FACILITY CONTROL SYSTEM

CustomPanel Development Guide

OGLML and ogScript
Version 9.4
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-007-9.4
- Release Date: April 11, 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
# Company Address

<table>
<thead>
<tr>
<th>Ross Video Limited</th>
<th>Ross Video Incorporated</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 John Street</td>
<td>P.O. Box 880</td>
</tr>
<tr>
<td>Iroquois, Ontario, K0E 1K0</td>
<td>Ogdensburg, New York</td>
</tr>
<tr>
<td>Canada</td>
<td>USA 13669-0880</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>General Business Office: (1) 613 • 652 • 4886</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fax: (1) 613 • 652 • 4425</td>
</tr>
<tr>
<td>Technical Support: (1) 613 • 652 • 4886</td>
</tr>
<tr>
<td>After Hours Emergency: (1) 613 • 349 • 0006</td>
</tr>
<tr>
<td>E-mail (Technical Support): <a href="mailto:techsupport@rossvideo.com">techsupport@rossvideo.com</a></td>
</tr>
<tr>
<td>E-mail (General Information): <a href="mailto:solutions@rossvideo.com">solutions@rossvideo.com</a></td>
</tr>
<tr>
<td>Website: <a href="http://www.rossvideo.com">http://www.rossvideo.com</a></td>
</tr>
</tbody>
</table>
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 ........................................................................................................................................................... 135

collection (device context) ....................................................................................................................... 136

subscription ........................................................................................................................................... 137

meta ....................................................................................................................................................... 138

widgets ................................................................................................................................................... 139
### ogScript Reference

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>About ogScript</td>
<td>191</td>
</tr>
<tr>
<td>JavaScript</td>
<td>191</td>
</tr>
<tr>
<td>Commonly Used Functions</td>
<td>192</td>
</tr>
<tr>
<td>Functions Set in the User Interface</td>
<td>192</td>
</tr>
<tr>
<td>multiSetScriptable Object</td>
<td>192</td>
</tr>
<tr>
<td>nkScript Object</td>
<td>192</td>
</tr>
<tr>
<td>createFileInput</td>
<td>201</td>
</tr>
</tbody>
</table>

### Device Resource Declarations

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resource XML File</td>
<td>160</td>
</tr>
<tr>
<td>config</td>
<td>164</td>
</tr>
<tr>
<td>constraint</td>
<td>166</td>
</tr>
<tr>
<td>card</td>
<td>167</td>
</tr>
<tr>
<td>frame</td>
<td>168</td>
</tr>
<tr>
<td>menu</td>
<td>168</td>
</tr>
<tr>
<td>menugroup</td>
<td>169</td>
</tr>
<tr>
<td>statusmenu</td>
<td>170</td>
</tr>
<tr>
<td>configmenu</td>
<td>171</td>
</tr>
<tr>
<td>params</td>
<td>172</td>
</tr>
<tr>
<td>param</td>
<td>172</td>
</tr>
<tr>
<td>param (struct)</td>
<td>174</td>
</tr>
</tbody>
</table>

### Device Resource Tags

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>menugroup</td>
<td>177</td>
</tr>
<tr>
<td>menu</td>
<td>177</td>
</tr>
<tr>
<td>param</td>
<td>178</td>
</tr>
<tr>
<td>constraint</td>
<td>180</td>
</tr>
<tr>
<td>buttonbar</td>
<td>180</td>
</tr>
<tr>
<td>editor</td>
<td>181</td>
</tr>
<tr>
<td>summary</td>
<td>182</td>
</tr>
<tr>
<td>statuscombo</td>
<td>182</td>
</tr>
</tbody>
</table>

### Macro Expansion

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>%frame%</td>
<td>184</td>
</tr>
<tr>
<td>%device%</td>
<td>185</td>
</tr>
<tr>
<td>%slot%</td>
<td>185</td>
</tr>
<tr>
<td>%value%</td>
<td>186</td>
</tr>
<tr>
<td>%widget%</td>
<td>187</td>
</tr>
<tr>
<td>%const%</td>
<td>187</td>
</tr>
<tr>
<td>%baseoid%</td>
<td>188</td>
</tr>
<tr>
<td>%fully-qualified-id%</td>
<td>188</td>
</tr>
<tr>
<td>%panel-path%</td>
<td>188</td>
</tr>
<tr>
<td>%app-path%</td>
<td>189</td>
</tr>
<tr>
<td>%id%</td>
<td>189</td>
</tr>
<tr>
<td>%eval[ogscript]%</td>
<td>189</td>
</tr>
</tbody>
</table>

### ogScript Reference

About ogScript ....................................................................................................................... 191
JavaScript ................................................................................................................. 191
Commonly Used Functions ........................................................................................ 184
Device Resource Tags ............................................................................................... 176
Device Resource Declarations .................................................................................. 160
Macro Expansion ........................................................................................................... 184
ogScript Reference ...................................................................................................... 191

