available strings for drop-down widget

Examples

The following constraint provides five string options for a String parameter.

```
<param constrainttype="STRING_CHOICE" name="Name" oid="0x504"
type="STRING_PARAM">
  <constraint>Zeus Test Card</constraint>
  <constraint>ZTC</constraint>
  <constraint>Johnny</constraint>
  <constraint>Matilda</constraint>
</param>
```

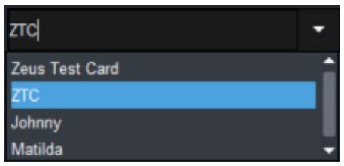


Figure 81 – String Choice

constraint (Alarm Table)

Alarm constraints map a set of alarms as bitfields into an INT16_PARAM or INT32_PARAM. Each bit represents an independent alarm which may have a message and severity assigned to it. Alarm parameters contribute to the device's overall alarm status in DashBoard; the most severe alarm set will determine the device's overall reported alarm status.

Syntax

```
<param constrainttype="ALARM_TABLE" type="param-type" attributes >
   <constraint key="bit-number" severity="severity">value</constraint>
   <constraint key="bit-number" severity="severity">value</constraint>
   . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
type	INT16_PARAM INT32_PARAM		Parameter must be integer type.
key	*Integer*	INT16: 0..15 INT32: 0..31	The bit position for the alarm (0 is LSB).
severity	*Integer*		The severity of the alarm: 0 = OK 1 = WARN 2 = ERROR
value	*String*		Alarm message text

Examples

The following constraint creates an alarm table:

```
<param constrainttype="ALARM_TABLE" name="Alarm" oid="0x504"
type="INT16_PARAM">
  <constraint key="0" severity="0">Hardware OK</constraint>
  <constraint key="1" severity="2">Hardware Error</constraint>
  <constraint key="2" severity="1">Flash Memory Full</constraint>
</param>
```

constraint (Struct Constraints)

Struct Constraints allow a parameter to define a complex structure of multiple parameters. The Struct Constraint is applied to each parameter that is an instance of a Struct.

Syntax

```
<param constrainttype="STRUCT" structtype="struct-type"
templateoid="template-oid"
   type="STRUCT" param-attributes>
```

Attributes

Attribute	Values	Restrictions	Description
type	*String*	Required	Set to "STRUCT"
templateoid	*String*		Specifies a template OID to pre-populate the structure. All parameters, constraints and widgets for the sub-OIDs are copied from the template.
structtype	*String*	Must be unique	Defines the structure type. Used by PanelBuilder to type-check custom widgets against defined struct parameters.

Examples

The following code is an example of a struct definition.

```
<param constrainttype="STRUCT" name="Clip Info" oid="clipInfo"
       structtype="playinfo" type="STRUCT" widget="36">
   <value>
       <subparam name="Clip Name" suboid="ClipName" type="STRING"
value="Test"/>
       <subparam name="Director" suboid="Director" type="STRING"
value="Test"/>
       <subparam name="Date" suboid="AirDate" type="STRING"
value="Test"/>
       <subparam name="Author" suboid="Author" type="STRING"
value="Test"/>
   </value>
</param>
```

The following declaration utilizes the previous example as a template, by specifying the templateoid attribute:

```
<param constrainttype="STRUCT" name="Clip List" oid="clipList"
   structtype="playinfo" templateoid="clipInfo" type="STRUCT_ARRAY"
widget="36">
   <value>
      <subparam suboid="ClipName" value="Winter is Coming"/>
      <subparam suboid="Director" value="Tim Van Patten"/>
      <subparam suboid="OriginalAirDate" value="April 24, 2011"/>
      <subparam suboid="Author" value="David Benoiff &amp;amp; D.B. Weiss"/>
   </value>
   <value>
      <subparam suboid="ClipName" value="The Kingsroad"/>
      <subparam suboid="Director" value="Brian Kirk"/>
      <subparam suboid="OriginalAirDate" value="April 24, 2011"/>
      <subparam suboid="Author" value="David Benoiff &amp;amp; D.B. Weiss"/>
   </value>
   <value>
      <subparam suboid="ClipName" value="Lord Snow"/>
      <subparam suboid="Director" value="Brian Kirk"/>
      <subparam suboid="OriginalAirDate" value="May 1, 2011"/>
      <subparam suboid="Author" value="David Benoiff &amp;amp; D.B. Weiss"/>
   </value>
   <value>
      <subparam suboid="ClipName" value="A Golden Crown"/>
      <subparam suboid="Director" value="Daniel Minahan"/>
      <subparam suboid="OriginalAirDate" value="May 22, 2011"/>
      <subparam suboid="Author" value="David Benioff &amp;amp; D. B.
Weiss"/>
   </value>
</param>
```

params

The parent container for parameters defined within the OGLML document. This tag may only contain `<param>` tags.

Syntax

```
<params>
   <param param-attributes />
   <param param-attributes />
   . . .
</params>
```

Attributes

None.

timer

The timer tag fires events at regular intervals. Timers can operate on their own or linked to other timers. ogScript commands exist to start/stop/reset timers (see ogScript documentation for more details).

Tasks are attached to listener tags to process data received.

Attributes

Attribute	Values	Restrictions	Description
id	String	Optional	The ID used to reference this timer. Required for ogScript, child timers, or external <timertask/> tags to interact with the timer.
source	String	Optional. Must be the ID of another timer.	If used, the timer being defined will be a child of the timer with the given ID.
rate	Long	Not applicable if “source” is set.	The rate (in milliseconds) at which the timer fires.
delay	Long	Not applicable if “source” is set.	The delay (in milliseconds) before the timer initially fires.
pattern	String		The display pattern for the timer’s current time: https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html
start	Long or time in format of “pattern”		The start value of the timer. If start > stop, timer counts down. If start is undefined, the timer is ‘clock mode’
stop	Long or time in format of “pattern”		The start value of the timer. If start > stop, timer counts down. If start is undefined, the timer is ‘clock mode’
autostart	True *false	Default value is true if ‘clock mode’ is used.	Whether or not the timer automatically starts. If it is not automatically started, an ogScript command must be issued to the timer to start it.

listener

The listener tag allows an OGLML page to process network communications using protocols not already available. It is designed for small and simple protocols only.

The listener tag can work in two different modes: listen for incoming connections (server mode) or establish a connection (client mode). In both cases, the listener tag will listen for incoming data from the remote system.

Tasks are attached to listener tags to process data received.

Attribute	Values	Restrictions	Description
connecthost	String	Cannot be used if listenport is defined.	The hostname of the remote host to connect to.
connectport	Integer	Cannot be used if listenport is defined.	The port to connect to on the remote host.
listenport	Integer	Cannot be used if connectport/connect host are defined.	The local port to listen on for new connections.
delimitertype	newline bytes fixedlen varlen string	Required.	The mechanism used to separate one incoming message from another. “newline” = read bytes until 0x0A is received “bytes” = convert value in “delimiter” attribute into a byte array and wait for those bytes. “fixedlen” = read a fixed number of bytes for each message. “string” = convert value in “delimeter” into UTF-8 bytes and wait for those bytes. “varlen” = convert value in “delimiter” to an integer “n”. The first [n] bytes of the message indicate how many bytes follow.
delimiter		May be required depending on value of **delimitertype**	The data for the delimiter. Changes depending on the value of **delimitertype** bytes: The bytes in the message delimiter. E.g. to listen for a Carriage Return/Line Feed combination “0D0A”. fixedlen: The number of bytes in each message. String: The UTF-8 String to wait for to indicate the end of a message. E.g. “END” varlne: The number of bytes to read to determine message length. E.g. if your protocol defines a 2-byte length count at the beginning of each message, the value would be “2”.
syncword		Optional	Defines an array of bytes to read at the start of an incoming message. E.g. for openGear protocol, the sync word would be “BAD2ACE5”

Attribute	Values	Restrictions	Description
blockingpause	true **false**		When processing tasks, blockingpause means that all message processing is done in the message RX Thread. This means that if a “pause” task is encountered, all RX of messages will pause too.
buttontype	toggle **none**		If no button is defined, the listener is automatically started. If a button is defined, this allows the user to toggle the listener off/on.
autostart	true false		Whether or not the listener should be automatically started. This is always true if no **buttontype** has been defined.

Example

```
<listener autostart="true" delimitertype="newline" listenport="12345">
   <task tasktype="ogscript">if (event.isMessageEvent())
   {
      var rec = event.getBytesAsString().trim();
      var response = '';
      for (var i = rec.length - 1; i &gt;= 0; i--)
         {
           response += rec.charAt(i);
         }
      this.writeString('REVERSE: ' + response + '\n', false);
   }
   </task>
</listener>
```

task

Defines a block of ogScript to be run when an event happens in the system. Tasks inside of **label** tags are fired when the label is clicked. Tasks inside of **button** tags are fired when the button is pressed. Tasks inside of **listener** tags are fired whenever a connection is established or whenever data is received.

The text content of the tag contains the actual ogScript to be executed.

Syntax

```
<component>
         <task tasktype="task-type">ogScript-code</task>
</component>
```

Attributes

Tag	Values	Restrictions	Description
tasktype	*ogscript robot vdcp rosstalk ogparamset timercontrol		This attribute tells the editor user interface what type of task is contained in the tag body. Manually-edited tasks should simply use ogscript.

timertask

Defines a block of ogScript to be run when a timer goes off. The timer must be in the same scope as the timertask. The text content of the tag contains the actual ogScript to be executed.

Syntax

```
<container>
  <timer id="timer-id"/>
  <container>
        <container>
                  <timertask tasktype="task-type" timerid="timer-id">
                    ogScript Code
                 </timertask>
         </container>
  </container>
</container>
```

Attributes

Tag	Values	Restrictions	Description
tasktype	*ogscript robot vdcp rosstalk ogparamset timercontrol		This attribute tells the editor user interface what type of task is contained in the tag body. Manually-edited tasks should simply use ogscript.
timerid	String	Must match the **id** attribute of a timer tag accessible in this tasks's scope.	Defines the ID of a timer to fire this timertask. This allows a timer to be defined at the document root but perform actions on elements defined much deeper in the document structure.

include

This tag allows an OGLML document to be assembled from several individual XML files or fragments. The tag provides a URL, which is retrieved and then replaces the tag with the contents of the referenced OGLML document.

Attribute	Values	Restrictions	Description
*src	URL for http, https, or “eo”		Points to an OGLML document at the given URL. Documents are refreshed when a card is re-queried (i.e. either card sends an external object change, or OGP_RESTART, or user clicks “refresh”). HTTP fetches use if-modified-since header and ETag (as defined in RFC 2616 section 14.25 and 14.19 respectively) A DashBoard-specific scheme “eo” can be used to fetch content from an External Object. Examples would be “eo://1234” or “eo://0x4D2”). If this format is used, DashBoard will look for the OGLML document referenced by the provided external object (contained within an OGLML Descriptor). For more information, see OGLML URLs on page 53.

Device Resource Declarations

This section describes tags used to declare resources. These tags may be used in a stand-alone XML file (such as a `.ogd` or `.xml` file), or may be embedded within an OGLML document (typically within a `<meta>` block).

Resource XML File

Data store resources may be backed by an XML file. Below is an outline of the XML file structure:

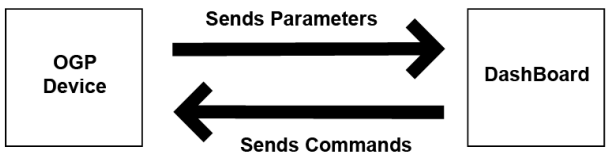
```
<?xml version="1.1" encoding="UTF-8"?>
   <frame>
      <card>
         <params>
            <param/>
            <param/>
            . . .
         </params>
         <statusmenu>
            <menu>
               <param/>
               <param/>
```

```
            . . .
        </menu>
        <menu/>
        . . .
    </statusmenu>
    <configmenu>
        <menu>
            <param/>
            <param/>
            . . .
        </menu>
        <menu/>
        . . .
    </configmenu>
    <menugroup>
        <menu>
            <param/>
            <param/>
            . . .
        </menu>
        <menu/>
        . . .
    </menugroup>
    <menugroup/>
    . . .
</card>
<card/>
. . .
</frame>
```

Resources within the `<card>` block may also be declared within an OGLML document, and should be located within a `<meta>` block.

commands

Defines an OGP command for a device. OGP commands provide a way to use the OGP connection to execute commands from other devices.



The primary difference between using commands and parameters, is that the DashBoard OGP Client does not keep track of the state of the parameters in a command. The value of each parameter is specific to the execution request. This allows DashBoard to send multiple crosspoint

command requests to the device and each one can have different values for the source/destination.

Once an OGP device has been added to DashBoard, you can use OGP commands to issue device commands directly from a CustomPanel. For example, the CustomPanel below shows a subset of a device commands that have been added to a CustomPanel. You can also create workflows using logic blocks in the Visual Logic Editor or editing the code directly in the ogScript Editor.

Syntax

```
"command1":{
    "oid": "command1",
    "name": "command 1",
    "type": "STRUCT",
    "readonly": false,
    "widget": "default",
    "value": ...
        },
"command2":{
    "oid": "command2",
    "name": "command 2",
    "type": "STRUCT",
    "readonly": false,
    "widget": "default",
    "value": ...
        }
```

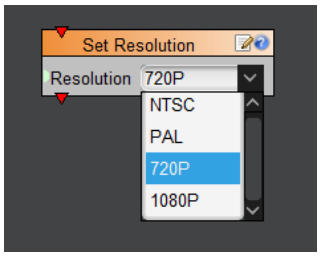
Attributes

Attribute	Values	Restrictions	Description
oid	*String*	*Required	Command oid.
name	*String*	Not required	Command name.
type	*String*	*Required	Data type for the command.
readonly	*Boolean*	Not required	If set to true, the parameter is read-only.
widget	*String*	Not required	The widget used to dislay the data in DashBoard.
constraint	*constraint Object*	Not required	Parameter Constraint.
config	*config Object*	Not required	Extended parameter configuration.
value	*String*	Not required	Value of the parameter. Defines an argument that can be passed to the command.

Examples

This example shows a command called `"SetResolution"` that has a `"Resolution"` argument that is constrained to the following choices: NTSC, PAL, 720P, and 1080P. The command is also shown in the Visual Logic Editor below.

Figure 82 –Visusal Logic Representation of the Command




```
"commands":{
    "SetResolution":{
        "oid":"SetResolution",
        "name":"Set Resolution",
        "readonly":false,
        "type":"STRUCT",
        "widget":"default",
        "value":[
            {
                "ResolutionOptions":{
                    "name":"Resolution",
                    "readonly":false,
                    "type":"STRING",
                    "widget":"text",
                    "maxlength":"0",
                    "totallength":"0",
                    "constraint":{
                        "value":"STRING_STRING_CHOICE",
                        "choices":[
                            {
                                "value":"NTSC",
                                "key":"NTSC"
                            },
                            {
                                "value":"PAL",
                                "key":"PAL"
                            },
                            {
                                "value":"720P",
                                "key":"720P"
                            },
                            {
                                "value":"1080P",
                                "key":"1080P"
                            }
                        ],
                        "strict":false
                    },
                    "value":"720P"
                }
```

```
            }
        ],
        "constraint":{

        },
        "response":true
    }
}
```

command

Defines an OGP command for a device. OGP commands provide a way to use the OGP connection to execute commands from other devices. For more information, see the entry above.

Syntax

```
"command1":{
    "oid": "command1",
    "name": "command 1",
    "type": "STRUCT",
    "readonly": false,
    "widget": "default",
    "value": ...
        }
```

Attributes

Attribute	Values	Restrictions	Description
oid	*String*	*Required	Command oid.
name	*String*	Not required	Command name.
type	*String*	*Required	Data type for the command.
readonly	*Boolean*	Not required	If set to true, the parameter is read-only.
widget	*String*	Not required	The widget used to dislay the data in DashBoard.
constraint	*constraint Object*	Not required	Parameter Constraint.
config	*config Object*	Not required	Extended parameter configuration.
value	*String*	Not required	Value of the parameter. Defines an argument that can be passed to the command.

config

Provides a container for extended configuration key-value pairs for elements related to a

parameter. Contents are dependent on other constraints, parameter types or widgets.

Syntax

```
<param>
    <config key="key">value</config>
    <config key="key">value</config>
    . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
key	*String*		Configuration parameter name
value	*String*		Configuration parameter value

Example

The following config object sets attributes of a graph widget:

```
<param oid="Fader_Bar" right="5" widget="256">
   <config key="w.time">5</config>
   <config key="w.autoadvance">true</config>
   <config key="w.plotbg">#dark</config>
   <config key="w.plotfg">#00FF00</config>
   <config key="w.grid">#panelfg</config>
   <config key="w.hidelegend">true</config>
   <config key="w.hidey">false</config>
   <config key="w.hidex">false</config>
</param>
```

constraint

Defines the choice constraint for a parameter. For `INT_CHOICE` constraints, the integer value is defined with the *key* attribute and the text to display is the text content of the tag. For `STRING_CHOICE` constraints, each constraint tag contains a value to populate a combo-box drop-down.

The parameter must have a constrainttype of `INT16_CHOICE`, `INT32_CHOICE` or `STRING_CHOICE`.

Syntax

```
<constraint key="choice1-key">choice1-value</constraint>
```

Attributes

Attribute	Values	Restrictions	Description
key	*Integer*	Not required for STRING_CHOICE constraints	Numeric assignment of current enumerated choice.
value	*String*		Text name for the current enumerated choice

Examples

The following constraint provides an enumerated choice:

```
<param constrainttype="INT_CHOICE" name="Channel" oid="0x503"
type="INT16_PARAM">
  <constraint key="0">Channel 01</constraint>
  <constraint key="1">Channel 02</constraint>
  <constraint key="2">Channel 03</constraint>
  <constraint key="3">Channel 04</constraint>
</param>
```

The following constraint provides a list of selections for a STRING parameter:

```
<param constrainttype="STRING_CHOICE" name="Name" oid="0x504"
type="STRING">
  <constraint>Jeremy Clarkson</constraint>
  <constraint>James May</constraint>
  <constraint>Richard Hammond</constraint>
  <constraint>The Stig</constraint>
</param>
```

card

Top-level container for a device within an `XML` or `OGD` file. Encapsulates a device within a `.frame` file. Note this tag should not be used as a container within an OGLML document.

Syntax

```
<card autosave="auto-save"
      online="true" slot="slotno" sourceframe="frame-node-id"
      sourceframename="device-name" sourceid="card-node-id"
      status="status-level" statustext="status-text" version="2.0">
```

Attributes

Attribute	Values	Restrictions	Description
autosave	true **false**		If true, DashBoard will automatically save contents of the resources specified in the file from data store periodically.
online	**true** false		Sets the device's online status. Normally should be set to **true**.
slot	*Integer*		Defines the slot-id for the device.
sourceframe	*String*		**node-id** of the frame or device.
sourceframename	*String*		Name of the device. This is the top-level name shown in the DashBoard Tree
sourceid	*String*		The original node-id of the virtual device (used when saved as the offline configuration of a real device)
status	0	Not required for PanelBuilder	Status OK
	1		Status WARN
	2		Status ERROR
statustext	*String*	Not required for PanelBuilder	Status text for the node.
version	*String*		Set to `2.0`.

Default values shown in **bold**.

Example

The following example defines a device (openGear card) installed in a frame called “Demo Frame”, slot 10. The device’s node id is “172.16.7.230:5253(Slot10)SPG-8260”.

```
<card online="true" slot="10" sourceframe="172.16.7.230:5253"
      sourceframename="Demo Frame"
      sourceid="172.16.7.230:5253&lt;br>Slot 10&lt;br>SPG-8260"
      status="0" statustext="OK" version="2.0">
</card>
```

frame

Top-level container for a frame within a `.frame` file. Note this tag should not be used as a container within an OGLML document. Frame files are created by DashBoard.

Syntax

```
<frame    name="frame-name" sourceid="node-id" >
   <card/>
   <card/>
   . . .
</frame>
```

Attributes

Attribute	Values	Restrictions	Description
Name	*String*		Display name of the frame
sourceid	*String*		The original node-id of the virtual device (used when saved as the offline configuration of a real device)

menu

Defines the controls to place within a menu tab or menu pop-up. `<param>` tags within the menu block may override the param’s default attributes for display within this menu.

Synax

```
<menu menuid="menu-id" menustate="state" name="name" staticid="static-
id">
   <param/>
   <param/>
   . . .
</menu>
```

Attributes

Attribute	Values	Restrictions	Description
menuid	*Integer*	Required	Numeric ID for the menu. Menu tabs within a menu group are displayed in numeric order, lowest first. This value may be changed to dynamically re-order menus.
menustate	0		Menu is hidden
	1		Menu is displayed, but params are read-only
	2		Menu is displayed and params are read/write (based upon individual parameter access permissions)
name	*String*		Name of the menu. This name will appear in the menu tab.
staticid	*Integer*	Required	Unique numeric identifier for this menu. This value must be only set once and not changed.

Example

The following example creates a menu called “Network Setup”.

```
<menu menuid="257" menustate="2" name="Network Setup" staticid="257">
    <param access="1" name="Addressing Mode" oid="0x0x711"/>
    <param access="1" name="IP Address" oid="0x712"/>
    <param access="1" name="Subnet Mask" oid="0x713"/>
    <param access="1" name="Default Gateway" oid="0x714"/>
</menu>
```

menugroup

Defines a menu group. The menugroup is a container for menus. When a menugroup is displayed, child menus are displayed as tabbed elements within the container.

Syntax

```
<menugroup menuid="id" name="menu-group-name">
    <menu/>
    <menu/>
    . . .
</menugroup>
```

Attributes

Attribute	Values	Restrictions	Description
menuid	*Integer*	Required	Numeric ID for the menu group. This value must be only set once and not changed. **menuid**=0 corresponds to the openGear status menu **menuid**=1 corresponds to the openGear configuration menu

Attribute	Values	Restrictions	Description
name	*String*		Name of the menu group.

Examples

The following example creates a menu group with two menus:

```
<menugroup menuid="0" name="Status">
    <menu menuid="0" menustate="2" name="Status" staticid="0">
        <param access="0" name="Card Status" oid="0x201"/>
        <param access="0" name="Reference" oid="0x204"/>
    </menu>
    <menu menuid="1" menustate="2" name="Product Info" staticid="1">
        <param access="0" name="Product" oid="0x105"/>
        <param access="0" name="Name" oid="0x107"/>
        <param access="0" name="Supplier" oid="0x102"/>
        <param access="0" name="Software Rev" oid="0x10B"/>
    </menu>
</menugroup>
```

statusmenu

Defines the Status Menu group for the default openGear menu layout. This tag behaves in the same manner as the <menugroup> tag when the **menuid**=0.

Syntax

```
<statusmenu menuid="id" name="menu-group-name">
    <menu/>
    <menu/>
    . . .
</statusmenu>
```

Attributes

Attribute	Values	Restrictions	Description
menuid	*Integer*	Optional. Should be set to 0.	Numeric ID for the menu group. This value must be only set once and not changed. Defaults to 0.
name	*String*		Name of the menu group.

Example

The following example creates a status menu group with two menus:

```
<statusmenu menuid="0" name="Status">
    <menu menuid="0" menustate="2" name="Status" staticid="0">
        <param access="0" name="Card Status" oid="0x201"/>
        <param access="0" name="Reference" oid="0x204"/>
    </menu>
    <menu menuid="1" menustate="2" name="Product Info" staticid="1">
        <param access="0" name="Product" oid="0x105"/>
        <param access="0" name="Name" oid="0x107"/>
        <param access="0" name="Supplier" oid="0x102"/>
        <param access="0" name="Software Rev" oid="0x10B"/>
    </menu>
</statusmenu>
```

configmenu

Defines the Config Menu group for the default openGear menu layout. This tag behaves in the same manner as the <menugroup> tag when the **menuid**=1.

Syntax

```
<configmenu menuid="id" name="menu-group-name">
    <menu/>
    <menu/>
    . . .
</configmenu>
```

Attributes

Attribute	Values	Restrictions	Description
menuid	*Integer*	Optional. Should be set to 1.	Numeric ID for the menu group. This value must be only set once and not changed. Defaults to 1.
name	*String*		Name of the menu group.

Example

The following example creates a status menu group with 2 menus:

```
<configmenu menuid="1" name="Status">
    <menu menuid="513" menustate="2" name="Network Setup" staticid="257">
        <param access="1" name="Addressing Mode" oid="0xFE11"/>
        <param access="1" name="IP Address" oid="0x712"/>
        <param access="1" name="Subnet Mask" oid="0x713"/>
        <param access="1" name="Default Gateway" oid="0x714"/>
    </menu>
    <menu menuid="514" menustate="2" name="Remote Control Setup"
staticid="258">
        <param access="1" name="Protocol" oid="0x411"/>
        <param access="1" name="Baud Rate" oid="0x412"/>
        <param access="1" name="Parity" oid="0x413"/>
```

```
        <param access="1" name="Stop Bits" oid="0x414"/>
    </menu>
</statusmenu>
```

params

The parent container for parameters defined within the OGLML document. This tag may only contain <param> tags.

Syntax

```
<params>
    <param param-attributes />
    <param param-attributes />
    . . .
</params>
```

Attributes

None.

param

Creates a parameter descriptor, which defines the parameter. Declaration of a param descriptor must be located within a <params> block. Constraints for the param may be included as an attribute (for range constraints), or as child tags (for choice constraints).

Syntax

```
<param oid="oid" attributes/>

<param oid="oid" attributes>
    <constraint/>
    <constraint/>
    . . .
    <config/>
    <config/>
    . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
oid	String	Required, except for subparams	The OID of the parameter (can be used to override an existing parameter).
suboid	String	Required for subparams	If the param declaration is a sub-param within a struct, the OID is specified in the suboid attribute.
access	0		Parameter is read-only in DashBoard
	1		Parameter is read-write in DashBoard
name	String		Parameter Name

Attribute	Values	Restrictions	Description
widget	Positive integer	Must be a valid widget hint	Defines the default widget hint for the param.
maxlength	Positive integer	Applies only to String/String Array parameters	The maximum length of any String element in the parameter.
precision	Positive integer		This field defines the number of digits following the decimal point displayed for printed numbers. It applies mainly to floating point numbers.
type	INT16		Param is 16-bit signed integer.
	INT16_ARRAY		Param is an array of 16-bit signed integer.
	INT32		Param is 32-bit signed integer.
	INT32_ARRAY		Param is an array of 32-bit signed integer.
	STRING		Param is a string.
	STRING_ARRAY		Param is an array of strings.
	FLOAT32		Param is a 32-bit (IEEE single) float.
	FLOAT32_ARRAY		Param is an array of 32-bit (IEEE single) float.
	STRUCT		Param is a struct.
	STRUCT_ARRAY		Param is an array of struct.
	BINARY_VALUE		Param is of unknown type.
constraint	*Cvalue*		See constraint tag for more details.
constrainttype	*Ctype*		See constraint tag for more details.
stateless	**False**		Parameters are saved to backing source
	True		Parameters are not saved
value	*Varies*	Value type must be compatible with the specified **type**.	Specifies the initial value of the param. Arrays may be initialized by separating values with ";".
config	*Varies*		Provides additional widget configuration parameters.

Default values shown in **bold**.

Example

```
<param access="1" maxlength="0" name="Message" oid="Message"
type="STRING" value="Reverse this message" widget="3"/>
```

param (struct)

Compound parameters may be defined through the use of the `STRUCT` param type. A **struct** contains a collection of parameters. **Structs** may not be nested. **Struct** must have a **constrainttype** of `STRUCT`. Members of the **struct** are declared through `subparam` tags within the `value` tag.

A struct may also use another param as a template to pre-populate the member sub-param declarations. This is done through the `templateoid` attribute.

Syntax

```
<param constrainttype="STRUCT" oid="oid" type="STRUCT" attributes>
   <value>
      <subparam suboid="sub-oid" sub-param-attributes/>
      <subparam suboid="sub-oid" sub-param-attributes/>
      . . .
   </value>
</param>
```

Attributes

Attribute	Values	Restrictions	Description
oid	*String*	Required	The OID of the parameter (can be used to override an existing parameter).
access	0		Parameter is read-only in DashBoard
	1		Parameter is read-write in DashBoard
name	*String*		Parameter Name
widget	Positive integer	Must be a valid widget hint	Defines the default widget hint for the param.
type	STRUCT		Must be set to STRUCT.
structtype	*String*		Defines the structure type. Specifies a dependency of a widget upon a global struct parameter with matching **structtype**. Currently this type checking is restricted only to PanelBuilder UI; a custom widget will only be available in PanelBuilder if a parameter exists with matching **structtype**.
templateoid	*String*		Specifies a template struct parameter to pre-populate the subparams.
constrainttype	STRUCT		Must be set to STRUCT
value			Container for **subparam** elements.
subparam	*param*	May not be a nested struct param	Member parameters, declared using the same syntax as a param declaration, with the exception that its oid is specified in the attribute **suboid**.

Default values shown in **bold**.

Example

The following declares a struct parameter.

```
<param access="1" constrainttype="STRUCT" name="Clip Info" oid="clipInfo"
type="STRUCT" widget="36">
    <value>
        <subparam name="Clip Name" suboid="ClipName" type="STRING"
    value="Test"/>
        <subparam name="Director" suboid="Director" type="STRING"
    value="Test"/>
        <subparam name="Air Date" suboid="AirDate" type="STRING"
    value="Test"/>
        <subparam name="Author" suboid="Author" type="STRING"
    value="Test"/>
    </value>
</param>
```

The following declares an array of struct params, using the previous example as its template. Note that any attributes specified explicitly will override the values provided in the template.

```
<param access="1" constrainttype="STRUCT" name="Clip List" oid="clipList"
templateoid="clipInfo" type="STRUCT_ARRAY" widget="36">
    <value>
       <subparam suboid="ClipName" value="Winter is Coming"/>
       <subparam suboid="Director" value="Tim Van Patten"/>
       <subparam suboid="AirDate" value="April 24, 2011"/>
       <subparam suboid="Author" value="David Benoiff &amp; D.B. Weiss"/>
    </value>
    <value>
       <subparam suboid="ClipName" value="The Kingsroad"/>
       <subparam suboid="Director" value="Brian Kirk"/>
       <subparam suboid="AirDate" value="April 24, 2011"/>
       <subparam suboid="Author" value="David Benoiff &amp; D.B. Weiss"/>
    </value>
    <value>
       <subparam suboid="ClipName" value="Lord Snow"/>
       <subparam suboid="Director" value="Brian Kirk"/>
       <subparam suboid="AirDate" value="May 1, 2011"/>
       <subparam suboid="Author" value="David Benoiff &amp; D.B. Weiss"/>
    </value>
</param>
```

Device Resource Tags

The following tags use resources provided by the same device that sent the OGLML document to DashBoard.

The following tags can be used to incorporate standard openGear UI elements into an OGLML document. For example the typical device page is composed of the following tagged resources.

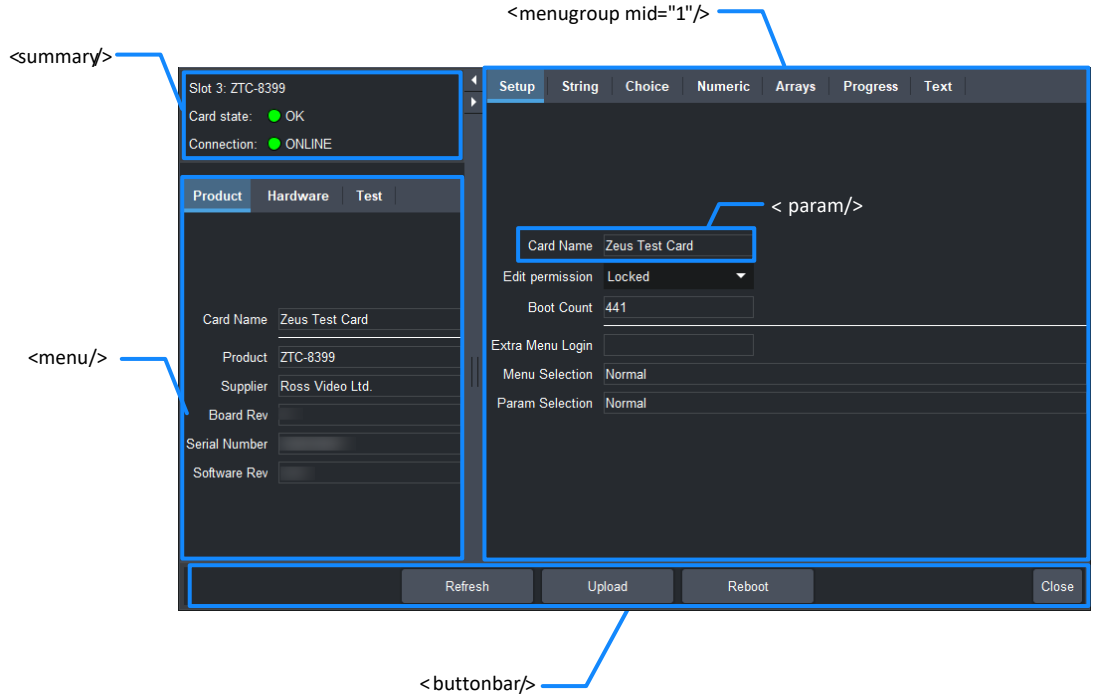


Figure 83 – Device Resource Tags

Note that the tags described in this section add a control to the UI for manipulating the underlying resource. These must be contained within a UI layout container.

Many of the tags are also used to define the underlying resource in the data store. Declarations may be contained within a `<meta>` block of an OGLML or stand-alone XML file.

menugroup

This tag is used to incorporate a top-level menu group as a single component. This includes all sub-menus and parameters that would appear in a default-layout OGP menu.

Syntax

```
<menugroup mid="id" />
```

Attributes

Attribute	Values	Restrictions	Description
Mid	integer	Must be a defined top-level menu.	**menuid** of a defined menu. 0 = Status Menu 1 = Configuration Menu 2 = “Extra” Menu

menu

This tag provides a mechanism to display a standard OGP Menu in two different ways:

- Display the entire menu as a single component
- Create a clickable button to display the menu in a balloon dialog (similar to a tool tip).

Syntax

```
<menu  mid="menu-id" popup="popup-flag" oglml="oglml-flag"
       tabposition="position" GeneralAttributes />
```

Attributes

In addition to General Attributes, the following attributes may be specified to the <menu> tag:

Attribute	Values	Restrictions	Description
Mid	integer	Must be a defined OID Menu.	The static ID of the OID Menu to draw.
Popup	true	**name** attribute must also be specified.	A button with the **name** attribute as its label is the component. When pressed, the menu will appear in a balloon dialog. For more information, see WIDGET_MENU_POPUP (20) on page 32.
	false		The menu is included as a single component.
Oglml	**true**		If the referenced menu has been overridden by an OGLML page, the OGLML version of the menu will be used.
	false		The standard OGP menu without any OGLML will be used.

Attribute	Values	Restrictions	Description
Tabposition	north east **south** west	How the tabs are rendered within their quadrant is determined by the look and feel (i.e. whether the tabs fill the available space, are positioned to the left, right, or center of the space, etc.)	Specifies the placement of the tabs for any 3rd-level submenus. 

Default values shown in **bold**.

param

Displays a widget to display and manipulate a param. Must be placed within a layout container tag. If the param is an array, multiple widgets are displayed (one for each element).

Syntax

```
<param oid="oid" attributes/>

<param oid="oid" attributes>
   <constraint/>
   <constraint/>
   . . .
   <config/>
   <config/>
   . . .
</param>
```

Attributes

Attribute	Values	Restrictions	Description
Showlabel	true **false**		Display the parameter name as a label beside the parameter elements.
Oid	*String*	Must be a defined OID	The OID of the parameter to show.
*mid	*String*	Must be the static menu ID of a defined OID Menu.	This is used to determine the user rights for a parameter. The menu with the a **staticid** matching the specified **mid** is treated as the parent menu of the parameter when checking read/write rights and whether it is on a status menu or a configuration menu. If no **mid** is defined, the parameter is always rendered as though it is on a configuration menu with full read/write rights.

Attribute	Values	Restrictions	Description
Element	List of array indices separated by commas	All array elements referenced must exist in the parameter value.	By default all elements of an array parameter are returned. This attribute can be used to return a subset of the array. If a list is provided, only the elements at the provided indices are returned (note- you can specify the elements in any order). This value should either be “0” or should not be provided for a non-array parameter.
Widget	*Positive integer*	The value must be a widget hint defined for the parameter’s type	By default, the widget hint provided by the parameter will be used. This attribute can be used to override the parameter’s widget hint with another one.
Expand	true **false**	Only applies to radio and toggle button parameters.	Return each radio or toggle button created by a choice constraint as a separate element.
Constrainttype	INT_CHOICE or eo://*external-object-OID*	Can only be applied to parameters that already use choice constraints.	Allows a device developer to override the choice constraint defined in the OGP Parameter Descriptor. The parameter must either contain the available choices in **constriant** tags inside of the **param** tag or an external object URL pointing to an external object that contains an integer choice constraint.
Onchange	ogScript String		The provided snippet of ogScript is triggered when the parameter value changes. A **ParamScriptable** object named `this` is created within the context of the **onchange** to view and manipulate the param.
Relative	true		Parameter is interpreted as a relative parameter within a widget. The widget instance’s **baseOID** will be prefixed to the param OID to create a fully-qualified OID.
	false		

Default values shown in **bold**.

*mid is optional but its use is **strongly** recommended for User Rights Management support.

constraint

Overrides the choice constraint for a parameter. For `INT_CHOICE` constraints, the integer value is defined with the *key* attribute and the text to display is the text content of the tag. For `STRING_CHOICE` constraints, each constraint tag contains a value to populate a combo-box drop-down.

The parameter must have a constrainttype of `INT16_CHOICE`, `INT32_CHOICE` or `STRING_CHOICE`.

Syntax

```
<constraint key="choice1-key">choice1-value</constraint>
```

Attributes

Attribute	Values	Restrictions	Description
Key	*Integer*	Not required for STRING_CHOICE constraints	Numeric assignment of current enumerated choice.
Value	*String*		Text name for the current enumerated choice

Examples

The following constraint provides an enumerated choice:

```
<param constrainttype="INT_CHOICE" name="Channel" oid="0x503"
type="INT16_PARAM">
  <constraint key="0">Channel 01</constraint>
  <constraint key="1">Channel 02</constraint>
  <constraint key="2">Channel 03</constraint>
  <constraint key="3">Channel 04</constraint>
</param>
```

The following constraint provides a list of selections for a STRING parameter:

```
<param constrainttype="STRING_CHOICE" name="Name" oid="0x504"
type="STRING">
  <constraint>Jeremy Clarkson</constraint>
  <constraint>James May</constraint>
  <constraint>Richard Hammond</constraint>
  <constraint>The Stig</constraint>
</param>
```

buttonbar

Creates the button bar containing the “Refresh”, “Upload”, “Reboot”, and “Close” buttons. Normally this appears at the bottom of a Device Tab. Only a single instance of this tag is permitted per OGLML document.

Syntax

```
<buttonbar />
```

Attributes

None.

Example

The following displays the button bar:

```
<buttonbar/>
```

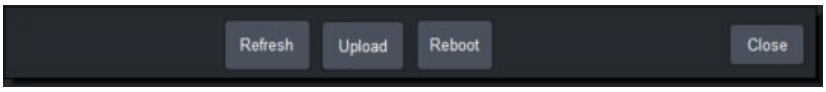


Figure 84 - <buttonbar/> tag

editor

Inserts the editor UI of another device node from the DashBoard Tree into the current container. The `editor` tag may insert either the full editor UI or a compact summary.

Syntax

```
<editor objectid="object-id" template="template-style" widgetroot="root-flag" />
```

Attributes

Attribute	Values	Restrictions	Description
Objected	*String*		ID of the device node to insert
Template	summary		Inserts a summary panel for the device.
Widgetroot	*Boolean*		Everything inside of the editor must be kept together. Individual elements cannot be dragged out to other panels.

Example

The following inserts the full UI for device with id `00.0f.9b.00.00.26(Slot 0)MFC-8310`:

```
<editor objectid="00.0f.9b.00.00.26&lt;br&gt;Slot 0&lt;br&gt;MFC-8310"
widgetroot="true" />
```

The following inserts a summary panel for the device:

```
<editor objectid="00.0f.9b.00.00.26&lt;br&gt;Slot 0&lt;br&gt;MFC-8310"
template="summary" widgetroot="true" />
```

Figure 85 – Summary Editor

summary

Creates the standard card status panel with card name, online state, and overall card status.

Syntax

```
<summary />
```

Attributes

None.

Example

The following displays the summary panel for a device:

```
<summary/>
```

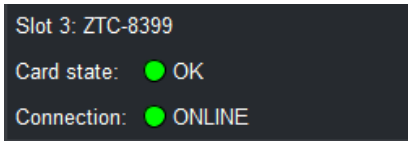


Figure 86 - <summary/> tag

statuscombo

Display a status icon for a single or multiple items from the DashBoard Tree View. When the status icon is clicked, a list of tree nodes is expanded; these nodes can be then clicked to open the editor for that node.

This is largely intended to be created by dragging/dropping items from the DashBoard Tree View or Advanced Tree View into a PanelBuilder CustomPanel document.

A hierarchy of <treeobject/> elements with the same attributes allows you to create combined status items.

Syntax

```
<statuscombo attributes>
    <treeElement name="node-name" objectid="node-id"/>
    <treeElement name="node-name" objectid="node-id"/>
    . . .
</statuscombo>
```

Attributes

Attribute	Values	Restrictions	Description
Objected	String	Must be the node-ID of a node in DashBoard tree view	The **node-id** of the element in the tree to display. If the object has children, they are automatically shown under the node.
Name	String		The display name of the item.

Example

The node-id of a node in the Tree View may be obtained by right-clicking the node and selecting "View Connection Settings".

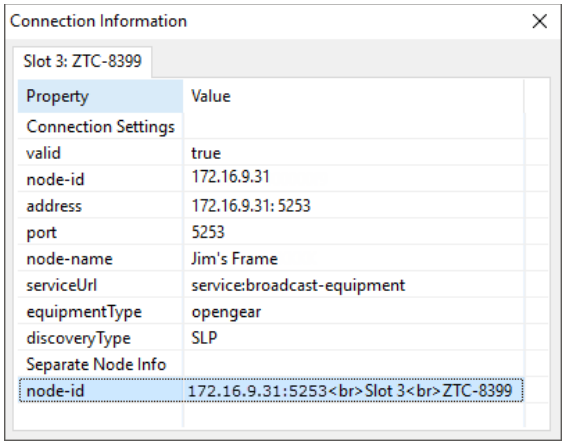


Figure 87 – Connection Settings

The following code creates a **statuscombo** with 2 nodes:

```
<statuscombo grid="false" left="448" name="Favorite Cards" top="118">
   <treeElement name="Slot 3: ZTC-8399"
objectid="172.16.9.31:5253&lt;br&gt;Slot 3&lt;br&gt;ZTC-8399"/>
   <treeElement name="Slot 5: SRA-8602"
objectid="10.1.9.36:5253&lt;br&gt;Slot 5&lt;br&gt;SRA-8602"/>
</statuscombo>
```

The result appears in DashBoard as:

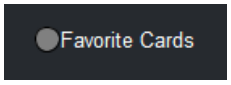


Figure 88 – statuscombo

When clicked, it expands as follows:

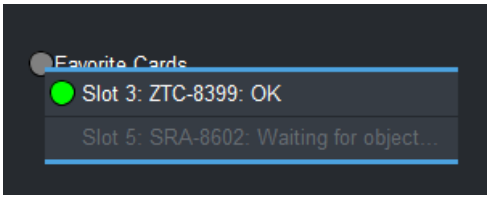


Figure 89 – statuscombo expanded

If the specified **treenode** has child nodes, it will appear as follows:

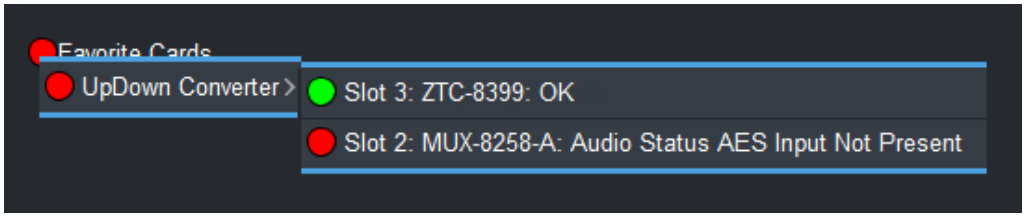


Figure 90 – statuscombo with child nodes

Macro Expansion

DashBoard includes several pre-defined macros which expand into specific useful information. The following macros are supported:

Macro	Description
%frame%	Expands to the node-id of the current frame
%device%	Expands to the node-id of the current device or card node
%slot%	Expands to the node-id of the specified slot within the current frame
%value%	Expands to a parameter's value
%widget%	Expands to a widget's id
%const%	Expands to a lookup value
%baseoid%	Expands to a widget's baseOID
%fully-qualified-id%	Expands to the full element id hierarchy
%panel-path%	Expands to the path of the current CustomPanel
%app-path%	Expands to the DashBoard installation directory
%id%	Expands to the id of the current component
%eval[ogscript]%	Performs a regular expression expansion

%frame%

Expands to the node-id of the frame within the current context.

Syntax

```
%frame%
```

Example

```
<label name="frame node-id is %frame%"/>
```

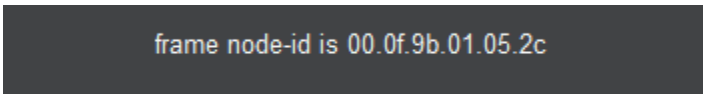


Figure 91 - %frame% macro

%device%

Expands to the node-id of the current device within the current context.

Syntax

```
%device%
```

Example

```
<label name="device node-id is %device%"/>
```

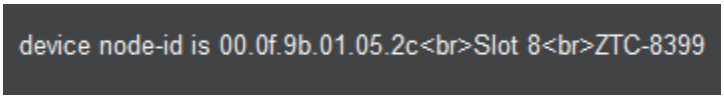


Figure 92 - %device% macro

%slot%

Expands to the node-id of the specified slot within the frame in the current context.

Syntax

```
%slot slot-number%
```

Parameters

Parameter	Values	Restrictions	Description
slot-number	*Integer*	Must be a valid slot number within the current frame	Slot number of the device whose **node-id** is to be returned.

Example

```
<label name="slot 2 node-id is %slot 2%"/>
```

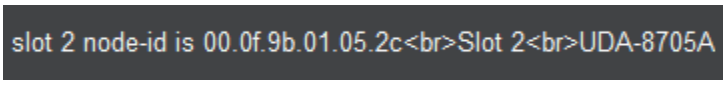


Figure 93 - %slot% macro

%value%

Expands to the value of a specified parameter.

Syntax

`%value ['param-oid'][element]%`

Parameters

Parameter	Values	Restrictions	Description
param-oid	*String*		The OID of the parameter whose value is returned
element	*Integer*		The array index to return. For non-array parameters this must be set to `0`.

Example

The following displays the value of a parameter:

```
<label name="the value of myParam is %value['myParam'][0]%"/>
```

The following example utilizes the `%value%` macro to allow the value of one parameter to specify which parameter to process. The parameter `OIDName` specifies the OID of the parameter which is displayed in the line below. Note that when the parameter `OIDName` is changed, it is necessary to manually reload the elements which display the results (`label1` and `label2`), as the `%value%` macro is expanded only when the control is rendered.

```
<params>
   <param name="OID Name" oid="OIDName" type="STRING" value="testOID2"/>
   <param name="test OID1" oid="testOID1" type="STRING" value="Fred"/>
   <param name="test OID2" oid="testOID2" type="STRING" value="George"/>
</params>

<abs>
   <param left="382" oid="params.OIDName" widget="3" width="243">
      <task tasktype="onchange">
         ogscript.reload ("label1");
         ogscript.reload ("label2");
      </task>
   </param>
   <label id="label1" left="382" name="The value of
%value['OIDName'][0]% is"/>
   <param id="label2" left="575" oid="%value['OIDName'][0]%"
widget="1"/>
</abs>
```

Figure 94 - %value% macro

%widget%

Expands to the **id** of widget within the current context.

Syntax

```
%widget%
```

Example

If used within a widget, the following displays the widget's ID:

```
<label name="the value of myParam is %widget%"/>
```

%const%

Expands to the value of a lookup. The lookup must have a specified **id**.

Syntax

```
%const['id']['key']%
```

Parameters

Parameter	Values	Restrictions	Description
id	String	Must be an id defined in a `<lookup>` tag	ID of the lookup tag.
key	String	Must be a valid key within the specified lookup.	Key within the lookup tag whose value will be returned.

Example

Given the following lookup:

```
<lookup id="family" scope="private">
    <entry key="father">Homer Simpson</entry>
    <entry key="son">Bart Simpson</entry>
    <entry key="mother">Marge Bouvier-Simpson</entry>
    <entry key="daughter">Lisa</entry>
    <entry key="baby">Maggie</entry>
</lookup>
```

The following code will display the label "The son is Bart Simpson".

```
<label "The son is %const['family']['son']%"/>
```

%baseoid%

Expands to the value of the baseOID attribute of the current widget.

Syntax

```
%baseoid%
```

Example

If you have a widget with a baseoid of params.audio.channels.1 with parameters for signal presence, EQ, etc., you could attach change handlers to them as follows:

```
<ogscript handles="onchange" oid="%baseoid%.eq" element="0">
ogscript.debug('EQ has changed for %baseoid%:' + this.getValue());
</ogscript>
```

%fully-qualified-id%

Expands to the fully-qualified id of the current context. If the current context is nested within other contexts, the hierarchy is expressed, separated by ".". Note that only containers with a specified id are included in the expansion.

Syntax

```
%fully-qualified-id%
```

Example

```
<abs id="abs1">
    <abs>
        <abs id="abs2">
            <label "The fully qualified ID is %fully-qualified-id%"/>
        </abs>
    </abs>
</abs>
```

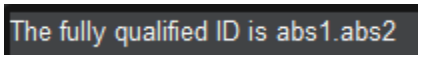


Figure 95 - %fully-qualified-id% macro

%panel-path%

Expands to the folder path which contains the current OGLML document.

Syntax

```
%panel-path%
```

Example

```
<label name="panel path is %panel-path%" />
```

Figure 96 - %panel-path% macro

%app-path%

Expands to the folder path which the current instance of DashBoard is installed.

Syntax

```
%app-path%
```

Example

```
<label name="DashBoard is installed in %app-path%" />
```

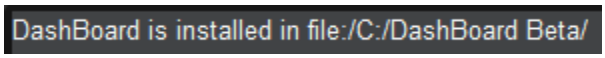


Figure 97 - %app-path% macro

%id%

Expands to the id of the current context.

Syntax

```
%id%
```

Example

```
<label height="62" left="0" name="Click to see my context's ID"
style="txt-align:center;" top="0" width="291">
   <task tasktype="ogscript">ogscript.debug('My Context\'s ID is
"%id%"');</task>
</label>
```

%eval[ogscript]%

Evaluates the ogscript and replace the %eval[ogscript]% with the value returned by the script.

Syntax

```
%eval [ogscript]%
```

Example

```
<label height="62" left="0" name="%eval[var text = ''; for (var i = 0; i
&lt; 10; i++){text += i + ' ';} text.trim();]%" style="txt-align:center;"
top="0" width="291"/>
```

ogScript Reference

About ogScript

Ross Video ogScript is a programming language developed by Ross Video to interact with DashBoard-enabled devices.

It also enables you to add functionality and logic to custom panels you create in DashBoard.

Ross Video ogScript uses JavaScript functions, syntax, and primitive object types. To enable CustomPanel developers to interact with panels and devices, ogScript adds some new global objects to JavaScript. Most JavaScript works in ogScript scripts, although you might run across an occasional item that does not work.

For information about ogScript objects and functions, refer to the topics in this section. For information about JavaScript commands and syntax, search for "JavaScript Reference" on the World Wide Web.

This section contains information about ogScript objects and functions. It includes the following major sections:

- ogscript Object
- params Object
- ParamScriptable Object
- rosstalk Object
- rosstalkex Object
- robot Object
- vdcp Object
- nkScript Object

JavaScript

Ross Video ogScript is a programming language developed by Ross Video to interact with DashBoard-enabled devices. It uses JavaScript functions, syntax, and primitive object types. To enable CustomPanel developers to interact with panels and devices, ogScript adds some new global objects to JavaScript. Most JavaScript works in ogScript scripts, although you might run across an occasional item that does not work.

For information about ogScript objects and functions, refer to the sections in this guide. For information about JavaScript commands and syntax, search for "JavaScript Reference" on the World Wide Web.

Commonly Used Functions

Ross Video recommends that you first learn the following commonly used functions:

Ogscript

- debug
- rename

params

- getValue
- setValue

Functions Set in the User Interface

Functions in the following objects are typically set through a user interface:

- rosstalk Object
- robot Object
- vdcp Object
- multiSetScriptable Object
- nkScript Object

multiSetScriptable Object

In ogScript, use the **multiSetScriptable** object to change the values of multiple parameters at once.

To create a multiSetScriptable object, use:

```
params.createMultiSet();
```

For example:

```
params.createMultiSet ('This is a message');
```

The following table lists the functions of the multiSetScriptable object. Detailed descriptions appear after the table. If you are reading this document on-screen, click a function name in the table to view its description.

Function	Parameters	Returns	Description
execute	N/A	Boolean	Execute the multiSet. Returns true if execution was successful; otherwise false.
setAllValues	Object [OID], Object [] [Values]	N/A	Update all values of the parameter with the specified OID using the values from the object array.
setValue	Object [OID], Int [Index], Object [Value]	N/A	Update the specified index using the value object.

nkScript Object

In ogScript, use the nkScript object to control NK Router OGLML tags used in Switchboard virtual control panels. Functions in the nkScript object are usually set through a user interface.

The nkScript global object is only accessible in OGLML contexts that are declared as having a NK Router context type or are beneath such a context in the OGML document hierarchy.

To call a general-purpose function, use:

```
nkscript.function name (parameters);
```

For example:

```
nkscript.setHost (Server01);
```

The following table lists the functions of the nkscript object.

Function	Parameters	Returns	Description
convertCommaSeperatedLevelsToMask	String [Levels], Boolean [SearchTags]	Long Levelmask	Allows conversion of a list of levels to the appropriate level mask. Level mask is a bit field where you can have up to 32 levels set 'on' at a time. SearchTags should always be 'true'.
doSwitch	N/A	Boolean	Equivalent of calling: doSwitch(getActiveDst(), getActiveSrc(), getLevelMask());
doSwitch	Int [Dst], Int [Src], Long [Levels]	Boolean	Do a switch on the active IPS to route the given dst to the given src on the given levels.
doSwitchWithLabels	String [Destination], String [Source], String [Levels]	Boolean	Allows you to switch between levels by name.
getActiveDst	N/A	Int	Get the active dst number (0-indexed). Returns -1 if there is no active destination.
getActiveDstName	N/A	String	Get the name of the active dst (from the switchboard configuration). Returns null if there is no active destination.
getActiveIPS	N/A	String	Get the serial number of the active IPS
getActiveIPSName	N/A	String	Get the name of the active IPS
getActiveSrc	N/A	Int	Get the active src number (0-indexed). Returns -1 if there is no active source.
getActiveSrcName	N/A	String	Get the name of the active src (from the switchboard configuration). Returns null if there is no active source.
getActiveSystem	N/A	NKSystem	Get the currently active NKSystem.
getDstName	String [Source]	String	Get the destination name of the given source.
getLevelMask	N/A	Long	Get the current level mask (as a bit field) Level mask is a bit field where you can have up to 32 levels set 'on' at a time.

Function	Parameters	Returns	Description
getLevelName	String [Source]	String	Get the level name of the given source.
getProtectStatus	String [Destination], String [Levels]	Boolean	Get the protect status of the destination level.
getSrcName	String [Source]	String	Get the source name.
getStatus	String [Destination], String [Level]	Int	Get the status of the given destination level.
isLevelActive	Int [Level Num]	Boolean	Is the current level active. Equivalent to asking: levelMask & (1 << levelNum) != 0;
isMCFlag	N/A	Boolean	Is the Machine Control flag set.
isProtected	N/A	Boolean	Is the active destination protected.
isProtected	Int [Destination], Long [Levels]	Boolean	True if the given destination is protected; otherwise false.
isProtectedByMe	N/A	Boolean	Is the active destination protected by this virtual panel.
isSrcActive	Int [Src]	Boolean	Is the given source active on the active destination any level.
isSrcActive	Int [Dst], Int [Src], Long [Levels]	Boolean	Is the given source active on the given destination on the given level mask.
isVirtual	N/A	Boolean	Is virtual routing in use (for switch commands and status requests).
setActiveDst	Int [Dst]	N/A	Set the active destination (0-indexed).
setActiveIPS	String [Serial]	Boolean	Set the IPS with the given serial number as the active IPS to receive commands and send status. Deactivate any currently active IPS.
setActiveSrc	Int [Src]	N/A	Set the active source (0-indexed).
setLevelActive	Int [Level Num], Boolean [Active]	Boolean	Set the given level as active.
setLevelMask	Long [Level Mask]	N/A	Set the complete level mask bitfield.
setMCFlag	Boolean	Boolean	Set the Machine Control flag to true or false.
setProtected	Boolean	Boolean	Request the router to protect the active destination.
setVirtual	Boolean	Boolean	Set virtual routing on/off for switch commands and status requests.

Function	Parameters	Returns	Description
verifyConfiguration	N/A	Boolean	Re-activate the current IPS.

asyncExec

Executes a function outside of the UI current thread.

This is especially useful for operations that take time to complete. You can use **`asyncExec`** to run such operations while continuing to execute the rest of your tasks.

Syntax

```
ogscript.asyncExec(function);
ogscript.asyncExec(function, delay);
```

Parameters

Parameter	Type	Required	Description
function	Function reference	Yes	Reference to the function to be executed. Can also be an anonymous function.
delay	Long	No	Delay (in milliseconds) before executing the function. **Note**: If the asyncExec thread is busy executing another task at the specified time, the function will execute as soon as the asyncExec thread is free.

Returns

N/A

Example 1

This example displays two buttons. Each button runs a function named `reallyLongFunction`, which increments a parameter named `Number` until it reaches `500000`. The `Number` parameter is displayed in the top left corner of the panel.

The button labeled **Start Count** executes the function normally. No other tasks can start while the count proceeds. The display of the `Number` parameter isn't refreshed until the count is complete.

The button labeled **Start Count Using asyncExec** executes the function asynchronously. The panel can start other tasks while the count proceeds. The display of the `Number` parameter is updated as its value changes.

The interface for this example appears as follows:

asyncFTPListFiles

Asynchronously gets a list of all files at a specified directory on an FTP server. Returns an array of FTPFile objects, on which the following methods can be called:

- file.getName()

- file.getTimestamp() (is a java.util.Calendar object)
- file.getSize()
- file.isFile()
- file.isDirectory()

Syntax

```
ogscript.asyncFTPListFiles(host, port, username, password, path, callback);
ogscript.asyncFTPListFiles(host, port, username, password, path, fileName,
callback);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host address
Port	Int	Yes	Host port
Username	String	Yes	Login username
Password	String	Yes	Login password
Path	String	Yes	Source path
fileName	String	Optional	Source file name, can contain the ".*" wildcard.
callback	Function reference	Yes	Callback function. Invoked after FTPListFiles is complete. Callback is passed success, list of files, and exception

Returns

Returns an array of FTPFile objects.

FTPFile class is used to represent information about files stored on an FTP server.

Example 1

Outputs the file and directory names located at the directory '/Media/Sports/Sens' on an FTP server.

The source code for this example is as follows:

```
function outputResults(success, files, exception)
{
    if (!success)
    {
      ogscript.debug("NO SUCCESS");
      return;
    }
    else if (files != null)
    {
    /*
      * files[i].getName()
      * files[i].getTimestamp  // returns java.util.Calendar
      * files[i].getSize()     // returns file size in bytes
      * files[i].isFile()      // returns true if the file is a File (not a
      directory)
      * files[i].isDirectory() // returns true if the file is a Directory
      */
    ogscript.debug("GOT " + files.length + " FILES");
```

```
    for (var i = 0; i < files.length; i++)
      {
      var jsTime = (new Date(files[i].getTimestamp().getTimeInMillis()));
      if (files[i].isDirectory())
      {
        • ogscript.debug("GOT DIRECTORY: " + files[i].getName());
      }
      else
      {
        • ogscript.debug("GOT FILE: " + files[i].getName() + " " + jsTime);
      }
    }
  }
}
ogscript.asyncFTPListFiles('CAPRICABVS', 21, 'blackstorm', 'blackstorm',
'/Media/Sports/Sens', outputResults);
```

asyncHTTP

Send an asynchronous request to the given URL. Call the given function when the request has completed. The data retrieved from the HTTP request is passed as a string as the first variable in the method.

If the MIME type of the HTTP response is image or binary, the result will be a byte array containing what is fetched.

Syntax

```
ogscript.asyncHTTP(URL, Method, Content_Type, Data, Callback);
ogscript.asyncHTTP(URL, Method, Content_Type, Data, Callback,
Include_Response_Code);
```

Parameters

Parameter	Type	Required	Description
URL	String	Yes	Http url
Method	String	Yes	The method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions.
Content_Type	String	Yes	The content type of the request.
Data	Object	Yes	Data can be a string, byte array, XML, or JSON object
Callback	Function reference	Yes	Function to call after the request completes.
Include_Response_Code	Boolean	No	True to include response code; otherwise false.

Returns

N/A

Example 1

Coming soon.

asyncPost

Send an asynchronous post to the given URL. Call the given function when the post has completed. The data retrieved from the HTTP Post is passed as a string as the first variable in the method.

If the MIME type of the HTTP response is image or binary, the result will be a byte array containing what is fetched.

Syntax

```
ogscript.asyncPost (URL, HTTP Post Data, Callback Function);
ogscript.asyncPost (URL, HTTP Post Data, Callback Function, Include Response);
```

Parameters

Parameter	Type	Required	Description
URL	String	Yes	URL to send a post.
HTTP Post Data	String	Yes	Post to send to the specified URL.
Callback Function	Function	Yes	Function to call after the post completes.
Include Response	Boolean	No	If true, result is a JSON Object **{** **responseCode = HTTP RESONSE CODE,** **contentType = HTTP MIME TYPE** **url = URL Requested** **bytes= BYTES RECEIVED** **}** Otherwise, it is content fetched over HTTP parsed as though it's a string (as before).

Returns

N/A

Example

Coming soon.

closePanel

Closes the DashBoard panel that the command was called from.

Syntax

```
ogscript.closePanel();
```

Parameters

N/A

Returns

N/A

Example 1

```
// Close the panel that command is called from
ogscript.closePanel();
```

colorToHSL

Converts an RGB color to an HSL color

Color parameter must be either an integer representation of an RGB color, or a string representation of an RBG color.

Syntax

```
ogscript.colorToHSL (int color);
ogscript.colorToHSL (string color);
```

Parameters

Parameter	Type	Required	Description
color	Int	Yes	Integer representation of RGB color (in decimal)
color	String	Yes	String representation of RGB color (in hex)

Returns

Returns a float array containing the HSL version of the color parameter.

Example 1

ogscript.colorToHSL(16777215);

ogscript.colorToHSL('#FFFFFF');

Will both return HSL for the color white

copyText

Copies text to the operating system's clipboard.

Syntax

```
ogscript.copyText(text);
```

Parameters

Parameter	Type	Required	Description
Text	String	Yes	Text to be copied to clipboard.

Returns

N/A

Example 1

```
// Will set the system clipboard to the text "Hello World!"
ogscript.copyText('Hello World!');
```

createAMPSender

Creates a library of commands for controlling video servers using the Advanced Media Protocol (AMP).

Syntax

```
ogscript. createAMPSender();
```

Parameters

N/A

Returns

Returns an AMPCommands object.

Example

```
// To create and store a new AMP sender, you can use
var ampSender = ogscript.createAMPSender();
```

createAsyncExec

Creates a new asynchronous thread with the specified ID.

Syntax

```
ogscript.createAsyncExec(thread ID);
```

Parameters

Parameter	Type	Required	Description
Thread ID	String	Yes	Desired ID for new thread

Returns

Returns an asynchronous thread with the specified ID if it was created, null otherwise.

Example 1

```
// Create and save an asynchronous thread with the id "new_thread"
var asyncThread = ogscript.createAsyncExec("new_thread");
```

createByteArray

Creates a byte array with the specified length.

Syntax

```
ogscript.createByteArray(length);
```

Parameters

Parameter	Type	Required	Description
length	Int	Yes	Length of the byte array (Must be greater than or equal to zero)

Returns

Returns a byte array of the desired length.

Example 1

```
// Create a new byte array with length of 10
var byteArray = ogscript.createByteArray(10);
```

createFileInput

Creates a new FileInputParser on a File object. Can call close, getSize, and isClosed on the FileInputParser object.

Syntax

```
ogscript.createFileInput(fileObject);
```

Parameters

Parameter	Type	Required	Description
File Object	Object	Yes	Destination file object

Returns

Returns a FileInputParser object, on which the functions close(), getSize(), and isClosed() can be called.

Example 1

```
// If we have a file object called fileObject, and we want to debug output it's size:
var fileInputParser = ogscript.createFileInput(fileObject);\
var fileSize = fileInputParser.getSize();
ogscript.debug(fileSize);
```

createListener

Create a new listener with its own ID, settings, and task.

Syntax

```
ogscript.createListener(id, listenerSettings, listenerTask);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID for new listener
Listener Settings	Object	Yes	Settings for new listener
Listener Task	Function reference	Yes	Task for new listener

Returns

Returns an IServerWithClose object, which contains functions close, setPort, start, and stop.

Example 1

ogscript.createListener('listener1', listener1Settings, listener1Task);

createVDCPSender

Creates a library of commands for using the video disk control protocol (VDCP).

Syntax

```
ogscript.createVDCPSender();
```

Parameters

N/A

Returns

Returns a VDCPCommands object.

Example

// Create a new VDCP Sender

var vdcpLibrary = ogscript.createVDCPSender();

focus

Sets the focus to a component with a specified ID.

Syntax

```
ogscript.focus(id);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	Component ID to focus

Returns

N/A

Example 1

Coming soon.

ftp

Saves an object to a destination path on an FTP server. Useful to store statistics, images, and any other data on a server.

Syntax

```
ogscript.ftp(host, port, username, password, destPath, destName, binary,
data);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host address
Port	Int	Yes	Host port
Username	String	Yes	Login username
Password	String	Yes	Login password
Destination Path	String	Yes	Data destination path
Destination Name	String	Yes	Data destination name
Binary	Boolean	Yes	True if data is binary (.jpg, .mp3), false if data is ascii (.txt, .html).
Data	Object	Yes	Data to be transferred

Returns

Returns an FTPResponse object which contains a boolean 'success', an object 'data', and an exception 'ex'.

Example 1

ogscript.ftp('localhost', 567, 'username', 'password', '/dashboard/', 'stats.txt', false, statTextObject);

ftpGet

Gets a file from the source path on an FTP server, and stores it in the destination object. Useful to grab statistics, images, or any other data from a server.

Syntax

```
ogscript.ftpGet(host, port, username, password, srcPath, srcName, binary,
destination);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host address
Port	Int	Yes	Host port
Username	String	Yes	Login username
Password	String	Yes	Login password
Source Path	String	Yes	Source path
Source Name	String	Yes	Source name
Binary	Boolean	Yes	True if data is binary (.jpg, .mp3), false if data is ascii (.txt, .html).
Destination File	Object	Yes	Destination file object

Returns

Returns an FTPResponse object which contains a boolean 'success', an object 'data', and an exception 'ex'.

Example 1

// Get a file stats.txt (ascii) from a directory "dashboard" on an ftp server

ogscript.ftpGet('localhost', 567, 'username', 'password', '/dashboard/', 'stats.txt', false, destinationObject)

ftpListFiles

Gets a list of all files at a specified directory on an FTP server.

Returns an array of FTPFile objects, on which the following methods can be called:

- file.getName()
- file.getTimestamp() (is a java.util.Calendar object)
- file.getSize()
- file.isFile()
- file.isDirectory()

Syntax

```
ogscript.ftpListFiles(host, port, username, password, srcPath);
ogscript.ftpListFiles(host, port, username, password, srcPath, fileName);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host address
Port	Int	Yes	Host port
Username	String	Yes	Login username
Password	String	Yes	Login password
Source Path	String	Yes	Source path
File Name	String	No	Source file name

Returns

Returns an array of FTPFile objects.

FTPFile class is used to represent information about files stored on an FTP server.

Example 1

```
// Gets a list of all files under the /photos/ directory on the FTP server
ogscript.ftpListFiles('localhost', 557. 'username', 'password', '/photos/');
```

getApplicationPath

Returns the path to the installation location of DashBoard.

Syntax

```
ogscript.getApplicationPath();
```

Parameters

N/A

Returns

Returns a String representation of the path to the DashBoard installation location.

Example 1

```
// Get and store dashboard installation loca
var dashboardLocation =  ogscript.getApplicationPath();
```

getAsyncExecById

Finds and returns an asynchronous thread with a specified ID.

Syntax

```
ogscript.getAsyncExecById(thread id);
```

Parameters

Parameter	Type	Required	Description
Thread ID	String	Yes	ID of desired thread.

Returns

Returns an asynchronous thread with the specified ID if one was found; otherwise null.

Example 1

// If we have an asynchronous thread with the id "thread1", we can get it using

ogscript.getAsyncExecById('thread1');

getBrowserById

Finds and returns a browser object with a specified ID. If browser with specified ID was not found, returns null.

Syntax

```
ogscript.getBrowserById(BrowserID);
```

Parameters

Parameter	Type	Required	Description
Browser ID	String	Yes	ID of browser to look for.

Returns

If found, returns a browser element with the specified ID, null otherwise.

Example

// Get the browser with the ID "TestBrowser"

ogscript.getBrowserById("TestBrowser");

getContextId

Gets and returns the current context ID if it exists.

Syntax

```
ogscript.getContextId();
```

Parameters

N/A

Returns

Returns a string representation of the context ID if it exists; otherwise null.

Example 1

```
// Get the current context ID
var contextID = ogscript.getContextId();
```

getFile

Finds and returns a file at a given path.

Syntax

```
ogscript.getFile(filePath);
```

Parameters

Parameter	Type	Required	Description
filePath	String	Yes	Path to desired file

Returns

Returns the File object found at the specified path if it was found, null otherwise.

Example 1

```
// Get a file from the path "C://Users/John/Desktop/test.txt"
var file = ogscript.getFile('C://Users/John/Desktop/test.txt');
```

getFileSize

Used to find the size (in bytes) of a file at a specified path.

Syntax

```
ogscript.getFileSize(filePath);
```

Parameters

Parameter	Type	Required	Description
filePath	String	Yes	Path to desired file

Returns

Returns a long equal to the size of the file in bytes.

Example 1

```
// Save the size of the file located at "C://Users/John/Desktop/helloworld.txt"
var fileSize = ogscript.getFileSize('C://Users/John/Desktop/helloworld.txt');
```

getImageById

Finds and returns an image with a specified ID.

Syntax

```
ogscript.getImageById(imageID);
```

Parameters

Parameter	Type	Required	Description
Image ID	String	Yes	ID of desired image

Returns

Returns an image if one matching the ID was found, null otherwise.

Example 1

```
// Find and return an image with the id "image1"
ogscript.getImageById('image1');
```

getPanelPath

Gets the path of the panel the function was called by.

Syntax

```
ogscript.getPanelPath();
```

Parameters

N/A

Returns

Returns a String representation of the path to the calling panel.

Example 1

```
// If the calling panel is stored at "C:\Users\Test\DashBoard\" on the disk,
ogscript.getPanelPath();
// will return "C:\Users\Test\DashBoard\"
```

getPanelRelativeURL

Gets the full URL of a path with respect to the panel it is called from. Could be used to get the full path of an "images" or "stats" directory.

Syntax

```
ogscript.getPanelRelativeURL(path);
```

Parameters

Parameter	Type	Required	Description
path	String	Yes	Relative path

Returns

Returns a String representing the full path of the relative path with respect to the panel's path.

Example

// If we have a panel stored at C:\Users\Test\Panels\ and we store images in a // directory \Images\ located in the same \Panels\ folder that the panel itself is located in, we can // use the line

ogscript.getPanelRelativeURL('\Images\');

// to get the String "C:\Users\Test\Panels\Images\".

hslToColorString

Converts an float array containing HSL data (hue, saturation, lightness) to a color string (Color string displays the color in hexadecimal).

Syntax

```
ogscript.hslToColorString(hslFloat[]);
```

Parameters

Parameter	Type	Required	Description
HSL Float Array	Float32_Array	Yes	Float array – first element is hue, second element is saturation, third element is lightness.

Returns

Returns a hex string representation of the HSL color; if HSL float array was invalid, returns null.

Example

// If we have an hslFloat array containing 91 in index 0, 0.89 in index 1, and 0.61 in index 2

ogscript.hslToColorString(hslFloatArray);

// Returns the string "#98F442"

http

Used to fetch content from a web server or call restful API.

Syntax

```
ogscript.http(URL, method, requestContentType, dataObject, includeResponse);
```

Parameters

Parameter	Type	Required	Description
URL	String	Yes	http URL
Method	String	Yes	The method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions.
Request Content Type	String	Yes	The content type of the request.
Data Object	Object	Yes	Data can be a string, byte array, XML, or JSON object
Include Response	Boolean	Yes	True to include response; otherwise false.

Returns

Returns either string data or a JSON object.

Example 1

Coming soon.

installTimer

Create a timer with the given ID and register it in the ContextTimerManager. Start the timer after the specified delay. If requested, repeat the timer at the specified frequency. When the timer fires, run the specified ogScript function.

Syntax

```
ogscript.installTimer (Timer ID, Repeat, Delay, Repeat Delay, Task);
ogscript.installTimer (Timer ID, Repeat, Delay, Repeat Delay, Boolean, Task);
```

Parameters

Parameter	Type	Required	Description
Timer ID	String	Yes	ID of the timer to create and register in the ContextTimerManager.
Repeat	Boolean	Yes	true — repeat the timer using the specified Delay and Repeat Delay. false — only run the timer once, do not repeat the timer.
Delay	Long	Yes	Number of milliseconds to wait before starting the timer.
Repeat Delay	Long	Yes	How frequently the associated function runs, in milliseconds.
Execute in Timer	Boolean	No	If true, task will execute in timer thread
Task	Function	Yes	ogScript function to run when the timer fires.

Returns

N/A

Example 1

This example creates a label named "Time" and a button named "Install Timer". When a user clicks the "Install Timer" button, an associated task runs a function named myFunction (), which creates a timer.

It also retrieves the time value every 30 seconds, and loads it into a variable named str which is displayed on the "Time" label. The myFunction () function uses the installTimer function to create the timer and set the rate at which the time data is updated.

```
<label height="80" id="timeLabel" left="43" name="Time"  style="txt-
align:west" top="26" width="275"/>
<button buttontype="push" height="57" left="48" name="Install Timer"
top="133" width="184">
   <task tasktype="ogscript">function myFunction()
      {
         var date = new Date();
         var str = date.getHours() + ':' + date.getMinutes() + ':' +
         date.getSeconds();
         ogscript.rename('timeLabel', 'Time: ' + str);
      }
   //create a timer that starts immediately and runs myFunction  every 30
   seconds (30000 milliseconds)
   ogscript.installTimer('myTimer', true, 0, 30000, myFunction);
   </task>
</button>
```

isClosed

Will return true if the context is closed or does not exist, and false otherwise .

‘closed’ means that the tab is closed, DashBoard is closed, or the panel is reloaded.

Syntax

```
ogscript.isClosed();
```

Parameters

N/A

Returns

Returns true if the context is closed or does not exist; otherwise false.

Example

// Get if the context is closed.

var closed = ogscript.isClosed();

jsonToString

Transforms a JSON object into a String.

Syntax

```
ogscript.jsonToString(NativeObject);
```

Parameters

Parameter	Type	Required	Description
JSON native object	NativeObject	Yes	The JSON to be converted to a String

Returns

Returns a String representation of the JSON native object.

Example 1

```
// If we have a JSON object named jsonObj, we can convert it to a string using:
var jsonString = ogscript.jsonToString(jsonObj);
```

pasteText

Gets the contents of the operating system clipboard, if the contents can be represented as a string.

Syntax

```
ogscript.pasteText();
```

Parameters

N/A

Returns

Returns a String containing the contents of the system clipboard.

Example 1

```
// If the system clipboard contains the text "Hello World!"
ogscript.pasteText();
// will return a string containing "Hello World!"
```

addRemoteTrigger

Allows remote execution of a script inside of a CustomPanel through the RossTalk GPI command. The function can be removed by calling close on the object returned.

Syntax

```
ogscript.addRemoteTrigger (function);
ogscript.addRemoteTrigger (triggerID, function);
ogscript.addRemoteTrigger (triggerID, triggerName, function);
```

Parameters

Parameter	Type	Required	Description
Function	String	Yes	The function to execute, including its parameters (if any).
Trigger ID	String	Yes	String that triggers the specified function to execute.
Trigger Name	String	Yes	Shows on the button in the web UI.

Returns

Returns an object that contains one function named **`close`**. When executed, **`close`** removes the function.

Example

```
// Add a remote trigger with a function named testFunction
ogscript.addRemoteTrigger('testFunction()');
```

ogscript Object

In ogScript, use the **ogscript** object to access a library of general-purpose functions. To call a general-purpose function, use:

```
ogscript.function name(parameters);
```

For example:

```
ogscript.debug ('This is a message');
```

The following table lists the functions of the **ogscript** object. Detailed descriptions appear after the table. If you are reading this document on-screen, click a function name in the table to view its description.

Function	Parameters	Returns	Description
addRemoteTrigger	String [Trigger] String [Function]	Returns an object that contains one function named **close**. When executed, **close** removes the function.	Allows remote execution of a script inside of a CustomPanel through the RossTalk GPI command. Function can be removed by calling **close** on the object returned.
appendXML	String [Container ID] String [XML snippet]	N/A	Adds a section of OGLML code to the panel identified by the Container ID parameter. The OGLML is added during runtime and does not affect the .grid file. Valid only in <abs/> containers.
asyncExec	function	N/A	Executes a function outside of the UI current thread.

Function	Parameters	Returns	Description
asyncFTP	post port username password destPath destName binary sourceFilePath callback	N/A	Sends a file to an FTP server.
asyncFTPGet	host port username password srcPath srcName binary destFilePath or null callback	N/A	Retrieves a file from FTP server.
asyncPost	String [URL] String [HTTP Post Data] Function [Callback Function] Boolean [include response]	N/A	Send an asynchronous post to the given URL.
cancelTimer	Timer ID	N/A	Cancel, stop and clean-up, a timer with the given ID.
copyByteArray	src offset length	byte array	Creates a full or partial copy of a byte array.
createByteArray	length	an empty byte array	Creates an empty byte array of a specified size.
createFileInput	String [File path]	FileInputParser (like MessageParser but with getSize(), close(), and isClosed()	Access a file as a byte array with the same capabilities as MessageParser to read raw bytes
createFileOutput	String [File path] Boolean [appendToExistingfile]	FileOutputBuilder, which is same as MessageBuilder with added functions for clear() (overwrite file), close(), getSize(), flush(), and isClosed()	Create a new file or append to an existing file. Instead of saving XML or string data, gives access to write raw bytes (or strings, or shorts, or ints, etc.). Also gives the ability to append to a file. Once open, it does not close the file until the panel is closed or close() is called. This is handy for logging.
createMessageBuilder	N/A	Returns a MessageBuilder object used to build byte arrays (generally for creating network messages).	Creates a message builder, which enables you to construct a message.
createMessageParser	messageBytes	Returns a MessageParser object (generally used to parse the various pieces of messages received over the network).	Creates a message parser, which enables you to parse a message.

Function	Parameters	Returns	Description
debug	String [Message]	N/A	Write a string to the openGear Debug Information View.
fireGPI	String [trigger] String [state] Boolean [global]	N/A	Sends Trigger GPI string [trigger] to execute component task lists. Sends optional [state] data string, which can be read by the script. When [global]' value is 'true', applies to all open panels. When [global] is 'false', applies only to the current active panel.
getAllById	String [Object ID]	Object []	Get all Objects accessible in the current context that have the associated ID.
getAttribute	String [Attribute ID]	Object	Get an attribute registered in the context with the given ID.
getBuild	N/A	DashBoard version number (same value that appears in Help>About DashBoard)	Gets the version of DashBoard running the panel.
getComponentsById	String [Object ID]	Component []	Get all Java Swing components accessible in the current context that have the associated ID.
getCurrentUser	N/A	String	Returns the username of the current DashBoard user.
getIncludeById	String [Include ID]	IncludeReloadableContainer	Returns the first include with the given ID.
getListenerById	ID	getListenerById returns an object representing the listener. This object has three public methods you can call: start(), stop(), and isStarted(). The return depends on which of the three methods is used: If the start() method is used, return is true if the listener started successfully; otherwise false. If the stop() method is used, return is true if the listener stopped successfully; otherwise false. If the isStarted() method is used, return is true if the listener is started; otherwise false.	Starts or stops a listener. Can also check whether a listener is started.
getModificationDate	String [File Path]	Returns the time the specified file was last modified, in Unix Epoch time (also known as POSIX time), as a LONG value.	Retrieves the time the specified file was last modified.
getObject	String [Key]	String	Retrieves stored object

Function	Parameters	Returns	Description
getPosition	String [ID]	JAVA point object with point.x and point.y available.	Retrieves the horizontal (x) and vertical (y) position of the object, in pixels.
getPrivateString	String [Lookup ID] String [Key]	String	Get a string defined in the lookup table with the specified lookup ID.
getScopedAttribute	String [Scope Name] String [Attribute ID]	Object	Get an attribute in the named scope that has the given ID. Scopes are often internally defined by DashBoard.
getSize	String [ID]	Dimension object with d.width and d.height available	Retrieves the width and height of the specified panel object.
getString	String [Key]	String	Get a string defined in the global lookup table.
getTimerManager	N/A	ContextTimerManager	Get the timer manager for the context to access timers and perform operations on selected timers. This function includes several methods.
hide	String [ID]	N/A	Hide the popup with the specified ID.
installTimer	String [Timer ID] Boolean [Repeat] Long [Delay] Long [Repeat Rate] Function [Task]	N/A	Create a timer with the given ID and register it in the ContextTimerManager. Start the timer after the specified delay, repeat the timer if requested at the specified rate. When the timer fires, run the specified ogScript function.
isTimerRunningisTimerRunning	String [Timer ID]	Boolean	Report whether or not a timer exists and is in the “running” state. true — a timer with the given ID exits and is in the “running” state. false — a timer with the give ID does not exist or is not in the “running” state.
parseXML	String	org.w3c.dom.Document	Parse and return an XML document using the org.w3c.dom.Document API.
putObject	String [Key] String [Value]	N/A	Defines a stored object.
putPrivateString	String [LookupID] String [Key] String [Value]	N/A	Add or replace a string in a private lookup table.
putString	String [key] String [value]	N/A	Add or replace a string in the global lookup table.
reload	String [ID]	Null, if null is provided as the ID.	Rebuild the UI element with the given ID. If no ID is provided, rebuilds entire document.
rename	String [ID] String [Name]	N/A	Modify the text for a tab name, button, or label with the specified ID.

Function	Parameters	Returns	Description
reposition	String [ID] Integer [x position] Integer [y position]	N/A	Moves object to specified XY pixel location
repositionByPercent	String [ID] Integer [percent x] Integer [percent y] Boolean [center x] Boolean [center y]	N/A	Moves object to the specified location, as percentage of the container width or height. Center x and center y, when true, center the object at the location horizontally (x only), vertically (y only), or both (x and y).
reveal	String [ID]	N/A	Open a popup with the specified ID, or bring the tab with the specified ID to the foreground.
runXPath	String [XPath] XML Document or XML Element	NodeList	Execute the given XPath command on the given Document or Element and return the results as a NodeList.
saveToFile	path data overwrite	Returns true, if data is written successfully; otherwise false.	Saves data to a file. This function is typically used to save a byte array, string, or XML document to a file.
sendUDPAsBytes	String [Host] Integer [Port] Byte[] [Data]	N/A	Send the given Data bytes to the provided Host/Port through UDP.
sendUDPString	String [Host] Integer [Port] String [Data]	N/A	Convert the given Data string to UTF-8 bytes and send them to the provided Host/Port through UDP.
setAnchorPoints	String [ID] Boolean [top] Boolean [left] Boolean [bottom] Boolean [right]	N/A	Specifies how an object moves if the user interface is resized for different monitor and window sizes. Anchors or releases an object to/from the top, left, bottom, or right sides of its container.
setSize	String [ID] String [width] String [height]	N/A	Resizes a panel object to the specified size. Valid only in <abs/> containers.
setStyle	String [ID] String [Style]	N/A	Set Style parameters for the component with the given ID if it exists.
setXML	String [ID] String [new XML Content]	N/A	Dynamically generates UI components through ogscript. Replaces the contents of an element with a string of XML code.
toBottom	String [ID]	N/A	Displays the object below all others in the same container. Objects are layered. If they overlap, higher layers are drawn over lower layers.
toTop	String [ID]	N/A	Displays the object above all others in the same container. Objects are layered. If they overlap, higher layers are drawn over lower layers.
upload	File [Upload File]	N/A	Open the File Upload dialog with the specified file.

addOnClose

Runs a function when the panel is closed.

'closed' means that the tab is closed, DashBoard is closed, or the panel is reloaded.

Syntax

```
ogscript.addOnClose(Function);
```

Parameters

Parameter	Type	Required	Description
Function	Function reference	Yes	Function to be added on close.

Returns

N/A

Example 1

ogscript.addOnClose(functionName);

addRemoteTrigger

Allows remote execution of a script inside of a CustomPanel through the RossTalk GPI command. The function can be removed by calling close on the object returned.

Syntax

```
ogscript.addRemoteTrigger (trigger,function);
```

Parameters

Parameter	Type	Required	Description
Trigger	String	Yes	String that triggers the specified function to execute.
Function	String	Yes	The function to execute, including its parameters (if any).

Returns

Returns an object that contains one function named **`close`**. When executed, **`close`** removes the function.

Example

Coming soon.

appendXML

Adds a section of OGLML code to the panel identified by the **`Container ID`** parameter. The OGLML is added during runtime and does not affect the .grid file.

The appendXML function is supported within the <abs> tag only.

Syntax

```
ogscript.appendXML (container ID, XML snippet);
```

Parameters

Parameter	Type	Required	Description
container ID	String	Yes	ID of the container to append to. Valid only in <abs/> containers.
XML snippet	String (XML object)	Yes	XML code to append

Returns

N/A

Example

Coming Soon.

asyncExec

Executes a function outside of the UI current thread.

This is especially useful for operations that take time to complete. You can use **`asyncExec`** to run such operations while continuing to execute the rest of your tasks.

Syntax

```
ogscript.asyncExec (function);
```

Parameters

Parameter	Type	Required	Description
function	Function	Yes	Reference to the function to be executed. Can also be an anonymous function.

Returns

N/A

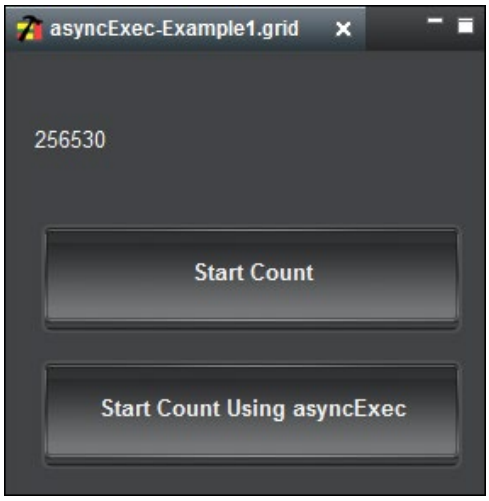
Example 1

This example displays two buttons. Each button runs a function named `incrementFunction`, which increments a parameter named `Number` until it reaches `500000`. The `Number` parameter is displayed in the top left corner of the panel.

The button labeled **Start Count** executes the function normally. No other tasks can start while the count proceeds. The display of the `Number` parameter isn't refreshed until the count is complete.

The button labeled **Start Count Using asyncExec** executes the function asynchronously. The panel can start other tasks, and the user interface continues to function normally, while the count proceeds. The display of the `Number` parameter is updated as its value changes.

The interface for this example appears as follows:



The source code for this example is as follows:

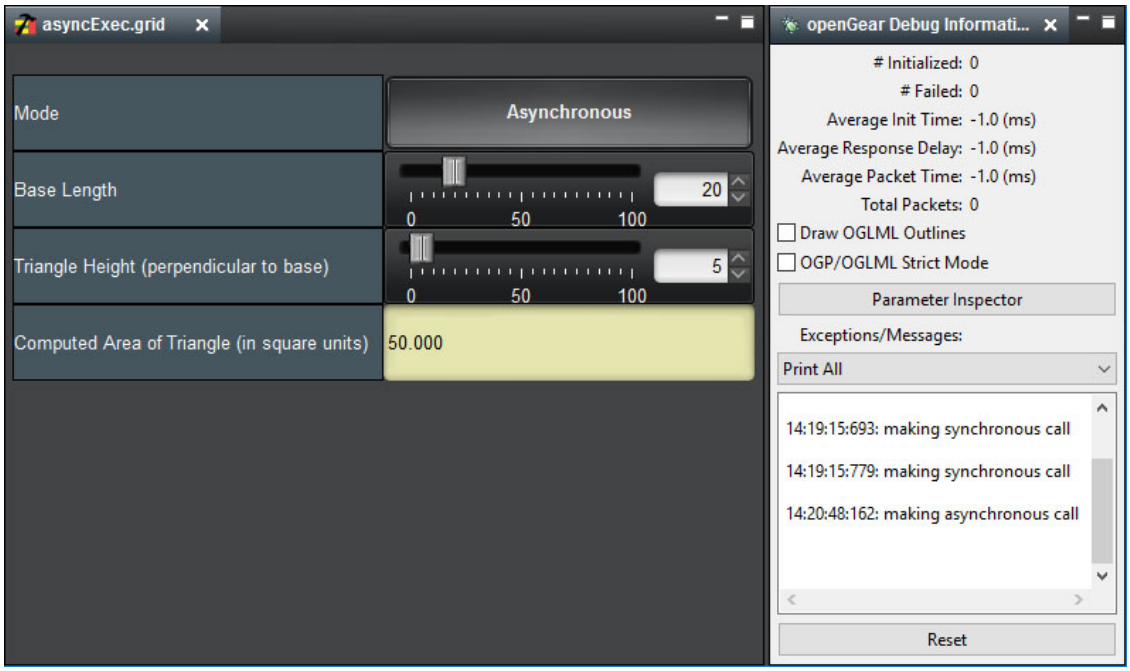
```
<abs contexttype="opengear">
  <meta>
    <params>
      <param access="1" constraint="0.0;500001.0;0.0;500001.0;1"
      constrainttype="INT_STEP_RANGE" name="Number" oid="Number"
      precision="0" type="INT32" value="0" widget="label"/>
    </params>
    <api>function reallyLongFunction()
{
<!-- &lt; represents less than and &gt; represents greater than -->
  for (var i = 0; i &lt; 500001; i++)
  {
    params.setValue('Number', 0, i);
  }
}</api>
  </meta>
  <param expand="true" height="62" left="17" oid="Number" top="20"
  width="205"/>
  <button buttontype="push" height="66" left="20" name="Start Count"
  top="100" width="250">
    <task tasktype="ogscript">reallyLongFunction();</task>
  </button>
  <button buttontype="push" height="66" left="20" name="Start Count Using
  asyncExec" top="180" width="250">
    <task tasktype="ogscript">ogscript.asyncExec(reallyLongFunction);</task>
  </button>
</abs>
```

Example 2

The `ogscript.asyncExec` function does not allow you to pass parameters directly to the function you want to call. This example demonstrates how to work around this limitation, to asynchronously execute functions that require parameters, using a “wrapped function” technique.

In this example, which calculates the area of a triangle, the user can toggle between executing the calculation function synchronously or asynchronously. Each time the calculation function is executed, the openGear debug console receives a message indicating whether the execution call was synchronous or asynchronous.

The interface for this example, including the openGear debug console, appears as follows:



The source code for this panel uses a variable named `async` to control whether the function named `callMyFunction` is executed synchronously or asynchronously.

The source code for this example is as follows:

```
<abs contexttype="opengear" style="">
  <meta>
    <ogscript handles="onchange" id="ogs-onchange-base" name="Base Change
    Handler" oid="a">calcArea();</ogscript>
    <ogscript handles="onchange" id="ogs-onchange-height" name="Height Change
    Handler" oid="b">calcArea();</ogscript>
    <api id="api-asyncExec-demo" name="asyncExec Demo">function calcArea () {
  var async = params.getValue('mode', 0) === 1;

  function callMyFunction (base, height) {
  //Note: This example uses two parameters, but you can use as few or as many
  as required.
    return function () {
      params.setValue('area', 0, (base * height/2));
    }
  }

  if (async) {
    ogscript.debug ('making asynchronous call');
    ogscript.asyncExec(callMyFunction(params.getValue('a',0),
    params.getValue('b',0)));
  } else {
    ogscript.debug ('making synchronous call');
    callMyFunction(params.getValue('a',0), params.getValue('b',0))();
    //Note: The parentheses at the end of the previous line are required to
    call the wrapped function.
  }

}</api>
    <params>
```

```
      <param access="1" constraint="0.0;100.0;0.0;100.0;1.0"
      constrainttype="FLOAT_STEP_RANGE" name="A" oid="a" precision="0"
      type="FLOAT32" value="10.0" widget="default"/>
      <param access="1" constraint="0.0;100.0;0.0;100.0;1.0"
      constrainttype="FLOAT_STEP_RANGE" name="B" oid="b" precision="0"
      type="FLOAT32" value="10.0" widget="default"/>
      <param access="1" constrainttype="FLOAT_NULL" name="Area" oid="area"
      precision="3" type="FLOAT32" value="50.0" widget="default"/>
      <param access="1" constrainttype="INT_CHOICE" name="Mode" oid="mode"
      precision="0" type="INT16" value="0" widget="default">
        <constraint key="0">Synchronous</constraint>
        <constraint key="1">Asynchronous</constraint>
      </param>
    </params>
  </meta>
  <simplegrid cols="2" height="219" left="5" top="20" width="525">
    <label header="true" name="Mode" style="txt-align:west"/>
    <param expand="true" oid="mode" showlabel="false" widget="toggle"/>
    <label header="true" name="Base Length" style="txt-align:west"/>
    <param expand="true" oid="a"/>
    <label header="true" name="Triangle Height (perpendicular to base)"
    style="txt-align:west;"/>
    <param expand="true" oid="b"/>
    <label header="true" name="Computed Area of Triangle (in square units)"
    style="txt-align:west"/>
    <param editable="false" expand="true" oid="area" widget="text-display"/>
  </simplegrid>
</abs>
```

asyncFTP

Sends a file to an FTP server. If a callback is provided, **`asyncFTP`** calls it when the operation is complete.

Note: As the file is transferred, a progress attribute is updated. You can add an ogscipt handler to monitor changes to the attribute to show progress.

Syntax

```
ogscript.asyncFTP (host, port, username, password, destPath, destName,
binary, sourceFilePath, callback);
```

Parameters

Parameter	Type	Required	Description
host	String	Yes	The host name of the destination computer.
port	Integer	Yes	The port number to which the data is to be sent.
username	String	Yes	The username required to log onto the destination computer.
password	String	Yes	The password required to log onto the destination computer.
destPath	String	No	The directory path where the data is to be saved on the destination computer.
destName	String	No	The name of the destination file. Can be used to rename the existing file. If a file with the same name exists in the destination path, that file is overwritten.
binary	Boolean	Yes	Specifies the transfer mode. When true, binary transfer is used. When false, ASCII transfer is used.
sourceFilePath	String	Yes	The directory path to the source file. The path can be absolute or relative.
callback	function reference	No	The callback is called when the operation is complete, whether or not the operation is successful.

Returns

N/A

Example 1

The following example is a task. It uses variable to populate the parameters of the **asyncFTP** function. It also includes a callback to indicate success or failure of the transfer.

```
<task tasktype="ogscript">function callback(success, sourceFilePath,
exception)
    {
        if (success)
        {
          ogscript.rename('label.bytes', 'SUCCESS!');
        }
        else
        {
          ogscript.rename('label.bytes', 'FAIL!');
        }
    }
    ogscript.rename('label.bytes', 'TRYING TO SEND FILE');  var host =
    params.getStrValue('params.host', 0);
    var port = params.getValue('params.port', 0);
    var user = params.getStrValue('params.username', 0);
    var password = params.getStrValue('params.password', 0);  var file =
    params.getStrValue('params.file', 0);
    var destPath = params.getStrValue('params.destpath', 0);  var
    destFileNameOverride = null;
    var isBinary = true;
    ogscript.asyncFTP(host, port, user, password, destPath,
    destFileNameOverride, isBinary, file, callback);
    ogscript.rename('label.bytes', 'Waiting...');
```

```
</task>
```

Example 2

The following is an example of an ogscript handler for monitoring and reporting the progress of the transfer.

```
<ogscript attribute="com.rossvideo.ftp.event"  handles="attributechange">
    var progressEvent = event.getNewValue();
    if (progressEvent == null)
    {
      ogscript.debug('No progress');
    }
    else
    {
      ogscript.rename('label.bytes',  (progressEvent.getTotalBytesTransferred()
      / 1024) + 'kb');
    }
</ogscript>
```

asyncFTPGet

Retrieves a file from FTP server.

Syntax

```
ogscript.asyncFTPGet (host, port, username, password, srcPath, srcName,
binary, destFilePath or null, callback);
```

Parameters

Parameter	Type	Required	Description
host	String	Yes	The host name of the source computer, from which the file is to be retrieved
port	Integer	Yes	The port number required to access the source computer.
username	String	Yes	The username required to log onto the source computer.
password	String	Yes	The password required to log onto the source computer.
srcPath	String	No	The directory path where the source file is located.
srcName	String	Yes	The name of the file to be retrieved.
binary	Boolean	Yes	Specifies the transfer mode. When true, binary transfer is used. When false, ASCII transfer is used.
destFilePath or null	String	No	The directory path where the file is to be saved on the local computer. If null, the file is saved in the same directory as the panel.
callback	function reference	No	The callback is called when the operation is complete, whether or not the operation is successful.

Returns

N/A

Example

Coming soon.

asyncPost

Send an asynchronous post to the given URL. Call the given function when the post has completed. The data retrieved from the HTTP Post is passed as a string as the first variable in the method.

If the MIME type of the HTTP response is image or binary, the result will be a byte array containing what is fetched.

Syntax

```
ogscript.asyncPost (URL, HTTP Post Data, Callback Function, Include
Response);
```

Parameters

Parameter	Type	Required	Description
URL	String	Yes	URL to send a post.
HTTP Post Data	String	Yes	Post to send to the specified URL.
Callback Function	Function	Yes	Function to call after the post completes.

Include Response	Boolean	No	If true, result is a JSON Object
			`{`
			`responseCode = HTTP RESONSE CODE,`
			`contentType = HTTP MIME TYPE`
			`url = URL Requested`
			`bytes= BYTES RECEIVED`
			`}`
			Otherwise, it is content fetched over HTTP parsed as though it's a string (as before).

Returns

N/A

Example

Coming soon.

cancelTimer

Cancel, stop and clean up, a timer with the given ID.

Note: For information about creating a timer function, see installTimer on page 242.

Syntax

```
ogscript.cancelTimer (Timer ID);
```

Parameters

Parameter	Type	Required	Description
Timer ID	String	Yes	ID of the timer to stop and clean up.

Returns

N/A

Example

```
//Stop the timer that was created with installTimer
ogscript.cancelTimer('myTimer');
```

copyByteArray

Creates a full or partial copy of a byte array.

Syntax

```
ogscript.copyByteArray(src, offset, length)
```

Parameters

Parameter	Type	Required	Description
src	byte array	Yes	The byte array to be copied.
offset	Integer	Yes	Index of the first byte to be copied. Use 0 for the start of the array.
length	Integer	Yes	The number of bytes to copy. Tip: To copy the entire array, use src.length.

Returns

byte array

Example 1

In the following example, the contents of a byte array named **`srcArray`** are copied into a variable named **myCopy**.

```
var myCopy=ogscript.copyByteArray (srcArray,0,srcArray.length);
```

Example 2

In the following example, the 20 bytes of a byte array named **`srcArray`**, starting at byte **`4`**, are copied into a variable named **myCopy**.

```
var myCopy=ogscript.copyByteArray (srcArray,4,20);
```

createByteArray

Creates an empty byte array of a specified size.

Syntax

```
ogscript.createByteArray (length);
```

Parameters

Parameter	Type	Required	Description
length	Integer	Yes	The size of the new array, in bytes.

Returns

An empty byte array.

Example

```
var myNewByteArray = ogscript.createByteArray (12);
```

createFileInput

Accesses a file as a byte array with the same capabilities as **`MessageParser`**, to read raw bytes. See also createMessageParser on page 228.

Syntax

```
ogscript.createFileInput (File path);
```

Parameters

Parameter	Type	Required	Description
File path	String	Yes	Path of the file to open (can be relative to the panel)

Returns

```
FileInputParser (like MessageParser but with getSize(), close(), and
isClosed().
```

Example

Coming Soon.

createFileOutput

Creates a new file or appends to an existing file. Instead of saving XML or string data, gives access to write raw bytes (or strings, or shorts, or ints, etc.). Also gives the ability to append to a file. Once open, it does not close the file until the panel is closed or **`close()`** is called. This is handy for logging.

Similar to MessageBuilder (see createMessageBuilder on page 228).

Syntax

```
ogscript.createFileOutput (File path, appendToExistingfile);
```

Parameters

Parameter	Type	Required	Description
File path	String	Yes	File path of the file to be created or appended.
appendToExistingfile	Boolean	Yes	When true, data is appended to existing file. When false, a new file is created.

Returns

`FileOutputBuilder`, which is same as **`MessageBuilder`** with added functions for **`clear()`** (overwrite file), **`close()`**, **`getSize()`**, **`flush()`**, and **`isClosed()`**.

Example

Coming soon.

createMessageBuilder

Creates a message builder, which enables you to construct a message. The message is created as a byte array, can contain multiple data types.

Syntax

```
ogscript.createMessageBuilder ();
```

Parameters

N/A

Returns

Returns a MessageBuilder object used to build byte arrays (generally for creating network messages).

Example

In the following example, a variable named myMessage is created to contain message content created by a message builder. Then data of various data types are added to the message. The variable messageArray is defined to contain the message content as a byte array.

Tip: You can use the createMessageParser function to parse messages.

```
var myMessage = ogscript.createMessageBuilder();
myMessage.writeBoolean(true);  myMessage.writeByte(255);
myMessage.writeByte(255);
myMessage.writeShort(65535);  myMessage.writeShort(65535);
myMessage.writeChar('a');  myMessage.writeInt(65536);
myMessage.writeLong(4294967296);  myMessage.writeFloat(0.000001);
myMessage.writeDouble(0.000002);  myMessage.writeString('abcd');
myMessage.writeUTF('Hello World'); //includes 2-byte length count
var messageArray = myMessage.toByteArray();
```

createMessageParser

Creates a message parser, which enables you to parse a message.

Syntax

```
ogscript.createMessageParser (messageBytes);
```

Parameters

Parameter	Type	Required	Description
messageBytes	byte array	Yes	The source byte array.

Returns

Returns a MessageParser object (generally used to parse the various pieces of messages received over the network).

Example

In the following example, a variable named messageArray contains several pieces of data of various data types to be extracted by a message parser. A variable named parsedMessage is created to contain the extracted message content. Each element of the array is parsed and sent to the debug utility.

Tip: You can use the createMessageBuilder function to create messages.

```
var parsedMessage = ogscript.createMessageParser(messageArray);
ogscript.debug(parsedMessage.readBoolean());
ogscript.debug(parsedMessage.readByte());
ogscript.debug(parsedMessage.readUnsignedByte());
```

```
ogscript.debug(parsedMessage.readShort());
ogscript.debug(parsedMessage.readUnsignedShort());
ogscript.debug(parsedMessage.readChar());
ogscript.debug(parsedMessage.readInt());
ogscript.debug(parsedMessage.readLong());
ogscript.debug(parsedMessage.readFloat());
ogscript.debug(parsedMessage.readDouble());
ogscript.debug(parsedMessage.readString(4));
ogscript.debug(parsedMessage.readUTF());</task>
```

debug

Write a string to the openGear Debug Information view.

The openGear Debug Information view must be open to view debug messages. To open the openGear Debug Information view, select openGear Debug Information from the Views menu in DashBoard.

Syntax

```
ogscript.debug (Message);
```

Parameters

Parameter	Type	Required	Description
Message	String	Yes	Message to display in the openGear Debug Information View.

Returns

N/A

Example 1

```
ogscript.debug ('This is a message');
```

Example 2

```
var data = params.getValue(0x12,0);
ogscript.debug ('Parameter 0x12 (score): ' + data);
```

Example 3

```
ogscript.debug ('Parameter 0x12 (score): ' + params.getValue(0x12,0));
```

fireGPI

Sends a Trigger GPI message to panels. When buttons, labels, and displayed parameters that have a matching GPI Trigger receive the message, their task lists are executed.

Tip: This function can be used for inter-panel communication, by triggering globally.

Syntax

```
ogscript.fireGPI (Trigger), (State), (Global);
```

Parameters

Parameter	Type	Required	Description
Trigger	String	Yes	GPI Trigger message.
State	String	No	Sends optional data string, which can be read by the script.

Parameter	Type	Required	Description
Global	Boolean	Yes	When true, applies to all open panels. When false, applies only the panel initiating the trigger.

Returns

N/A

Example

In this example, the GPI trigger message 'StartClock' and the state data 'ResetClock' are sent to all open panels.

```
ogscript.fireGPI ('StartClock','ResetClock',true);
```

getAllById

Get all Objects accessible in the current context that have the associated ID.

Syntax

```
ogscript.getAllById (Object ID);
```

Parameters

Parameter	Type	Required	Description
Object ID	String	Yes	ID of the objects in the current context to get.

Returns

Object []

Example

Coming soon.

getAttribute

Get an attribute registered in the context with the given ID.

Syntax

```
ogscript.getAttribute (Attribute ID);
```

Parameters

Parameter	Type	Required	Description
Attribute ID	String	Yes	ID from which to get a registered in context attribute.

Returns

Object

Example

Coming soon.

getBuild

Returns the DashBoard version number. This is the same version number you see in DashBoard if you click **About DashBoard** on the **Help** menu.

Syntax

```
ogscript.getBuild ();
```

Parameters

N/A

Returns

DashBoard version number, similar to the following:

```
Version 7.0.0I 2015-06-12 T09:54
```

getComponentsById

Get all Java Swing components accessible in the current context that have the associated ID.

Syntax

```
ogscript.getComponentsById (Object ID);
```

Parameters

Parameter	Type	Required	Description
Object ID	String	Yes	ID from which to get all Java Swing components accessible in the current context.

Returns

Component []

Example

Coming soon.

getCurrentUser

Returns the username of the current DashBoard user.

When a User Rights Management server is present, this function returns the username of the user signed-in to DashBoard.

When no User Rights Management Server is found, this function returns the computer account name.

Syntax

```
ogscript.getCurrentUser ( );
```

Parameters

N/A

Returns

String

Example

This example uses the getCurrentUser function to read the user name, and then uses the rename function to rename a label. For more information about the rename function, see rename on page 247.

The label is defined in the .grid file as follows:

```
<label height="49" id="Welcome Label" left="136" name="Welcome"  style="txt-
align:west;" top="275" width="188"/>
```

The script to read the user name and then rename the label is as follows:

```
//read the login user name
var loginName = ogscript.getCurrentUser();


//display the user name in the Welcome label  var message = 'Welcome ' +
loginName;  ogscript.rename('Welcome Label',message);
```

getIncludeById

Returns the first include with the given ID. The include must have been created using the <include> tag.

Syntax

```
ogscript.getIncludeById (Include ID);
```

Parameters

Parameter	Type	Required	Description
Include ID	String	Yes	ID of the include to find.

Returns

IncludeReloadableContainer

Example

Coming soon.

getListenerById

Starts or stops a listener. Can also check whether a listener is started.

Syntax

```
ogscript.getListenerById (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the listener.

Returns

getListenerById returns an object representing the listener.

This object has three public methods you can call: start(), stop(), and isStarted().

The return depends on which of the three methods is used:

- If the start() method is used, return is true if the listener started successfully; otherwise false.
- If the stop() method is used, return is true if the listener stopped successfully; otherwise false.
- If the isStarted() method is used, return is true if the listener is started; otherwise false.

Example

```
var myListener = ogscript.getListenerById (myId); myListener.start ();
myListener.stop ();

myListener.isStarted ();
```

getModificationDate

Retrieves the time the specified file was last modified.

Syntax

```
ogscript.getModificationDate (file path);
```

Parameters

Parameter	Type	Required	Description
File path	String	Yes	Path to the file.

Returns

Returns the time the specified file was last modified, in Unix Epoch time (also known as POSIX time), as a LONG value.

Example

Coming soon.

getObject

You can create an object and reference it in other parts of the code. Some possible uses include:

- Storing parsed XML data in an object so you don't have to re-parse it.
- Storing the results of an async HTTP post so you don't have to re-fetch it.
- Storing connection code so you cna reference it wherever your code needs to establish that connection.

The getObject function works in conjunction with the putObject function. The putObject function defines the object. The getObject function references the object. The scope of a defined object is global, so you can reference it from anywhere in your panel code.

For information about the putObject function, see putObject on page 244.

Syntax

```
ogscript.getObject(Key);
```

Parameters

Parameter	Type	Required	Description
Key	String	Yes	The name used to reference what is being stored.

Returns

String.

Example

The following example parses and stores data from an XML file in a variable so it can be used globally without the need to re-parse the XML data each time you want to use it.

It defines a function named loadTheXML, which uses the parseXML function to retrieve XML data from a file and load it into a variable named myObject. It then uses the putObject function to copy the data into a variable named myXML. The readTheXML function loads the data into a variable named otherObject.

```
function loadTheXML()
{
var myObject = ogscript.parseXML('file:/c:/mydocument.xml');
ogscript.putObject('myXML',myObject);
}


function readTheXML()
{
var otherObject = ogscript.getObject('myXML');
// Do anything you want with the data, now contained in the  otherObject
variable.
}
```

getPosition

Retrieves the horizontal (x) and vertical (y) position of a panel object, in pixels.

Syntax

```
ogscript.getPosition (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	The ID of the panel object.

Returns

JAVA point object containing public variables x and y, populated with values for the horizontal (y) and vertical (y) position of the object, in pixels.

Example

The following example draws a label that can be resized and repositioned. When the user drags the middle of the label, it moves. When the user drags the bottom right corner of the label, the label is resized.

```
<abs bottom="0" contexttype="opengear" left="0" right="0" top="0">
    <meta>
        <ogscript handles="onmousedown" targetid="move-label">var size =
        ogscript.getSize('move-label');
        if (event.getX() &lt; size.width - 10 &amp;&amp; event.y &lt;
        size.height - 10)
            {
            ogscript.putObject('mode', 'move');  ogscript.putObject('position',
            ogscript.getPosition('move-label'));  ogscript.putObject('offsetX',
            event.x);  ogscript.putObject('offsetY', event.y);
            }
        else
            {
            ogscript.putObject('mode', 'size');
```

```
            }
        </ogscript>
        <ogscript handles="ondrag" targetid="move-label">
        if (ogscript.getObject('mode') == 'size')
            {
            ogscript.setSize('move-label', event.getX(), event.getY());
            }
        else if (ogscript.getObject('mode') == 'move')
            {
            var origin = ogscript.getObject('position');  var offsetX =
            ogscript.getObject('offsetX');  var offsetY =
            ogscript.getObject('offsetY');
            ogscript.reposition('move-label', origin.x + event.x -  offsetX,
            origin.y + event.y - offsetY);
            ogscript.putObject('position',  ogscript.getPosition('move-label'));
            }
        </ogscript>
    </meta>
    <label height="116" id="move-label" left="27"  style="bdr:etched;bg#FF0000"
    top="38" width="215"/>
</abs>
```

getPrivateString

Get a string defined in a private lookup table that matches the specified lookup ID.

Note: Use the getPrivateString function if the lookup table has an ID. If the lookup table has no ID, use the getString function. For more information about the getString function, see getString on page 237.

Syntax

```
ogscript.getPrivateString (Lookup ID, Key);
```

Parameters

Parameter	Type	Required	Description
Lookup ID	String	Yes	ID of the string to find in the specified lookup table.
Key	String	Yes	Private lookup table in which to find the specified string.

Returns

String

Example

This example uses the getPrivateString function to read an IP address stored in a lookup table. The lookup table is defined at the beginning of the .grid file, and can be accessed by any script.

The lookup table definition for this example is as follows:

```
<lookup id="hosts">
    <entry key="XPression.host">10.0.2.210</entry>
    <entry key="XPression.port">7788</entry>
</lookup>
```

The script to read an entry from the lookup table is as follows:

```
//Get the IP Address associated with entry key XPression.host
var host = ogscript.getPrivateString('hosts',' XPression.host ');
```

getScopedAttribute

Get an attribute in the named scope that has the given ID. Scopes are often internally defined by DashBoard.

Syntax

```
ogscript.getScopedAttribute (Scope Name, Attribute ID);
```

Parameters

Parameter	Type	Required	Description
Scope Name	String	Yes	Name of the scope in which to get and attribute.
Attribute ID	String	Yes	ID of the attribute to get in the named scope.

Returns

Object

Example

Coming soon.

getSize

Retrieves the width and height of the specified panel object, in pixels.

Syntax

```
ogscript.getSize (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the panel object.

Returns

Dimension object with d.width and d.height available.

Example

The following example draws a label that can be resized and repositioned. When the user drags the middle of the label, it moves. When the user drags the bottom right corner of the label, the label is resized.

```
<abs bottom="0" contexttype="opengear" left="0" right="0" top="0">
    <meta>
        <ogscript handles="onmousedown" targetid="move-label">var size =
        ogscript.getSize('move-label');
        if (event.getX() &lt; size.width - 10 &amp;amp;&amp;amp; event.y &lt;
        size.height - 10)
            {
            ogscript.putObject('mode', 'move');  ogscript.putObject('position',
            ogscript.getPosition('move-label'));  ogscript.putObject('offsetX',
            event.x);  ogscript.putObject('offsetY', event.y);
            }
        else
            {
            ogscript.putObject('mode', 'size');
            }
        </ogscript>
        <ogscript handles="ondrag" targetid="move-label">
        if (ogscript.getObject('mode') == 'size')
            {
            ogscript.setSize('move-label', event.getX(), event.getY());
            }
        else if (ogscript.getObject('mode') == 'move')
            {
            var origin = ogscript.getObject('position');  var offsetX =
            ogscript.getObject('offsetX');  var offsetY =
            ogscript.getObject('offsetY');
            ogscript.reposition('move-label', origin.x + event.x -  offsetX,
            origin.y + event.y - offsetY);
            ogscript.putObject('position',  ogscript.getPosition('move-label'));
            }
        </ogscript>
    </meta>
    <label height="116" id="move-label" left="27" style="bdr:etched;bg#FF0000"
    top="38" width="215"/>
</abs>
```

getString

Get a string defined in the global lookup table.

Note: Use the getString function if the lookup table has no ID. If the lookup table has an ID, use the getPrivateString function. For more information about the getPrivateString function, see getPrivateString on page 235.

Syntax

```
ogscript.getString (Key);
```

Parameters

Parameter	Type	Required	Description
Key	String	Yes	Private lookup table from which to get string.

Returns

Object

Example

This example uses the getString function to read an IP address stored in a lookup table. The lookup table definition for this example is as follows:

```
<lookup>
    <entry key="Tom">television</entry>
</lookup>
```

The script to read an entry from the lookup table is as follows:

```
//Get the string associated with entry key Tom ogscript.getString('Tom');
```

getTimerManager

Get the timer manager for the context to access timers and perform operations on selected timers.

Syntax

```
ogscript.getTimerManager ( );
```

Parameters

N/A

Methods

The getTimerManager function is an object that has several methods. The following methods can be run on an existing timer. A timer can be created using the installTimer function or using the graphical editor. For more information about the installTimer function, see installTimer on page 242.

Method	Parameter Required	Description
isRunning()	N/A	Checks whether the time is running.
startTimer(Boolean reset)	Yes true or false	Starts the timer. If the boolean parameter is set to true, the timer resets to the starting time when the function is performed. If the boolean parameter is set to false, the function is performed at the timer's current time.
stopTimer(Boolean reset)	Yes true or false	Stops the timer. If the boolean parameter is set to true, the timer resets to the starting time when the function is performed. If the boolean parameter is set to false, the function is performed at the timer's current time.
resetTimer()	N/A	Resets the timer to the start time.
setStart(Long valueInMilliseconds)	Yes Milliseconds (Long)	Sets the start time of the timer.

Method	Parameter Required	Description
setStop(Long valueInMilliseconds)	Yes Milliseconds (Long)	Sets the stop time of the timer.
setTime(Long valueInMilliseconds)	Yes Milliseconds (Long)	Sets the current time of the timer.
getStart()	N/A	Returns the timer's start time in milliseconds (Long).
getStop()	N/A	Returns the timer's stop time in milliseconds (Long).
getCurrent()	N/A	Returns the timer's current value in milliseconds (Long).
incrementTime(Long difference)	Yes Milliseconds (Long)	Increments the timer value by the specified number of milliseconds
setPattern(String dateTimePattern)	Yes Time format definition	Sets the time format pattern for displaying time values.

Returns

ContextTimerManager

Example 1 — getTimerManager function using isRunning method

```
//verify if timer named 'selftimer' is currently running
if (ogscript.getTimerManager().getTimer('selftimer').isRunning())
    {
    ogscript.debug('running = true');
    }
else
    {
     ogscript.debug('running = false');
    }
```

Example 2 — getTimerManager function using startTimer method

```
//Starts a timer named 'selftimer'
ogscript.getTimerManager().getTimer('selftimer').startTimer(false);
```

Example 3 — getTimerManager function using stopTimer method

```
//Stops a timer named 'selftimer'
ogscript.getTimerManager().getTimer('selftimer').stopTimer(false);
```

Example 4 — getTimerManager function using resetTimer method

```
//Resets a timer named 'selftimer' to the start time
ogscript.getTimerManager().getTimer('selftimer').resetTimer();
```

Example 5 — getTimerManager function using setStart method

```
//Set the start time of a timer named'selftimer' to 30 seconds (30000ms)
ogscript.getTimerManager().getTimer('selftimer').setStart(30000);
```

Example 6 — getTimerManager function using setStop method

```
//Set the stop time of a timer named 'selftimer' to two minutes (120000 ms)
ogscript.getTimerManager().getTimer('selftimer').setStop(120000);
```

Example 7 — getTimerManager function using setTime method

```
//Set the current time of a timer named 'selftimer' to 59 seconds (59000 ms)
ogscript.getTimerManager().getTimer('selftimer').setTime(59000);
```

Example 8 — getTimerManager function using getStart method

```
// Get the start time of a timer named  'selftimer' var startTime =
ogscript.getTimerManager().getTimer('selftimer').getStart();
```

Example 9 — getTimerManager function using getStop method

```
// Get the stop time of a timer named'selftimer' var stopTime =
ogscript.getTimerManager().getTimer('selftimer').getStop();
```

Example 10 — getTimerManager function using getCurrent method

```
// Get the current time of a timer named'selftimer' var currentTime =
ogscript.getTimerManager().getTimer('selftimer').getCurrent();
```

Example 11 — getTimerManager function using incrementTime method

```
//increase the current time of a timer named   'selftimer' by 30 seconds
ogscript.getTimerManager().getTimer('selftimer').incrementTime(30000)
;


//decrease the current time of a timer named 'selftimer' by 5 seconds
ogscript.getTimerManager().getTimer('selftimer').incrementTime(-5000)
;
```

Example 12 — getTimerManager function using setPattern method

The following table describes the syntax for setting the time format. For some formats, repeating the letter returns more digits or a variation of the format. For example, when specifying M for month, one M shows the month number with no leading zero, two Ms adds a leading zero for months 0 to 9, three Ms shows the three letter month (such as Jan), and four or more Ms shows the full month name (such as January).

Letter	Date or Time Component	Presentation	Examples
D	Day	Number	189
H	Hour of the day (0-23)	Number	8
m	Minute of the hour	Number	30
s	Second of the minute	Number	55
S	Millisecond	Number	768
G	Era designator	Text	AD
Y	Year	Number	1969; 69
M	Month of the year	Text or number	September; Sep; 09
w	Week of the year	Number	27
W	Week of the month	Number	3
d	Day of the month	Number	12
F	Day of the week in the month	Number	1 If the day of the week is Tuesday, 1 would denote the first Tuesday of the month
E	Day of the week	Text	Friday; Fri

Letter	Date or Time Component	Presentation	Examples
k	Hour of the day (1-24)	Number	22
K	Hour in AM/PM (0-11)	Number	0
h	Hour in AM/PM (1-12	Number	10
a	AM/PM marker	Text	PM
z	Time zone	General Time Zone	Pacific Standard Time, PST,
Z	Time zone	RFC 822 time zone	-0800

The following code example returns the date and time. An example of the date and time as returned by this example is Sep 30, 2013 2:35:34 PM.

```
//Sets the display format of a timer named 'simpleclock' to show full  date and time
ogscript.getTimerManager().getTimer('simpleclock').setPattern('MMM  dd, yyyy h:mm:ss a');
```

hide

Hide the popup associated with the specified ID.

Note: to use the hide function, a popup must already exist. Popups can be created only in the JavaScript source, not in DashBoard.

Syntax

```
ogscript.hide (Popup ID);
```

Parameters

Parameter	Type	Required	Description
Popup ID	String	Yes	ID of the popup to hide.

Returns

N/A

Example

This example includes two sections of XML code to be added to the .grid file. The first creates a button that opens a popup. The second creates a button that hides the popup.

```
//This example creates a button which, when clicked by a user, opens  the popup area.
<popup id="popup1" left="20" name="Click here to open the Popup"  top="25">
   <abs height="300" left="200" style="bdr:etched;" top="200"  width="300">
   </abs>
</popup>

//This example creates a button which, when clicked by a user, hides the popup.
<button buttontype="push" height="50" left="50" name="Click here to hide the Popup" top="500" width="200">
   <task tasktype="ogscript">ogscript.hide('popup1');</task>
</button>
```

installTimer

Create a timer with the given ID and register it in the ContextTimerManager. Start the timer after the specified delay. If requested, repeat the timer at the specified frequency. When the timer fires, run the specified ogScript function.

Syntax

```
ogscript.installTimer (Timer ID, Repeat, Delay, Repeat Delay, Task);
```

Parameters

Parameter	Type	Required	Description
Timer ID	String	Yes	ID of the timer to create and register in the ContextTimerManager.
Repeat	Boolean	Yes	true — repeat the timer using the specified Delay and Repeat Delay. false — only run the timer once, do not repeat the timer.
Delay	Long	Yes	Number of milliseconds to wait before starting the timer.
Repeat Delay	Long	Yes	How frequently the associated function runs, in milliseconds.
Task	Function	Yes	ogScript function to run when the timer fires.

Returns

N/A

Example

This example creates a label named "Time" and a button named "Install Timer". When a user clicks the "Install Timer" button, an associated task runs a function named myFunction (), which creates a timer.

It also retrieves the time value every 30 seconds, and loads it into a variable named str which is displayed on the "Time" label. The myFunction () function uses the installTimer function to create the timer and set the rate at which the time data is updated.

```
<label height="80" id="timeLabel" left="43" name="Time"  style="txt-
align:west" top="26" width="275"/>
<button buttontype="push" height="57" left="48" name="Install Timer"
top="133" width="184">
   <task tasktype="ogscript">function myFunction()
      {
         var date = new Date();
         var str = date.getHours() + ':' + date.getMinutes() + ':' +
         date.getSeconds();
         ogscript.rename('timeLabel', 'Time: ' + str);
      }
   //create a timer that starts immediately and runs myFunction  every 30
   seconds (30000 milliseconds)
   ogscript.installTimer('myTimer', true, 0, 30000, myFunction);
   </task>
</button>
```

isTimerRunning

Report whether or not a timer exists and is in the “running” state.

Syntax

```
ogscript.isTimerRunning (Timer ID);
```

Parameters

Parameter	Type	Required	Description
Timer ID	String	Yes	true — a timer with the given ID exits and is in the “running” state. false — a timer with the give ID does not exist or is not in the “running” state.

Returns

Boolean

Example

```
//verify if the timer is currently running
var runtime = ogscript.isTimerRunning('selftimer');
```

parseXML

Parse and return an XML document using the org.w3c.dom.Document API. The XML document to parse can be provided in the following ways:

- Piece of well-formatted XML
- URL relative to a CustomPanel
- File URL (file:/c:/…)
- http URL (http://…)

The document is loaded via a blocking call that is run in the DashBoard User Interface thread.

Calls to load documents over a network (for example, using http://) are strongly discouraged and can have undesired impacts on the UI performance.

Syntax

```
ogscript.parseXML (Document);
```

Parameters

Parameter	Type	Required	Description
Document	String	Yes	XML document to parse.

Returns

XML Document

For more information about returns, refer to the following URL:

http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/Document.html

Example

The following example loads an XML file from the web using an asynchronous http request. An XPath expression extracts data from the XML and displays it on a label.

```
function myFunc(pageContent)
    {
    var xmlPageContent = '&lt;?xml version="1.0"  encoding="UTF-8"?&gt;\n' +
    pageContent;
    var document = ogscript.parseXML(xmlPageContent);  var nodeList =
    ogscript.runXPath('/response/sports/sportsItem/leagues/leaguesIt
    em/teams/teamsItem/name', document);  var teamList = '&lt;html&gt;';
    ogscript.debug(nodeList.getLength());
    for (var i = 0; i &lt; nodeList.getLength(); i++)
        {
        teamList = teamList + nodeList.item(i).getTextContent() +
        '&lt;br/&gt;';
        }
    ogscript.rename('resultLabel', teamList + '&lt;/html&gt;');
    }


ogscript.asyncPost('http://api.oursports.com/v1/sports/hockey/league/
teams/?_accept=text%6Axml&amp;apikey=ksjdur7euejru47fkbos85kg',null,
myFunc);
```

putObject

You can create an object and reference it in other parts of the code. Some possible uses include:

- Storing parsed XML data in an object so you don't have to re-parse it.
- Storing the results of an async HTTP post so you don't have to re-fetch it.
- Storing connection code so you cna reference it wherever your code needs to establish that connection.

The putObject function works in conjunction with the getObject function. The putObject function defines the object. The getObject function references the object. The scope of a defined object is global, so you can reference it from anywhere in your panel code.

For information about the getObject function, see getObject on page 233.

Syntax

```
ogscript.putObject(Key, Value);
```

Parameters

Parameter	Type	Required	Description
Key	String	Yes	The name of the object in which the data is being stored.
Value	String	Yes	The value to be stored.

Returns

N/A.

Example

The following example parses and stores data from an XML file in a variable so it can be used globally without the need to re-parse the XML data each time you want to use it.

It defines a function named loadTheXML, which uses the parseXML function to retrieve XML data from a file and load it into a variable named myObject. It then uses the putObject function to copy the data into a variable named myXML. The readTheXML function loads the data into a variable named otherObject.

```
function loadTheXML()
    {
        var myObject = ogscript.parseXML('file:/c:/mydocument.xml');
        ogscript.putObject('myXML',myObject);
    }
function readTheXML()
    {
        var otherObject = ogscript.getObject('myXML');
        // Do anything you want with the data, now contained in the otherObject
        variable.
    }
```

putPrivateString

Add or replace a string in a private lookup table.

Note: Use the putPrivateString function if the lookup table has an ID. If the lookup table has no ID, use the putString function. For more information about the putString function, see putString on page 246.

Syntax

```
ogscript.putPrivateString (Lookup ID, Key, Value);
```

Parameters

Parameter	Type	Required	Description
Lookup ID	String	Yes	ID of the string to create or replace in the specified lookup table.
Key	String	Yes	Private lookup table in which to create or replace the specified string.
Value	String	Yes	New value for the specified string.

Returns

N/A

Example

This example uses the putPrivateString function to replace a datum in a lookup table. The lookup table definition for this example is as follows:

```
<lookup id="hosts">
    <entry key="XPression.host">10.0.2.210</entry>
    <entry key="XPression.port">9999</entry>
</lookup>
```

The script to replace an entry in the lookup table is as follows:

```
//Replace the port number associated with entry key XPression.host
ogscript.putPrivateString('hosts',' XPression.port ', '7788');
```

putString

Add or replace a string in the global lookup table.

Note: Use the putPrivateString function if the lookup table has no ID. If the lookup table has an ID, use the putPrivateString function. For more information about the putPrivateString function, see putPrivateString on page 245.

Syntax

```
ogscript.putString (Lookup ID, Value);
```

Parameters

Parameter	Type	Required	Description
Lookup ID	String	Yes	ID of the string to create or replace in the global lookup table.
Value	String	Yes	New value for the specified string.

Returns

N/A

Example

This example uses the putString function to replace a datum in a lookup table. The lookup table definition for this example is as follows:

```
<lookup>
   <entry key="Tom">television</entry>
</lookup>
```

The script to replace an entry in the lookup table is as follows:

```
//Replace the string associated with entry key Tom
ogscript.putString('Tom','telephone');
```

reload

Rebuild the user interface element with the specified ID. If the ID is for an <include> tag, re-fetch the included document before rebuilding the user interface.

If no ID is provided, rebuilds the entire document.

Syntax

```
ogscript.reload (User Interface ID);
```

Parameters

Parameter	Type	Required	Description
User Interface ID	String	Yes	ID of the user interface element to rebuild.

Returns

Null, if null is provided as the ID.

Example

In this example, the ogscript.reload function is used to rebuild a drop-down list to show new options.

```
//create a new array of colours
var color = new Array("Red","Green","Blue");

//populate the dropdown color_list with the color array
params.replaceIdentifiedConstraint('color_list',
params.createIntChoiceConstraint(color));

//reload the dropdown list to view the new options
ogscript.reload('color_list');
```

rename

Modify the text associated with a tab name, label, or button. Use the Component ID to specify the component to rename. Do not use the Object ID (OID).

To view the ID of a component, double-click the component in PanelBuilder to open the Edit Component dialog box. The ID box displays the ID of the selected component.

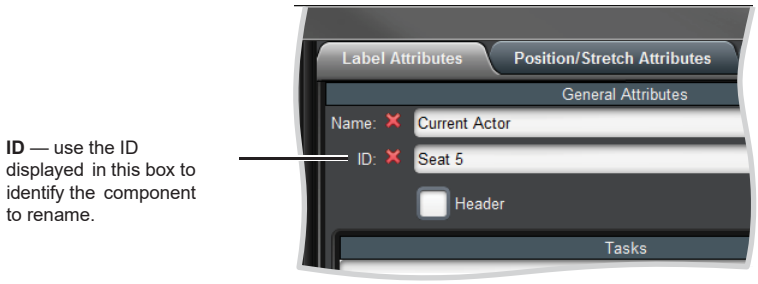


Figure 2.1 *Component ID in the Edit Component dialog box*

Syntax

```
ogscript.rename (Component ID, Name);
```

Parameters

Parameter	Type	Required	Description
Component ID	String	Yes	ID of the user interface component to rename.
Name	String	Yes	New text to display on the screen for the specified user interface component.

Returns

N/A

Example 1

```
// Set the item with ID='Seat 5' to have the text 'Mika Andersen'
ogscript.rename ('Seat 5','Mika Andersen');
```

Example 2

```
// Read the value of a parameter into a variable named data  var data =
params.getValue(0x12,0);

// Use the variable named data to make a new ID and set the ID to have the text
'Mika Andersen'
ogscript.rename('Seat ' + data,'Mika Andersen');
```

reposition

Moves a component to an absolute position, defined as an X - Y pixel position.

Alternatively, you can specify a component's position by percentage of the container's width and height. For more information, see repositionByPercent on page 248.

Syntax

```
ogscript.reposition (ID, x position, y position);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the component you want to reposition
x position	Integer	Yes	Number of pixels from the left
y position	Integer	Yes	Number of pixels from the right

Returns

N/A

Example

In this example, the task associated with the "Top Left" button uses the ogscript.reposition function to reposition a label.

```
<label height="40" id="myLabel" left="160" name="myLabel"  style="txt-
align:center" top="100" width="160"/>
<button buttontype="push" height="40" left="160" name="Top Left"  top="200"
width="160">
   <task tasktype="ogscript">ogscript.reposition('myLabel', 0, 0);
   </task>
</button>
```

repositionByPercent

Moves a component to an absolute position, defined as a percentage of container width and height.

Alternatively, you can specify a component's position by pixel. For more information, see reposition on page 248.

Syntax

```
ogscript.repositionByPercent (OID, x percent, y percent, center x, center y);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of the component you want to reposition
x percent	Integer	Yes	Distance from the left, as a percentage of container width
y percent	Integer	Yes	Distance from the top, as a percentage of container height
center x	Boolean	Yes	true — Shows the full width of the object. false — Crops the object if it extends beyond the horizontal boundaries of the container.
center y	Boolean	Yes	true — Shows the full height of the object. false — Crops the object if it extends

Parameter	Type	Required	Description
			beyond the vertical boundaries of the container.

Returns

N/A

Example

In this example, the task associated with the One Quarter button uses the ogscript.reposition function to reposition a label 25% from the left, and 25% from the top. Centering is set to false in both the x and y axes, so if the label overhangs the edges of the container the overhanging portion is not shown.

```
<label height="41" id="myLabel" left="160" name="myLabel"  style="txt-
align:center" top="101" width="160"/>
<button buttontype="push" height="40" left="160" name="One Quarter"  top="200"
width="159">
   <task tasktype="ogscript">ogscript.repositionByPercent('myLabel',  25, 25,
   false, false);
   </task>
</button>
```

reveal

Open a popup with the specified ID, or bring the tab with the specified ID to the foreground.

This function is especially useful for tab sets that have their placement set to the center, meaning that there are no tabs showing for users to click. Using the reveal function is the only way to display the specified tab.

Syntax

```
ogscript.reveal (User Interface ID);
```

Parameters

Parameter	Type	Required	Description
User Interface ID	String	Yes	ID of the popup to open or the tab to bring to the foreground.

Returns

N/A

Example

This example includes a definition for a set of tabs with its position set to center, and uses the ogscript.reveal function to select a particular tab to be shown.

Tip: When tab position is set to center, tabposition="none" in the tab set's XML source code.

```
<tab height="91" left="580" tabposition="none" top="373" width="221">
   <abs id="page1" name="Tab 1"/>
   <abs id="page2" name="Tab 2"/>
   <abs id="page3" name="Tab 3"/>
</tab>

//Select Tab2  ogscript.reveal('page2');
```

runXPath

Execute the given XPath command on the given XML Document or XML Element and return the results as a NodeList.

```
ogscript.runXPath (XPath, Document);
```

or

```
ogscript.runXPath (XPath, Element);
```

Parameters

Parameter	Type	Required	Description
XPath	String	Yes	The XPath command to execute on the given XML Document or XML Element
Document	String	Yes	XML Document on which to execute the given XPath command.
Element		Yes	XML Element on which to execute the given XPath command.

For more information about the required parameters, refer to the following URLs:

- http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/Document.html
- http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/Element.html
- http://docs.oracle.com/javase/6/docs/api/org/w3c/dom/NodeList.html
- http://www.w3schools.com/xpath/

Returns

NodeList

Example

Coming soon.

saveToFile

Saves data to a file. This function is typically used to save a byte array, string, or XML document to a file.

Syntax

```
ogscript.saveToFile(path, data, overwrite);
```

Parameters

Parameter	Type	Required	Description
path	String	Yes	The directory path to the destination file.
data	String, byte[], or XML	Yes	The data to be saved to file.
overwrite	Boolean	Yes	When true, existing file of the same name is overwritten. When false, existing file of the same name is not overwritten.

Returns

Returns true, if data is written successfully; otherwise false.

Example

```
ogscript.saveToFile('files/my-new-file.txt','This is my data',true);
```

sendUDPAsBytes

Converts ASCII string data to a byte array, and sends it as bytes to the specified host/port through UDP. The ASCII data is converted to Hexadecimal bytes, and can consist only of the following characters:

- 0 to 9
- A to F
- Spaces and commas (as delimiters)

Syntax

```
ogscript.sendUDPAsBytes (Host, Port, Data);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host name to send the given data through UDP.
Port	Integer	Yes	Port number on the given host to be sent given data through UDP.
Data	ASCII string	Yes	Data to be converted to bytes and sent through UDP to the specified host/port.

Returns

N/A

Example

```
ogscript.sendUDPAsBytes (myComputer, 7788,'7A, 3C, FF');
```

sendUDPBytes

Send the given data bytes to the specified host/port through UDP.

Syntax

```
ogscript.sendUDPBytes (Host, Port, Data);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host name to send the given Data bytes through UDP.
Port	Integer	Yes	Port number on the given Host to send given Data byte through UDP.
Data	Byte	Yes	Data bytes to send through UDP to the given Host and Port.

Returns

N/A

Example

Coming soon.

sendUDPString

Convert a string to UTF-8 bytes and send the bytes to the provided host/port through UDP.

Syntax

```
ogscript.sendUDPString (Host, Port, Data);
```

Parameters

Parameter	Type	Required	Description
Host	String	Yes	Host name to send the given Data string through UDP.
Port	Integer	Yes	Port number on the given Host to send given Data string through UDP.
Data	String	Yes	Data string to convert to bytes and send through UDP to the given Host and Port.

Returns

N/A

Example

This example uses the sendUDPString function to send a message to a particular host/port.

```
var host = ogscript.getPrivateString('hosts',' Panel.host ');

var port = parseInt(ogscript.getPrivateString('hosts',' Panel.port  '));
var message = "Hello, can you hear me?";

ogscript.sendUDPString(host,port,message);
```

setAnchorPoints

Specifies how an object moves if the user interface is resized for different monitor and window sizes. Anchor points are relative to the container in which they are located (for example, a tab, a split pane, etc.).

The setAnchorPoints function allows you to anchor or release an object to/from the top, left, bottom, or right sides. By setting these values, you can effectively anchor an object to a corner, a side, or the center.

Syntax

```
ogscript.setAnchorPoints (ID, top, left, bottom, right);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the object you want to anchor.
top	Boolean	Yes	true — object is anchored to the top false — object is not anchored to the top
left	Boolean	Yes	true — object is anchored to the left false — object is not anchored to the left
bottom	Boolean	Yes	true — object is anchored to the bottom false — object is not anchored to the bottom
right	Boolean	Yes	true — object is anchored to the right false — object is not anchored to the right

Returns

N/A

Example

The button in this example has a task that anchors an object (with ID 'dialog') to the top left.

```
<button buttontype="push" name="anchorTopLeft">
   <task tasktype="ogscript">ogscript.setAnchorPoints('dialog', true,  true,
   false, false);
   </task>
</button>
```

setSize

Resizes a panel object the to the specified width and height, in pixels. Valid only in <abs/> containers.

Syntax

```
ogscript.setSize (ID, width, height);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the panel object to be resized. Valid only in <abs/> containers.
width	Integer	Yes	New width of the panel, in pixels.
height	Integer	Yes	New height of the panel, in pixels.

Returns

N/A

Example

Coming soon.

setStyle

Set Style parameters for the component with the given ID if it exists. Style commands are additive. They can be added or modified, but not removed.

Tip: To view syntax examples for particular styles, use the PanelBuilder user interface to add the style on the Style tab, and then view the resulting code in the Source tab.

For openGear Style Hints for the available style options, refer to the openGear documentation.

Syntax

```
ogscript.setStyle (Component ID, Style);
```

Parameters

Parameter	Type	Required	Description
Component ID	String	Yes	ID of the Component to style with the given Style parameters.
Style	String	Yes	Style parameters with which to style the given Component.

Returns

N/A

Example 1

This example defines the style of a label, and then makes three style changes.

```
//label definition
<label height="45" id="label1" left="330" name="Change the style of  this
label" style="txt-align:west;" top="100" width="325"/>


//first change - set the background to red
ogscript.setStyle('label1',"bg#FF0000");


//second change - set the text colour to black and text size to big
ogscript.setStyle('label1',"fg#000000;size:big");


//third change - modify the text alignment from left to right
ogscript.setStyle('label1',"txt-align:east");
```

Example 2

This example creates a pre-defined style, and applies it to a component. Pre-defined styles can add or replace a component's style settings, but not remove them.

```
//create a pre-defined style
<style id="Style1" name="Style1" value="size:Big;bg#6F63FB;bdr:etched;"/>
//Add a predefined Style to a component
ogscript.setStyle('label1',"style:Style1")
```

setXML

Dynamically generates UI components through ogscript. Replaces the contents of an element with a string of XML code.

Notes:

- The .grid file is not affected by `setXML()` so its effects do not persist after the CustomPanel is closed.
- `setXML()` is not synchronous with the calling code which can lead to subtle problems. For example, if you used this method to instantiate a customwidget you would not be able to access it on the line immediately following `setXML()` call.
- If used to inject OGLML that includes `<ogscript/>` either directly, or as part of a customwidget it's necessary to release any resources used by the injected objects before subsequently overwriting the same <abs/>. Failure to do this will cause resource leaks, and possibly unwanted behaviour.

Syntax

```
ogscript.setXML (ID, new XML content)
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the component in which you want to replace XML
new XML content	String	Yes	The new XML content

Returns

N/A

Example 1

This simplified example illustrates how to use `ogscript.setXml`.

In this example, the value of the variable `oglml` is XML content (a label named `myLabel`). The `setXML` command populates the `abs` canvas named `Destination` with the value of the `oglml` variable. The result simply displays the label name **myLabel**.

```
<abs>
   <abs id="my-abs" name="Destination" />
   var oglml '<label name="myLabel"/>',
   ogscript setXML ('my-abs', oglml);
</abs>
```

Example 2

This example displays a table with two rows of two columns. The first row contains a parameter named **TYPE LABEL TEXT:** that allows the user to type in a white box. The second row contains a button named **setXml** and a blank label. When the user clicks the **setXml** button, the associated task populates the blank label with whatever text the user typed. The user can redefine the label contents as many times as they want.

In this example, the replacement XML is specified in a variable named `oglml` that uses `params.getValue` to retrieve the typed text from the parameter named `Text for Label`. The button task uses `ogscript.setXml` to populate the label (`id="my-abs"`) with the value of the variable `oglml`.

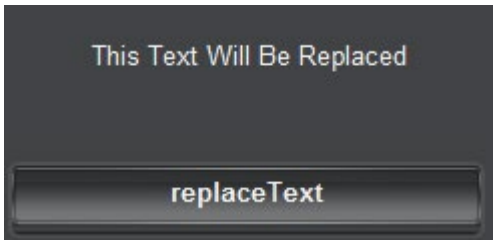
```
<abs contexttype="opengear">
   <meta>
      <params>
         <param access="1" maxlength="0" name="Text for Label" oid="txt"
type="STRING" value="Type Here!" widget="text"/>
      </params>
   </meta>
   <table height="100" left="5" top="9" width="400">
      <tr>
         <label colspan="1" fill="both" header="true" name="TYPE LABEL TEXT:"
rowspan="1" style="txt-align:center;" weightx="1.0" weighty="1.0"/>
         <param colspan="1" expand="true" fill="both" oid="txt" rowspan="1"
style="txt-align:center;" weightx="1.0" weighty="1.0"/>
      </tr>
      <tr>
         <button buttontype="push" colspan="1" fill="both" name="setXml"
rowspan="1" weightx="1.0" weighty="1.0">
            <task tasktype="ogscript">var oglml = '&lt;label name="' +
params.getValue('txt',0) + '" style="txt-align:center;" anchor="center"
```

```
top="0" bottom="0" left="0" right="0" /&gt;'
ogscript.setXML('my-abs', oglml);</task>
         </button>
         <abs anchor="center" colspan="1" fill="both" id="my-abs" rowspan="2"
style="bdr:etched;" weightx="1.0" weighty="1.0"/>
      </tr>
   </table>
</abs>
```

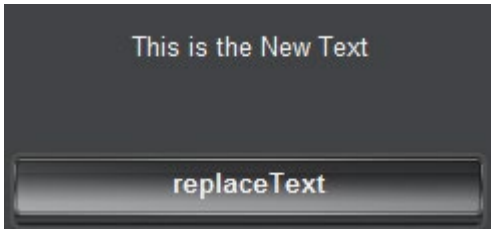
Example 3

This example has a label with text (`<abs id="0x4"> … </abs>`). It also has a button associated with a task that uses `ogscript.setXml` to replace the text by replacing the XML contents of the `<abs>` element. In this example, the replacement XML is contained within the task definition.

Before the button is clicked:



After the button is clicked:




```
<abs>
   <abs id="0x4">
      <label height="59" id="0x2" left="61" name="This Text Will Be Replaced"
      style="txt-align:center" top="40" width="238"/>
   </abs>
   <button buttontype="push" height="40" id="0x3" left="59"
   name="replaceText" top="121" width="240">
      <task tasktype="ogscript">ogscript.setXml('0x4', '&lt;label
      height="59" id="0x2" left="61" name="This is the New Text"  style="txt-
      align:center" top="40" width="238"/&gt;');
      </task>
   </button>
</abs>
```

toBottom

Displays the object below all others in the same container. Object display is layered. If objects overlap, higher layers are drawn over lower layers.

Syntax

```
ogscript.toBottom (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID object to be sent to the bottom

Returns

N/A

Example

This example includes two labels occupying the same position. LabelOne is defined second in the code, so it appears on top and is therefore visible. Button One runs a task that uses ogscript.toBottom to send Label One to the bottom of the stack. This makes Label Two visible. Button Two sends Label Two to the bottom.

```
<abs>
<label height="317" id="labelTwo" left="100" name="Label Two"
style="size:Biggest;bg#D92648;txt-align:center;" top="100"  width="350"/>
<label height="317" id="labelOne" left="100" name="Label One"
style="size:Biggest;bg#selectbg;txt-align:center;" top="100"  width="350"/>
<button buttontype="push" height="40" id="oneBottom" left="150"  name="Button
One" style="bg#selectbg;txt-align:center;" top="450"  width="100">
   <task tasktype="ogscript">ogscript.toBottom('labelOne');
   </task>
</button>

<button buttontype="push" height="40" id="twoBottomn" left="300"  name="Button
Two" style="bg#D92648;txt-align:center;" top="450"  width="100">
   <task tasktype="ogscript">ogscript.toBottom('labelTwo');
   </task>
</button>
</abs>
```

toTop

Displays the object above all others in the same container. Object display is layered. If objects overlap, higher layers are drawn over lower layers.

Syntax

```
ogscript.toTop (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID object to be sent to the top

Returns

N/A

Example

This example includes two labels occupying the same position. LabelTwo is defined second in the code, so it appears on top and is therefore visible. Button One runs a task that uses ogscript.toTop to send Label One to the top of the stack. This makes Label One visible. Button Two sends Label Two to the top.

```
<abs>
    <label height="317" id="labelOne" left="100" name="Label One"
    style="size:Biggest;bg#selectbg;txt-align:center;" top="100"  width="350"/>
    <label height="317" id="labelTwo" left="100" name="Label Two"
    style="size:Biggest;bg#D92648;txt-align:center;" top="100"  width="350"/>
    <button buttontype="push" height="40" id="oneTop" left="150" name="Button
    One" style="bg#selectbg;txt-align:center;" top="450"  width="100">
      <task tasktype="ogscript">ogscript.toTop('labelOne');
      </task>
    </button>
    <button buttontype="push" height="40" id="twoTop" left="300" name="Button
    Two" style="bg#D92648;txt-align:center;" top="450"  width="100">
      <task tasktype="ogscript">ogscript.toTop('labelTwo');
      </task>
    </button>
</abs>
```

upload

Open the File Upload dialog with the specified file.

Syntax

`ogscript.upload (`*Filename*`);`

Parameters

Parameter	Type	Required	Description
Filename	String	Yes	Name of the file with which to open the File Upload dialog box.

Returns

N/A

Example

Coming soon.

params Object

In ogScript, use the params object to access functions to interact with openGear Device parameters and constraints. The params object is also used to manipulate parameters stored in the .grid file.

The params object is accessible when a CustomPanel is associated with an openGear device or XML data file (.grid file). Scripts referencing a device must follow beneath the referenced device in the XML hierarchy.

To call an openGear Device function, use:

```
params.function name (parameters);
```

For example:

```
var data = params.getValue (0x12, 0);
```

Some params functions return a ParamScriptable object named **this**, which contains several methods that enable you to manipulate parameters. For more information, see ParamScriptable Object on page 277.

params Functions

The following table lists the functions of the params object. Detailed descriptions appear after the table. If you are reading this document on-screen, click a function name in the table to view its description.

Function	Parameters	Returns	Description
createCopy	Source OID Destination OID	ParamScriptable	Creates a copy of the parameter. The duplicate parameter is independent of the base parameter. Changing the value of one does not affect the other.
createIntChoiceConstraint	[choices]	N/A	Creates a choice constraint (which is a set of key/value pairs) for use in toggle buttons, combo box, radio buttons, etc. The choice constraint you create here can be used to replace a constraint for a parameter.
createLinkedCopy	Source OID Destination OID	ParamScriptable	Creates a copy of the parameter that is linked to the base parameter: Changing the value of the base parameter also changes the value of the duplicate parameter. Changing the value of the duplicate parameter does not affect the value of the base parameter.
createMultiSet	String [OID] Integer [Index] Object [Value]	multiset object	Replaces multiple parameter values all at once.
createParam	JSON parameter definition	N/A	Creates a parameter based on a JSON parameter definition.
deleteParam	OID of parameter to delete	N/A	Deletes the specified parameter.
getAllValues	String [OID]	The entire array of values within the parameter.	Retrieves the entire array of values within the parameter.
getConstraint	String [OID]	Constraint	Get the constraint from the parameter with the specified OID.
getDeviceStatus	String [OID]	Device status information	Checks the status of a device and returns an Integer value indicating that status:

Function	Parameters	Returns	Description
			0 - good - warning - error - unknown
getElementCount	Integer [Context ID] String [OID] Integer [Index]	ParamScriptable	Get the information about an element in a parameter with the specified OID.
getIdentifiedConstraint	String [ID]	String	Get the constraint with the specified ID. If the ID is an external object URL, get the constraint defined in the specified external object.
getParam	String [Context ID] String [OID] Integer [Index]	ParamScriptable	Gets information about an element in the parameter with the specified Object ID.
getParam (OID, Index).remove	String or Integer [OID] Integer [Index]	N/A	Removes a parameter element. If the parameter is an array with more than one element, the element at the index location is removed.
getStream	String [OID]	Boolean	Check whether streaming of parameter values to XPression is enabled.
getValue	String [OID] Integer [Index]	String	Get the value of a parameter with the specified OID. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.
getValueAsString	String [OID] Integer [Index]	String	Get a string representation of an element in a parameter with the specified OID.
isDeviceOnline	String [OID]	Online status of the device as Boolean	Queries a device to determine whether it is online.
isPrivateParamContext	N/A	Boolean	Returns true if local OGLML-based parameters are operating disconnected from the real device.
replaceConstraint	String [OID] String [Constraint ID]	N/A	Replace the constraint for the parameter with the specified OID with the constraint with the specified constraint ID.
replaceViewConstraint	String [view OID] String [constraint object]	N/A	Replaces the constraint object of a parameter view.
resetAllValues	String [parameter OID]	N/A	If the specified parameter is a copy of a base parameter, this function resets the parameter's values to those of the base parameter.
setAccess	String [OID], Integer [Access]	N/A	Set the access level of the parameter with the provided OID.
setAllValues	OID Object[] Values	N/A	For an array parameter, replaces the current array with the new array.
setMenuState	Integer [Static Menu ID], Integer [Menu State]	N/A	Set the menu state of the menu with the specified static menu ID.
setPrivateParamContext	Boolean [Value]	N/A	true — disconnect parameters defined in the OGLML document from the device.

Function	Parameters	Returns	Description
			false — re-connect parameters defined in the OGLML document from the device.
setStream	String [OID] Boolean [true/false]	N/A	Controls whether the parameter streams its values to XPression when XPression streaming is on. When true, streaming is ON. When false, streaming is OFF.
setValue	String [OID], Integer [Index], Object [Value]	N/A	Set the value of an element in a parameter with the provided OID to the provided value.
setValueRelative	String [OID], Integer [index], Integer [change in value]	N/A	Changes the value of a parameter. If the value is a string, it is replaced. If it is a float or int, the specified value is added to the current value.
	String		Subscribes to a device with subscriptions support.
	String		Unsubscribes to a device with subscriptions support.
toOid	String (OID)	N/A	Creates an OID object.

createCopy

Creates a copy of the parameter. The duplicate parameter is independent of the base parameter. Changing the value of one does not affect the other.

Syntax

`params.createCopy (`*Source OID, Destination OID*`);`

Parameters

Parameter	Type	Required	Description
Source OID	String	Yes	The OID of the parameter to copy
Destination OID	String	Yes	The OID of the new parameter.

Returns

Returns ParamScriptable. For more information, see ParamScriptable Object on page 277.

Example

Coming soon.

createIntChoiceConstraint

Syntax

`params.createIntChoiceConstraint (`*Choices*`);`

Parameters

Parameter	Type	Required	Description
Choices	String	Yes	Name of the array variable that contains the choices.

Returns

N/A

Example

Coming soon.

createLinkedCopy

Creates a copy of the parameter that is linked to the base parameter:

- Changing the value of the base parameter also changes the value of the duplicate parameter.
- Changing the value of the duplicate parameter does not affect the value of the base parameter.

Syntax

`params.createLinkedCopy (`*Source OID, Destination OID*`);`

Parameters

Parameter	Type	Required	Description
Source OID	String	Yes	The OID of the parameter to copy
Destination OID	String	Yes	The OID of the new parameter.

Returns

Returns ParamScriptable. For more information, see ParamScriptable Object on page 277.

Example

Coming soon.

createMultiSet

Changes the values of multiple parameters at once. This function will return a multiSetScriptable Object.

Syntax

`params.createMultiSet (`*OID, Index, Value*`);`

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of object of interest.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.
Value	Object	Yes	New value for the OID.

Returns

Multiset object.

Example

In the following example, four parameters named "Value 1" through "Value 4" are created with text values that are displayed on buttons when the example is run. When the user taps the Multi-Set button, the params.createMultiSet function changes the parameter values to those referenced by the function.

```
<abs contexttype="opengear" gridsize="20" style="">
   <meta>
      <params>
```

```
            <param access="1" maxlength="0" name="Value 1" oid="Value_1"
            stateless="true" type="STRING" value="Original Value 1"
            widget="100"/>
            <param access="1" maxlength="0" name="Value 2" oid="Value_2"
            stateless="true" type="STRING" value="Original Value 2"
            widget="100"/>
            <param access="1" maxlength="0" name="Value 3" oid="Value_3"
            stateless="true" type="STRING" value="Original Value 3"
            widget="100"/>
            <param access="1" maxlength="0" name="Value 4" oid="Value_4"
            stateless="true" type="STRING" value="Original Value 4"
            widget="100"/>
        </params>
    </meta>
    <param expand="true" height="40" left="20" oid="Value_1" top="20"
    width="340"/>
    <param expand="true" height="40" left="20" oid="Value_2" top="80"
    width="340"/>
    <param expand="true" height="40" left="20" oid="Value_3" top="140"
    width="340"/>
    <param expand="true" height="40" left="20" oid="Value_4" top="200"
    width="340"/>
    <button buttontype="push" height="60" left="20" name="Multi-Set" top="260"
    width="340">
        <task tasktype="ogscript">
            var multi = params.createMultiSet(); multi.setValue('Value_1', 0,
            'Multi-set 1');
            multi.setValue('Value_2', 0, 'Multi-set 2');
            multi.setValue('Value_3', 0, 'Multi-set 3');
            multi.setValue('Value_4', 0, 'Multi-set 4'); multi.execute();
        </task>
    </button>
</abs>
```

createParam

Creates a parameter based on a JSON parameter definition.

Syntax

`params.createParam (`*JSON parameter definition*`);`

Parameters

Parameter	Type	Required	Description
JSON parameter definition	String	Yes	JSON definition of the parameter.

Returns

N/A

Example

Coming soon.

deleteParam

Deletes the specified parameter.

Syntax

```
params.deleteParam (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of parameter to delete

Returns

N/A

Example

Coming soon.

getAllValues

Retrieves the entire array of values from a parameter.

Syntax

```
params.getAllValues (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	The OID of the parameter

Returns

The entire array of values from the parameter.

Example

Coming soon.

getConstraint

Get the constraint from the parameter with the specified Object ID.

Syntax

```
params.getConstraint (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of the object of interest.

Returns

Constraint

Example

Coming soon.

getDeviceStatus

Checks the status of a device and returns an Integer value indicating that status.

Syntax

```
params.getDeviceStatus (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of the device to be queried.

Returns

Device status, as an Integer:

- **0** — good
- **1** — warning
- **2** — error
- **3** — unknown

Example

Coming soon.

getElementCount

Gets the number of elements in a parameter array.

Syntax

```
params.getElementCount (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	The OID of the parameter.

Returns

The number of elements in the parameter array, as an Integer.

Example

Coming soon.

getIdentifiedConstraint

Get the constraint with the specified ID. If the ID is an external object URL, get the constraint defined in the specified external object.

Syntax

```
params.getIdentifiedConstraint (ID);
```

Parameters

Parameter	Type	Required	Description
ID	String	Yes	ID of the constraint of interest.

Returns

String

Example

Coming soon.

getParam

Gets information about an element in the parameter with the specified Object ID.

Syntax

```
params.getParam (Context ID, OID, Index);
```

Parameters

Parameter	Type	Required	Description
Context ID	String	No	The context ID of the component that contains the parameter of interest.
OID	String	Yes	Object ID of the object of interest.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.

Returns

ParamScriptable

Example

Coming soon.

getParam (OID, Index).remove

Removes a parameter element. If the parameter is an array with more than one element, the element at the index location is removed.

Syntax

```
params.getParam ([oid], [index]).remove();
```

Parameters

Parameter	Type	Required	Description
OID	String or Integer	Yes	OID can be a string or an integer, depending on how the parameter is defined.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0.

Returns

N/A

Example

Coming soon.

getStream

Checks whether streaming of parameter values to XPression is enabled for the parameter.

Syntax

```
params.getStream (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of the parameter

Returns

Boolean, to indicate whether streaming is enabled.

Example

Coming soon.

getValue

Gets the value of a parameter with the specified Object ID.

Syntax

```
params.getValue (OID, Index);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of object of interest.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.

Returns

String

Example

```
var data = params.getValue (0x12,0);
```

getValueAsString

Gets a string representation of an element in a parameter with the specified Object ID.

Syntax

```
params.getValueAsString (OID, Index);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of the object of interest.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.

Returns

ParamScriptable

Example

Coming soon.

isDeviceOnline

Queries a device to determine whether it is online.

Syntax

```
params.isDeviceOnline (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of device to query.

Returns

Online status of the device.

Example

Coming soon.

isPrivateParamContext

Returns true when the local OGLML-based parameters are operating disconnected from a real device. Changes and values are not sent to or fetched from the device if the parameter is defined in the OGLML document.

Syntax

```
params.isPrivateParamContext ();
```

Parameters

N/A

Returns

Boolean

Example

Coming soon.

replaceConstraint

Replace the constraint for the parameter with the specified Object ID with the constraint with the specified constraint ID. If the ID is an external object URL, replace the constraint with the constraint specified by the external object.

Syntax

`params.replaceConstraint (`*OID, Constraint ID*`);`

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of object of interest.
Constraint ID	String	Yes	ID of the constraint with which to replace the constraint for the parameter with the specified Object ID.

Returns

N/A

Example

Coming soon.

replaceViewConstraint

Replaces the constraint object of a parameter view.

Syntax

`params.replaceViewConstraint (`*view OID, constraint object*`);`

Parameters

Parameter	Type	Required	Description
view OID	String	Yes	OID of the view.
constraint object	String	Yes	constraint object to use.

Returns

N/A

Example

Coming soon.

resetAllValues

If the specified parameter is a copy of a base parameter, this function resets the parameter's values to those of the base parameter.

Syntax

`params.resetAllValues (`*parameter OID*`);`

Parameters

Parameter	Type	Required	Description
parameter OID	String	Yes	The OID of the parameter.

Returns

N/A

Example

Coming soon.

setAccess

Set the access level of the parameter with the specified Object ID.

Syntax

```
params.setAccess (OID, Access);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of object of interest.
Access	Integer	Yes	Access level to set for the specified OID. The available access levels are as follows: 0 — Read Only 1 — Read and Write

Returns

N/A

Example

Coming soon.

setAllValues

For an array parameter, replaces the current array with a new array.

Syntax

```
params.setAllValues([oid], [array]);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	The OID of the parameter.
Array	String	Yes	The new array.

Returns

N/A

Example

Coming soon.

setMenuState

Set the menu state of the menu with the provided static menu ID.

Syntax

```
params.setMenuState (Static Menu ID, Menu State);
```

Parameters

Parameter	Type	Required	Description
Static Menu ID	Integer	Yes	ID of the menu of interest.
Menu State	Integer	Yes	Menu state to set for the specified Static Menu ID. The available menu states are as follows: 0 — Hidden 1 — Disabled 2 — Normal

Returns

N/A

Example

Coming soon.

setPrivateParamContext

Control the context between the parameters defined in the OGLM document and a device. This function has no impact on parameters that are only defined on the device or only defined in the OGLML document.

Syntax

```
params.setPrivateParamContext (Value);
```

Parameters

Parameter	Type	Required	Description
Value	Boolean	Yes	The available contexts are as follows: true — disconnect parameters defined in the OGLML document from the device. false — re-connect parameters defined in the OGLML document from the device.

Returns

N/A

Example

Coming soon.

setStream

Controls whether a parameter streams its values to XPression when XPression streaming is on.

Syntax

`params.getStream (`*OID, true/false*`);`

Parameters

Parameter	Type	Required	Description
OID	String	Yes	OID of the parameter
true/false	Boolean	Yes	When true, streaming is ON. When false, streaming is OFF.

Returns

N/A

Example

Coming soon.

setValue

Set the value of a parameter for the provided Object ID.

Syntax

`params.setValue (`*OID, Index, Value*`);`

Parameters

Parameter	Type	Required	Description
OID	String	Yes	Object ID of object of interest.
Index	Integer	Yes	Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.
Value	Object	Yes	New value for the OID.

Returns

N/A

Example 1

```
// Set the parameter to 3: params.setValue (0x12,0,3);
```

Example2

```
// Set the value to 3 greater than it was.
var data = getValue (0x12,0); params.setValue (0x12,0,data + 3);
```

Example3

```
// Set the value of Param_A to match the value of Param_B
params.setValue('Param_A', 0, params.getValue('Param_B', 0));
```

setValueRelative

Increments or decrements a numeric value by a specified amount.

Syntax

`params.setValueRelative (`*OID, Index, Change in value*`);`

Parameters

Parameter	Type	Required	Description
OID	String or Integer	Yes	The OID of the object of which you want to change the value.
Index	Integer	Yes	Position of data in the parameter.
Change in Value	Integer	Yes	Amount by which the value is incremented. To decrement the value, use a negative integer.

Returns

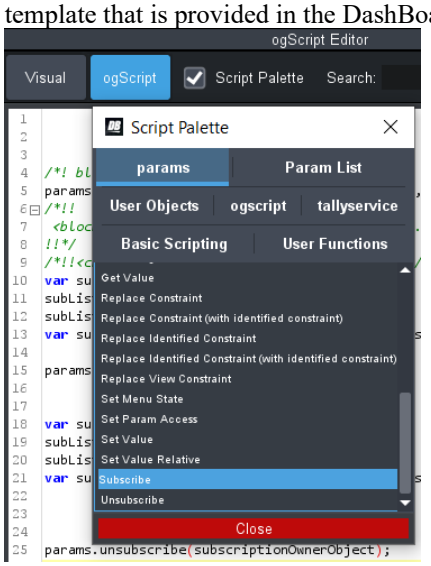
N/A

Example

Coming soon.

subscribe

You can use the **subscribe** or **unsubscribe** command templates or code syntax below to add the subscription list to a DashBoard device panel. You must add support to subscribe and/or unsubscribe from parameter updates in the device panel's OGLML structure. You can also use the command template that is provided in the DashBoard PanelBuilder Script Palette under **params**.



Syntax

```
var subList = new Array();
   subList.push("oid1");
   subList.push("oid2");
var subscriptionOwnerObject = params.subscribe(subList, callback);
```

Parameters	Type	Required	Description

subList, callback	[Array of strings, callback]	*Required to support devices with support for subscriptions.	Subscribes to parameters with the provided OIDs. To support subscriptions, the subscribe function is required to subscribe to parameter updates in the device panel's OGLML structure. You can also use the DashBoard PanelBuilder Script Palette to add the subscribe or unsubscribe functions using the template.

Returns

Returns `subscriptionOwnerObject` for later use to unsubscribe.

"params.subscribe" Example

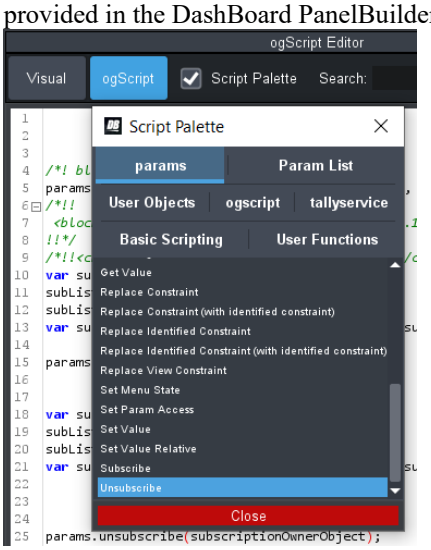
```
 <task tasktype="ogscript">
          var subList= new Array();
             subList.push("deviceoptions.speakerlevel");
             subList.push("db.touch.version.*");
          var subscriptionOwnerObject = params.subscribe(subList, callback);
   ogscript.putObject('my-subscription-owner-object',
subscriptionOwnerObject);
</task>
```

Explanation

In this example, the `ogscript.putObject` is used to retain the result of the params.subscribe function, which is later used to unsubscribe.

unsubscribe

You can use the subscribe or unsubscribe command templates or code syntax below to add the subscription list to a DashBoard device panel. You must add support to subscribe and/or unsubscribe from parameter updates in the device panel's OGLML structure. You can also use the template that is provided in the DashBoard PanelBuilder Script Palette.



Syntax

```
params.unsubscribe(subscriptionOwnerObject);
```

Function	Type	Returns	Description
unsubscribe	[subscriptionOwnerObject]	N/A	Unsubscribes from the OIDs provided by the subscriptionOwnerObject.

"params.subscribe" Example

```
<task tasktype="ogscript">
          var subscriptionOwnerObject = ogscript.getObject('my-subscription-
owner-object');
          params.unsubscribe(subscriptionOwnerObject);
</task>
```

Explanation

In the subscribe example above, the `ogscript.putObject` is used to retain the result of the params.subscribe function and ogscript.getObject fetches it when we want to unsubscribe (params.unsubscribe). You can see that the subscribe response object is used to unsubscribe.

Now that you have successfully implemented subscriptions support, make sure that you leverage the built-in automations within DashBoard to support subscriptions.

toOid

Creates an OID object.

Syntax

```
params.toOid (OID);
```

Parameters

Parameter	Type	Required	Description
OID	String	Yes	The value of the new OID object.

Returns

N/A

Example

This example is a function that uses the toOid function to create an OID with the value **`'my.special.oid'`**, then uses the **`getOid`** function to return the OID value.

```
function lookForSpecificOid(myParam)
{
var myOID = params.toOid('my.special.oid');  return myParam.getOid() == myOID;
}
```

ParamScriptable Object

Some params functions return a ParamScriptable object named `this`, which contains several methods that enable you to manipulate parameters.

In ogScript, use methods of the `this` object to manipulate parameters. To call a general-purpose function, use:

`this.`*methodname* `(`*parameters*`);`

For example:

```
this.replaceConstraint ("0.0;100.0;0.0;100.0;1");
```

The following table lists the methods of the ParamScriptable object.

Method	Parameters	Returns	Description
deleteParam	N/A	N/A	Deletes the parameter
getConstraint	N/A	Returns the parameter constraint	Gets the parameter constraint
getAllValues	N/A	The entire array of values within the parameter.	Retrieves the entire array of values within the parameter.
getElementCount	N/A	The number of elements in the parameter array, as an Integer.	Gets the number of elements in the parameter array.
getIndex	N/A	Returns the array index of the current element	Gets the array index of the current element
getOid	N/A	Returns the OID of the changed parameter	Gets the OID of the changed parameter
getValue	N/A	Returns the value of the changed element	Gets the value of the changed element
getValueAsString	N/A	Returns a string representation of the changed value	Gets a string representation of the changed value
getValueAt	Integer [index]	Returns a string representation of the value at the provided index	Gets a string representation of the value at the provided index
getValueAtAsString	Integer [index]	Returns a string representation of the value at the provided index	Gets a string representation of the value at the provided index
setValue	String [value]	N/A	Sets the value of the changed element to the provided value.
getName	N/A	Returns the parameter name	Gets the parameter name
replaceConstraint	String [Constraint]	N/A	Replaces the parameter's constraint to the provided value
remove	N/A	N/A	Removes the current array element
isArrayParameter	N/A	Returns true if the parameter is an array element	Checks whether the parameter is an array element
resetAllValues	N/A	N/A	If the parameter is a copy of a base parameter, this function resets the parameter's values to those of the base parameter.
setValueAt	Integer [index] String [value]	N/A	Sets the value of element at the provided index to the provided value.
getElementCount	N/A	Returns the number of elements in the array	Gets the number of elements in the array

rosstalk Object

In ogScript, use the rosstalk object to communicate over the network to other devices that speak RossTalk protocol. Functions in the rosstalk object are typically set through a user interface.

Also see, rosstalkex Object.

To call a general-purpose function, use:

`rosstalk.`*function name* `(`*parameters*`);`

For example:

```
rosstalk.setHost (Server01);
```

The following table lists the functions of the rosstalk object.

Function	Parameters	Returns	Description
setHost	String [Host]	N/A	Set a default host to use for RossTalk commands where no host has been defined.
getHost	N/A	String	Get the default host previously defined.
setPort	Integer [Port]	N/A	Set a default port to use for RossTalk commands where no host has been defined.
getPort	N/A	Integer	Get the default port previously defined.
sendAsBytes	String [Host], Int [Port], String [Bytes as Hex String]	N/A	Equivalent of calling: sendAsBytes(host, port, bytes, null);
sendAsBytes	String [Host], Int [Port], String [Bytes as Hex String], Function [Callback]	N/A	Convert bytes from string (where string is formatted as ASCII representations of bytes e.g. "FDDFEAAE12F9…") and send them to the provided host at the provided port. Invoke the callback function when done.
sendAsBytesWithResponse	String [Host], Int [Port], String [Bytes as Hex String], String [responseBytes], Function [Callback]	Response message provided by the recipient.	Convert bytes from string (where string is formatted as ASCII representations of bytes e.g. "FDDFEAAE12F9…") and send them to the provided host at the provided port. Invoke the callback function when done. The [responseBytes] string, when received from the recipient, indicates the end of the response message.
sendBytes	String [Host], Int [Port], Byte[] [Data to Send], Function [Callback]	N/A	Send the provided bytes to the provided host at the provided port. Invoke the callback function when done.
sendBytesWithResponse	String [Host], Int [Port], Byte[] [Data to Send], Byte [responseTerminator] Function [Callback]	Response message provided by the recipient.	Send the provided bytes to the provided host at the provided port. Invoke the callback function when done. The [responseTerminator] byte, when received from the

Function	Parameters	Returns	Description
			recipient, indicates the end of the response message.
sendMessage	String [RossTalk Command]	N/A	Equivalent of calling: sendMessage (getHost(), getPort(), RossTalk Command, null);
sendMessage	String [RossTalk Command], Function [Callback]	N/A	Equivalent of calling: sendMessage (getHost(), getPort(), RossTalk Command, Callback);
sendMessage	String [Host], Int [Port], String [RossTalk Command]	N/A	Equivalent of calling: sendMessage (Host, Port, RossTalk Command, null);
sendMessage	String [Host], Int [Port], String [RossTalk Command] Function [Callback]	N/A	Send the provided string as UTF-8 followed by CRLF bytes to the provided host at the provided port. Invoke the callback function when done.
sendMessageWithResponse	String [Host], Int [Port], String [RossTalk Command], String [responseTerminator], Function [Callback]	Response message provided by the recipient.	Send the provided string as UTF-8 followed by CRLF bytes to the provided host at the provided port. Invoke the callback function when done. The [responseTerminator] string, when received from the recipient, indicates the end of the response message.

rosstalkex Object

In ogScript, you can use the rosstalkex object to communicate over the network to other devices that speak RossTalkEx protocol. You can use RossTalk Ex commands to trigger specific events, or to send generic RossTalkEx commands. You can also send RossTalk commands through RossTalkEx, but you cannot do the reverse.

DashBoard sends RossTalkEx commands to XPression using an authenticated RossTalkEx connection. This differs from the method that other RossTalk commands use, which is an open TCP protocol.

To call a general-purpose function, use:

`rosstalkex.`*function name* `(`*parameters*`);`

For example:

```
rosstalkex.sendMessage("10.3.2.1", 8020, "DATALINQKEY 101:k1:v1", null);
```

The following table lists the functions of the rosstalkex object.

Function	Parameters	Returns	Description
getConnection	String [Host], Integer [Port], Boolean [Creation Flag]	If the handshake is successful, the connection object is returned. If the handshake is not successful, a null value is returned.	This is an optional command that users can use to open and authenticate a connection to an XPression. An authentication request will be sent. Once a connection is opened, it will remain open for the life of the panel (assuming it is not explicitly closed by either end). If users use FALSE as the creation flag, then the getConnection function will simply return the existing connection if it was previously opened, or null if it was not. The creation flag command is optional, because when the sendMessage or sendMessageWithResponse commands are executed in a panel, if the connection with the host is not open, then the getConnection function is first executed automatically. Once a connection is established, the message is sent using the XML API wrapper.
sendMessage	String [Host], Integer [Port], String [Message], Function [Callback]	If the authentication is not successful, no message is sent and an error message is thrown, otherwise, nothing is returned.	ADD DEFINITION. This command calls getConnection to initiate an authenticated connection. Once the connection is open, subsequent calls will not automatically trigger the getConnection function.

e

robot Object

In ogScript, use the robot object to communicate with CamBot robotic cameras through the CamBot PC User Interface. Functions in the robot object are typically set through a user interface.

To call a general-purpose function, use:

`robot.`*function* `name (`*parameters*`);`

For example:

```
robot.setHost (Server01);
```

The following table lists the functions of the robot object.

Function	Parameters	Returns	Description
setHost	String [Host]	N/A	Set a default host to use for CamBot commands where no host has been defined.
getHost	N/A	String	Get the default host previously defined.
setPort	Integer [Port]	N/A	Set a default port to use for CamBot commands where no host has been defined.
getPort	N/A	Integer	Get the default port previously defined.
sendCambot	String [CamBot Command]	N/A	Equivalent of calling: sendCambot (getHost(), getPort(), command, null)
sendCambot	String [CamBot Command] Function [Callback]	N/A	Equivalent of calling: sendCambot (getHost(), getPort(), CamBot Command, Callback);
sendCambot	String [Host], Int [Port], String [CamBot Command]	N/A	Equivalent of calling: sendCambot (Host, Port, CamBot Command, null);
sendCambot	String [Host], Int [Port], String [CamBot Command] Function [Callback]	N/A	Send the provided CamBot command to the provided host at the provided port. Invoke the callback function when done. Callback function signature: Function (Boolean success, String sentData, String receivedData, Exception javaException)

vdcp Object

In ogScript, use the vdcp object to communicate with BlackStorm video servers. Functions in the vdcp object are typically set through a user interface.

To call a general-purpose function, use:

`vdcp.function` *name* `(`*parameters*`);`

For example:

```
vdcp.setHost (Server01);
```

The following table lists the functions of the vdcp object.

Function	Parameters	Returns	Description
setHost	String [Host]	N/A	Set a default host to use for VDCP commands where no host has been defined.
getHost	N/A	String	Get the default host previously defined.
setPort	Integer [Port]	N/A	Set a default port to use for VDCP commands where no host has been defined.
getPort	N/A	Integer	Get the default port previously defined.
activeClip	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	Fetch the active clip ID for the provided channel from the server at the provided host/port. Invoke the callback with the active clip ID when done. Callback function signature: Function (Boolean success, String sentCommand, String resultString, Exception javaException)
clipDuration	String [Host], Int [Port], Int [Channel], String [ClipID], Function [Callback]	N/A	Fetch the duration [HH:MM:SS:FF] of the clip with the given ID. Invoke the callback with the clip duration when done. Callback function signature: Function (Boolean success, String sentCommand, String resultString, Exception javaException)
continuePlay	String [Host], Int [Port], Int [Channel]	N/A	Sends the vdcp continuePlay command.
cueClip	String [Host], Int [Port], Int [Channel]	N/A	
cueClip	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	
fastForward	String [Host], Int [Port], Int [Channel]	N/A	
fastForward	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	
listClips	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	
pause	String [Host], Int [Port], Int [Channel]	N/A	
pause	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	
play	String [Host], Int [Port], Int [Channel]	N/A	Sends the vdcp variPlay command.
rewind	String [Host], Int [Port], Int [Channel]	N/A	
rewind	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	
stop	String [Host], Int [Port], Int [Channel]	N/A	
stop	String [Host], Int [Port], Int [Channel], Function [Callback]	N/A	

nkScript Object

In ogScript, use the nkScript object to control NK Router OGLML tags used in Switchboard virtual

control panels. Functions in the nkScript object are usually set through a user interface.

The nkScript global object is only accessible in OGLML contexts that are declared as having a NK Router context type or are beneath such a context in the OGML document hierarchy.

To call a general-purpose function, use:

`nkscript.`*function name* `(`*parameters*`);`

For example:

```
nkscript.setHost (Server01);
```

The following table lists the functions of the nkscript object.

Function	Parameters	Returns	Description
convertCommaSeperatedLevelsToMask	String [Levels], Boolean [SearchTags]	Long Levelmask	Allows conversion of a list of levels to the appropriate level mask. Level mask is a bit field where you can have up to 32 levels set 'on' at a time. SearchTags should always be 'true'.
doSwitch	N/A	Boolean	Equivalent of calling: doSwitch(getActiveDst(), getActiveSrc(), getLevelMask());
doSwitch	Int [Dst], Int [Src], Long [Levels]	Boolean	Do a switch on the active IPS to route the given destination to the given source on the given levels.
doSwitchWithLabels	String [Destination], String [Source], String [Levels]	Boolean	Allows you to switch between levels by name.
getActiveDst	N/A	Int	Get the active destination number (0-indexed). Returns -1 if there is no active destination.
getActiveDstName	N/A	String	Get the name of the active destination (from the switchboard configuration). Returns null if there is no active destination.
getActiveIPS	N/A	String	Get the serial number of the active IPS.
getActiveIPSName	N/A	String	Get the name of the active IPS.
getActiveSrc	N/A	Int	Get the active source number (0-indexed). Returns -1 if there is no active source.
getActiveSrcName	N/A	String	Get the name of the active src (from the switchboard configuration). Returns null if there is no active source.
getActiveSystem	N/A	NKSystem	Get the currently active NKSystem.
getDstName	String [Source]	String	Get the destination name of the given source.
getLevelMask	N/A	Long	Get the current level mask (as a bit field) Level mask is a bit field where

Function	Parameters	Returns	Description
			you can have up to 32 levels set 'on' at a time.
getLevelName	String [Source]	String	Get the level name of the given source.
getProtectStatus	String [Destination], String [Levels]	Boolean	Get the protect status of the destination level.
getSrcName	String [Source]	String	Get the source name.
getStatus	String [Destination], String [Level]	Int	Get the status of the given destination level.
isLevelActive	Int [Level Num]	Boolean	Is the current level active. Equivalent to asking: levelMask & (1 << levelNum) != 0;
isMCFlag	N/A	Boolean	Is the Machine Control flag set.
isProtected	N/A	Boolean	Verifies whether the active destination is protected or not.
isProtected	Int [Destination], Long [Levels]	Boolean	Verifies whether the given destination is protected; or not.
isProtectedByMe	N/A	Boolean	Is the active destination protected by this virtual panel.
isSrcActive	Int [Src]	Boolean	Verifies whether the given source is active on the active destination of all levels.
isSrcActive	Int [Dst], Int [Src], Long [Levels]	Boolean	Verifies whether the provided source is active on the specified destination and level mask.
isVirtual	N/A	Boolean	Verifies whether virtual routing is in use (for switch commands and status requests).
setActiveDst	Int [Dst]	N/A	Set the active destination (0-indexed).
setActiveIPS	String [Serial]	Boolean	Set the IPS with the given serial number as the active IPS to receive commands and send status. Deactivate any currently active IPS.
setActiveSrc	Int [Src]	N/A	Set the active source (0-indexed).
setLevelActive	Int [Level Num], Boolean [Active]	Boolean	Set the given level as active.
setLevelMask	Long [Level Mask]	N/A	Set the complete level mask bitfield.
setMCFlag	Boolean	Boolean	Set the Machine Control flag to true or false.
setProtected	Boolean	Boolean	Request the router to protect the active destination.
setVirtual	Boolean	Boolean	Set virtual routing on/off for switch commands and status

Function	Parameters	Returns	Description
			requests.
verifyConfiguration	N/A	Boolean	Re-activate the current IPS.

Appendices

In This Section

See appendices in the bookmark navigation.

Appendix A: Widget Hint Definitions

```
// widget hints for all parameter types
#define WIDGET_DEFAULT                  0  // let DashBoard decide
#define WIDGET_TEXT_DISPLAY             1  // display as text, read only
#define WIDGET_HIDDEN                   2  // do not display

// widget hints for numeric types with NULL_CONSTRAINT or RANGE_CONSTRAINT
#define WIDGET_SLIDER_HORIZONTAL        3  // slider (RANGE only)
#define WIDGET_SLIDER_VERTICAL          4  // slider (RANGE only)
#define WIDGET_SPINNER                  5  // spinner
#define WIDGET_TEXTBOX                  6  // numeric entry field
#define WIDGET_PROGRESS_BAR            17  // progress bar (RANGE only)
#define WIDGET_AUDIO_METER             19  // audio meter (RANGE only)
#define WIDGET_MENU_POPUP              20  // popup menu with the ID(INT ONLY)
#define WIDGET_TIMER                   21  // countdown/up timer (RANGE only)
#define WIDGET_SLIDER_H_NO_LABEL       24  // unlabeled slider (RANGE only)
#define WIDGET_SLIDER_V_NO_LABEL       25  // unlabeled slider (RANGE only)
#define WIDGET_VERTICAL_FADER          26  // vertical fader bar (RANGE only)
#define WIDGET_TOUCH_WHEEL             27  // touch wheel (RANGE only)
#define WIDGET_HEX_SPINNER             28  // base 16 spinner (RANGE only)
#define WIDGET_ABSOLUTE_POSITIONER     29  // absolute x,y positioner
#define WIDGET_CROSSHAIR               30  // joystick-like x,y positioner
#define WIDGET_JOY_STICK               34  // joystick x,y positioner

// widget hints for integer types with CHOICE_CONSTRAINT
#define WIDGET_COMBO_BOX                7  // combo box - usually the default
#define WIDGET_CHECKBOX                 8  // two choices
#define WIDGET_RADIO_HORIZONTAL         9  // radio buttons
#define WIDGET_RADIO_VERTICAL          10  // radio buttons
#define WIDGET_BUTTON_PROMPT           11  // single choice
#define WIDGET_BUTTON_NO_PROMPT        12  // single choice
#define WIDGET_BUTTON_TOGGLE           13  // two choices
#define WIDGET_FILE_DOWNLOAD           18  // external object OID/filename pairs
#define WIDGET_RADIO_TOGGLE_BUTTONS    22  // display a toggle button for choices
#define WIDGET_TREE                    31  // display a tree with choices
#define WIDGET_TREE_POPUP              32  // display a tree in a combo box
```

```
// widget hints for INT32_PARAM
#define WIDGET_IP_ADDRESS               14  // nnn.nnn.nnn.nnn
#define WIDGET_COLOR_CHOOSER            23  // argb color chooser
#define WIDGET_COLOR_CHOOSER_POPUP      33  // argb color chooser in popup

// widget hints for integer arrays
#define WIDGET_ARRAY_HEADER_VERTICAL   15  // array layout specification
#define WIDGET_ARRAY_HEADER_HORIZONTAL 16  // array layout specification

// widget hints for STRING_PARAM
#define WIDGET_TEXT_ENTRY               3  // normal text entry field
#define WIDGET_PASSWORD                 4  // uses password entry field
#define WIDGET_TITLE_LINE               5  // layout hint - read only
#define WIDGET_LINE_ONLY                6  // layout hint - read only
#define WIDGET_TITLE_ONLY               7  // layout hint - read only
#define WIDGET_PAGE_TAB                 8  // layout hint - read only
#define WIDGET_LICENSE                  9  // RossKeys license adapter
#define WIDGET_TITLE_HEADER            10  // layout hint - read only
#define WIDGET_COMBO_ENTRY             11  // combo box plus entry field
#define WIDGET_ICON_DISPLAY            12  // icon plus text display
#define WIDGET_RICH_LABEL              13  // multi-line display (html format)
#define WIDGET_MULTILINE_TEXT_ENTRY    14  // multi-line text entry (non-html)


// widget hints for STRING_PARAM (used with special OID 255.1)
#define WIDGET_NAME_OVERRIDE_APPEND     0
#define WIDGET_NAME_OVERRIDE_REPLACE    1

// deprecated names - here for backward compatibility
#define WIDGET_NONE        WIDGET_DEFAULT
#define WIDGET_COMBO       WIDGET_COMBO_BOX
#define WIDGET_RADIO       WIDGET_RADIO_HORIZONTAL
#define WIDGET_HSLIDER     WIDGET_SLIDER_HORIZONTAL
#define WIDGET_VSLIDER     WIDGET_SLIDER_VERTICAL
```

Appendix B: Reserved Object IDs

Reserved OIDs

Parameter OIDs in the set 0xFF00 to 0xFFFF are reserved for future protocol messages. Apart from these, there are several other OIDs that have special significance in DashBoard.

Name	OID	Type	Constraint	Function
SUPPLIER_NAME*	0x0102	String	N/A	Name of the card manufacturer or OEM supplier (i.e. who customer should call for support). Reported as a generic card parameter by SNMP.
PRODUCT_NAME**	0x0105	String (32-bytes max)	N/A	The product name used to identify the card in DashBoard. This name should not change. For display purposes, an alternate name can be provided via OID 0xFF01. Reported as a generic card parameter by SNMP.
SERIAL_NUMBER	0x0106	String	N/A	Unique serial number.
SOFTWARE_REV*+	0x010B	String (20-bytes max)	N/A	This value is used by a card to report information about its software load. The value should be meaningful to the people supporting the card. Reported as a generic card parameter by SNMP.
FPGA_REV+	0x010C	String	N/A	This value is used by DashBoard compare software versions when uploading the Main Board FPGA Type (upload type 1).
OPTION_SOFTWARE_REV+	0x010D	String	N/A	This value is used by DashBoard compare software versions when uploading the Option Board Software Type (upload type 2).
OPTION_FPGA_REV+	0x010E	String	N/A	This value is used by DashBoard compare software versions when uploading the Option Board FPGA Type (upload type 3).
SMPTE_STATUS	0x0201	Int16	N/A	Card status to be reported via frame fault LEDs. Value of 0 indicates no error. Non-zero values indicate error state.
CURRENT_MILLIS	0x0205	Int16	N/A	Current consumption in milliamps at 12 V. This may be used by the fan controller to adjust fan speed for high-current cards.
EDIT_PERMISSION	0x0601	Int16	Choice	Tells DashBoard that the card is editable. If this OID is used, parameters on the card will be editable only if the parameter value is 0. If the parameter value is non-zero, the card will display as read-only.
FRAME_POWER_CAPABILITY	0xFE0F			This OID is broadcast regularly to every card in the frame. The value of the parameter is the power available to each slot a card occupies. This value is calculated using the power

Name	OID	Type	Constraint	Function
				rating of the power supplies installed in the frame (if the power supplies are different, the lowest rating is used), minus some overhead for the frame and frame controller card, divided by the number of slots in the frame. [(Power supply rating - overhead) / Number of slots in frame.] A card may consume the power of multiple slots, if the card occupies multiple slots. For example, a card occupying two slots may use two times the parameter value.
NAME_OVERRIDE	0xFF01	String	N/A	With a widget hint of 0, the value in this String will be appended to the device name (0x0105) when displayed in the DashBoard tree and tabs. With a widget hint of 1, the value in this String will be displayed instead of the value in 0x0105 in the DashBoard tree and tabs.
CONNECT_VERIFY	0xFF03	Mixed	N/A	This parameter is used for communicating DashBoard's connection handshake and response.
UPLOAD_URL	0xFF02	String	N/A	Alternate file upload target. This overrides the behavior of the DashBoard upload button. If this value is "disable", DashBoard will disable the upload button on the device page. If this value is a valid URL, DashBoard will upload files to this location via HTTP POST.
FRAME_ID	0xFF04	String	N/A	Reserved for use by an openGear frame's Network Interface Card. If this parameter is provided, its value MUST match the unique ID provided by SLP and manual SLP attribute queries. If it does not, DashBoard will close its connection to the frame.
BACKWARDS_COMPATIBILE	0xFF05	String (20-bytes max)	N/A	Specifies the lowest software version to maintain OID-compatibility with this software version. If this OID is not supplied, the lowest software version is assumed to be the version specified in the SOFTWARE_REV OID (0x010B). The card guarantees that all software versions bounded by the version numbers specified between 0xFF05 and 0x010B can be restored using the same stored set of parameter values.
RESTORE_SET_DELAY	0xFF06	Int16	N/A	Specifies the delay to use between each parameter set request during a card restore. The restore set messages will not be sent any faster than the specified delay. This number must be between 0 and 1000 milliseconds. If this value is not specified, a default of 0 is used. Parameters will be restored as quickly as the card can process the

Name	OID	Type	Constraint	Function
				PARAM_SET commands. If the value is -1, DataSafe is disabled for this card. Other negative values are not valid at this time and should not be used.
RESTORE_START	0xFF07	Int16	N/A	A parameter set request with a value of 1 will be sent to this parameter before the card data is restored (the equivalent of a button press in DashBoard). If this parameter is provided, its position in the list of OIDs returned by the OGP_GET_PARAM_OIDS Response defines where the range of saved parameter values should start. No parameters whose OID was returned before this OID will be restored by DataSafe.
RESTORE_STOP	0xFF08	Int16	N/A	A parameter set request with a value of 1 will be sent to this parameter after the card data is restore is complete (the equivalent of a button press in DashBoard). If this parameter is provided, its position in the list of OIDs returned by the OGP_GET_PARAM_OIDS Response defines where the range of saved parameter values should stop. No parameters whose OID was returned after this OID will be restored by DataSafe.
DATASAFE_NAME	0xFF09	String	N/A	Alternative card name for determining DataSafe compatibility.
UPLOAD_NAME	0xFF0A	Int16	Choice	Alternative card name for file upload purposes.
DISPLAY_OPTIONS	0xFF0B	Int16 ARRAY		Each array element is used to define a different display option. Element 0 controls display of the card: 0 (Default) = Display the card in the tree view 1 = Hide the card in the tree view Element 1 controls the display of the slot name before the card name: 0 (Default) = Display the slot name (e.g. Slot 1: UDC-8225-W) 1 = Hide the slot name (e.g. UDC-8225-W) All other array elements are reserved for future use.
DEVICE_ICON	0xFF0C	Int16	N/A	Contains an external object ID for an encapsulated icon.
DEVICE_INDEX_URL	0xFF0D	String	N/A	URL for a DashBoard Connect XML Definition.
OGLML_DESCRIPTOR	0xFF0E	String	N/A	Provides an OGLML URL that describes a

Name	OID	Type	Constraint	Function
				layout to use in place of the standard configuration screen in DashBoard.
DEDICATED_CONNECTION	0xFF0F	Binary	N/A	Allows a card that has its own Ethernet port to communicate directly with DashBoard, bypassing the CAN bus and MFC card. This allows traffic offloading from the CAN bus, and also allows messages to be sent to specific DashBoards rather than all of them. When connected, DashBoard will use this connection to send all messages to the card. DashBoard will continue to receive updates from both the dedicated OGP connection and the CAN Bus connection. UTF-8 String for the hostname UINT16 for the port UINT8 for the use 0 = Do not use 1 = Connect when UI is visible
DEVICE_IP_ADDRESS	0xFF10	Int32	IP_ADDRESS	Cards that have their own Ethernet port should use this OID to report their current IPv4 address.
FAN_SPEED_REQUEST	0xFF11	Int16	N/A	Used by cards in OG3-FR high power frame to request additional fan cooling. Card must send OGP_REPORT_PARAM for this OID periodically (not to exceed once per minute). Value of the parameter varies depending on the cooling capabilities of the frame.
OCCUPIED_SLOTS	0xFF12	Int16	N/A	Report the number of slots this card occupies. Value consists of two 8-bit fields, representing the number of additional slots to the left and right. Value = (left << 8) | (right)
UPLOAD_FILE_EXTENSIONS	0xFF13	String Array	N/A	Extensions of file types allowed to be sent to the device. Arrau elements have the format: “[Description]<ext:[extension without dot]>”
RESERVED	0xFF14 to 0xFFFF	…	…	Reserved for future use

** Required by DashBoard and SNMP.

* Required for SNMP.

+ Version numbers are important for software uploads and DataSafe. Please review section 5-9 for recommended version number encodings.

SMPTE_STATUS, CURRENT_MILLIS, and EDIT_PERMISSION are optional, but to avoid misinterpretation, these OIDs should not be used for other parameters.

Reserved MFC and DashBoard Connect (slot 0) OIDs

Parameter OIDs in the range 0xFE00 to 0xFEFF have special significance for the MFC network

controller (Slot 0) device. These also apply to any *DashBoard Connect* devices reporting on slot 0.

Name	OID	Type	Constraint	Function
DOOR_STATE	0x0709	Int16	N/A	Broadcast by the MFC every 10 seconds to indicate door status. 1= closed and 2= open **Deprecated** field, see FAN_DOOR_STATUS on page 293.
SLOT_NAMES	0x803	Int16_Array	N/A	This array has one element for each slot in the frame. Each element's value is the OID of a String parameter whose value should be used as the name for the device in the given slot.
SLOT_DATA_SAFE	0x802	Int16_Array	N/A	This array has one element for each slot in the frame. 0 = DataSafe is enabled for the slot [Element #] Default = DataSafe is disabled for slot [Element #] by the frame
URM_STATE	0xFE01	Int16	N/A	States whether the frame requires a User Rights Management (URM) -Enabled DashBoard (or a master password) is required to connect to DashBoard. 0 (Default) = URM is not supported by the frame 1 = URM is disabled/not required 2 = URM is enabled/required
MASTER_PASSWORD	0xFE02	String (20-bytes max)	N/A	This is the value of the master password required by DashBoard users to connect when the User Rights Management server is not available and the URM State is "Enabled"
APPLY_BUTTON	0xFE03	Int16	Choice	The button DashBoard must press to apply changes to the master password or URM state parameters.
CANCEL_BUTTON	0xFE04	Int16	Choice	The button DashBoard can press to cancel any changes to the master password or URM state parameters. After the apply button has been pressed, this button does nothing.
DEVICE_CATEGORY	0xFE05	String	N/A	Default: "openGear Devices" Controls how items are grouped in User Rights Management and in the DashBoard tree view. Items sharing the same category are kept together.
FRAME_ICON	0xFE06	Int16	N/A	Contains an external object ID for an encapsulated icon.
CONFIG_SLOT	0xFE07	Int16	N/A	Default: 0 The slot # for the device to open when the frame is 'opened' for configuration.
CONFIG_URL	0xFE08	String	N/A	Default: [none] If defined and non-empty, the URL of a web page to open when the frame is 'opened' for configuration.
INDEX_URL	0xFE09	String	N/A	URL for a DashBoard Connect XML Definition.
MASTER_PASSWORD_SAVE	0xFE0A	String	N/A	Same as 0xFE02 above, but used for internal storage on the MFC controller.

Name	OID	Type	Constraint	Function
FAN_DOOR_STATUS	0xFE0B	Int16	N/A	Broadcast by the MFC every 10 seconds to indicate door status. 1= closed 2= open This replaces legacy OID 0x0709.
FAN_AMBIENT_TEMP	0xFE0C	Int16	N/A	Broadcast by the MFC every 10 seconds to report the ambient temperature of inlet air. 0 = fan door is open Otherwise temperature in degrees Celsius.
FAN_SPEED_REPORT	0xFE0D	Int16	N/A	Broadcast by the MFC every 10 seconds to report current door fan speed. 0 = minimum speed (or fan door open) Higher values indicate increasing speed. Max value depends on DFR frame type.
RESERVED	0xFE0E – 0xFEFF	…	…	Reserved for future use