CreateFileInput ............................................................................................................. 201
createListener .................................................................................................................. 201
createVDCPSender ........................................................................................................ 202
ftp .................................................................................................................................. 203
ftpGet ............................................................................................................................. 203
ftpListFiles ..................................................................................................................... 204
getApplicationPath ........................................................................................................ 205
getAsyncExecById ......................................................................................................... 205
getBrowserById .............................................................................................................. 206
cancelTimer ..................................................................................................................... 225
copyByteArray .............................................................................................................. 226
createByteArray ........................................................................................................... 226
createFileInput ............................................................................................................... 227
createFileOutput .......................................................................................................... 227
createMessageBuilder ................................................................................................. 228
createMessageParser ..................................................................................................... 228
debug .............................................................................................................................. 229
deployGPI ....................................................................................................................... 229
getAllById ...................................................................................................................... 230
getAttribute .................................................................................................................. 230
getBuild .......................................................................................................................... 231
getComponentsById ...................................................................................................... 231
gerentCommand ............................................................................................................. 231
getIncludeById ............................................................................................................... 232
gerentListener ............................................................................................................... 232
getModificationDate ................................................................................................... 233
getObject ........................................................................................................................ 233
getPosition ..................................................................................................................... 234
gerentPrivateString ................................................................................................. 235
getScopedAttribute ...................................................................................................... 236
getSize .......................................................................................................................... 236
getString ......................................................................................................................... 237
getSystem ....................................................................................................................... 237
getSystemObject .......................................................................................................... 238
getTimerManager ......................................................................................................... 238
hide ............................................................................................................................... 241
installTimer .................................................................................................................... 242
isTimerRunning ............................................................................................................ 243
parseXML ......................................................................................................................... 243
Appendices

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 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.
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 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 device’s 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.
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.
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:

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

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:

<table>
<thead>
<tr>
<th>Access</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCESS_READWRITE</td>
<td>0x01</td>
<td>Parameter may be modified by the control client</td>
</tr>
<tr>
<td>ACCESS_READONLY</td>
<td>0x00</td>
<td>Parameter is read-only, and may not be set by the client</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Constraint Name</th>
<th>ctype</th>
<th>Param Types</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL_CONSTRAINT</td>
<td>0</td>
<td>All</td>
<td>Parameter is unconstrained.</td>
</tr>
<tr>
<td>RANGE_CONSTRAINT</td>
<td>1</td>
<td>INT16_PARAM, INT32_PARAM, INT16_ARRAY, INT32_ARRAY, FLOAT_PARAM, FLOAT_ARRAY</td>
<td>Parameter is bounded by a min-max range. Display min-max range may be different from the value range.</td>
</tr>
<tr>
<td>CHOICE_CONSTRAINT</td>
<td>2</td>
<td>INT16_PARAM, INT16_ARRAY</td>
<td>Parameter must be selected from a set (enumeration) of name-value pairs (up to 255 choices)</td>
</tr>
<tr>
<td>EXTENDED_CHOICE</td>
<td>3</td>
<td>INT16_PARAM, INT16_ARRAY</td>
<td>Parameter must be selected from a set (enumeration) of name-value pairs (more than 255 choices)</td>
</tr>
<tr>
<td>STRING_CHOICE</td>
<td>4</td>
<td>STRING_PARAM, STRING_ARRAY</td>
<td>Provides a set of available choices. Parameter may be selected from this set, but arbitrary values are also permitted.</td>
</tr>
<tr>
<td>RANGE_STEP_CONSTRAINT</td>
<td>5</td>
<td>INT16_PARAM, INT32_PARAM, INT16_ARRAY, INT32_ARRAY, FLOAT_PARAM, FLOAT_ARRAY</td>
<td>Parameter is bounded by a min-max range. Step size indicates the amount to increment/decrement the value each time it is changed.</td>
</tr>
<tr>
<td>ALARM_TABLE</td>
<td>10</td>
<td>INT16_PARAM, INT32_PARAM, INT16_ARRAY</td>
<td>Each bit in the parameter is a status flag, so param can display 16 or 32 concurrent named error conditions.</td>
</tr>
<tr>
<td>Constraint Name</td>
<td>ctype</td>
<td>Param Types</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>EXTERNAL_CONSTRAINT</td>
<td>11</td>
<td>All</td>
<td>Indicates that the constraint is encoded in an external object, rather than encoded within the descriptor.</td>
</tr>
</tbody>
</table>

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_CONSTRAINT 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_CONSTRAINT 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:

![Menu Layout](image)

*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:

<table>
<thead>
<tr>
<th>Widget Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET_DEFAULT</td>
<td>0</td>
<td>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).</td>
</tr>
<tr>
<td>WIDGET_TEXT_DISPLAY</td>
<td>1</td>
<td>shows a read-only version of the parameter value (uses same widget that is shown when WIDGET_DEFAULT parameter is set to read-only).</td>
</tr>
<tr>
<td>WIDGET_HIDDEN</td>
<td>2</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_LABEL</td>
<td>100</td>
<td>Displays the value of the parameter as a read-only label</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Widget Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET_TITLE_LINE</td>
<td>5</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_LINE_ONLY</td>
<td>6</td>
<td>displays a line across the content area of the menu page with no label on the left.</td>
</tr>
<tr>
<td>WIDGET_TITLE_ONLY</td>
<td>7</td>
<td>displays the value of the String parameter as a label with empty space in the content area of the menu page.</td>
</tr>
<tr>
<td>WIDGET_PAGE_TAB</td>
<td>8</td>
<td>creates a 3rd-level tab within the menu page. The value of the parameter is used as the tab label.</td>
</tr>
<tr>
<td>WIDGET_TITLE_HEADER</td>
<td>10</td>
<td>displays a title over the content area of the menu with the value of the parameter used as the header text.</td>
</tr>
</tbody>
</table>
**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.](image)

**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.](image)

**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](image)

**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.](image)

**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.

![Choice Selectors](image)

*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:

```
```

*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:

```
```

*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]
```

*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:

<table>
<thead>
<tr>
<th>Widget Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET_ARRAY_HEADER_VERTICAL</td>
<td>15</td>
<td>indicates that the associated array parameter and all subsequent parameters should be displayed in a vertical layout</td>
</tr>
<tr>
<td>WIDGET_ARRAY_HEADER_HORIZONTAL</td>
<td>16</td>
<td>indicates that the associated array parameter and all subsequent parameters should be displayed in a horizontal layout</td>
</tr>
</tbody>
</table>

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.](image-url)
**WIDGET_ARRAY_HEADER_HORIZONTAL (16)**

The **WIDGET_ARRAY_HEADER_HORIZONTAL** is used to create a header over a horizontal array. It will also 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.](image)

**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.

<table>
<thead>
<tr>
<th>Widget Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET_COMBO_BOX</td>
<td>7</td>
<td>Displays a dropdown list of selectable options. This is the default widget used for any choice parameter with more than 1 choice provided.</td>
</tr>
<tr>
<td>WIDGET_CHECKBOX</td>
<td>8</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_RADIO_HORIZONTAL</td>
<td>9</td>
<td>Displays a radio button for each integer value option. The radio buttons are placed beside each other horizontally on the page.</td>
</tr>
<tr>
<td>WIDGET_RADIO_VERTICAL</td>
<td>10</td>
<td>Displays a radio button for each integer value option. The radio buttons are placed in a vertical column.</td>
</tr>
<tr>
<td>WIDGET_BUTTON_PROMPT</td>
<td>11</td>
<td>Provides a button with confirmation prompt. Whenever the button is pressed and confirmed, the parameter value is sent to the device.</td>
</tr>
<tr>
<td>WIDGET_BUTTON_NO_PROMPT</td>
<td>12</td>
<td>Provides a button without confirmation prompt. Whenever the button is pressed, the parameter value is sent to the device.</td>
</tr>
<tr>
<td>WIDGET_BUTTON_TOGGLE</td>
<td>13</td>
<td>Displays a toggle button. This hint applies only to parameters with exactly 2 choices. The first choice is true, second choice is false.</td>
</tr>
<tr>
<td>Widget Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WIDGET_FILE_DOWNLOAD</td>
<td>18</td>
<td>Displays a file download widget. This hint requires an external object with an OID matching the value of the parameter.</td>
</tr>
<tr>
<td>WIDGET_MENU_POPUP</td>
<td>20</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_RADIO_TOGGLE_BUTTONS</td>
<td>22</td>
<td>Displays a toggle button for each integer value option. The toggle buttons are placed beside each other horizontally on the page.</td>
</tr>
<tr>
<td>WIDGET_TREE</td>
<td>31</td>
<td>Displays a tree control. Tree elements are defined by the elements of the choice constraint. The tree hierarchy is defined by &quot;-&quot; characters at the beginning of the choice. See detailed description below for more information.</td>
</tr>
<tr>
<td>WIDGET_TREE_POPUP</td>
<td>32</td>
<td>Displays the tree (same definition as WIDGET_TREE) in a combo box control. See detailed description below for more information.</td>
</tr>
</tbody>
</table>

**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](image)

**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](image)
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.

![Radio Choice](image)

*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.

![Radio Choice](image)

*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.

![Factory Defaults](image)

*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](image)

**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 as 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](image)

### 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).
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.
**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](image)

**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](image)

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

1. Element 1
2. +Element 1 - 1
3. +Element 1 - 2
4. +Element 1 - 2 - 1
5. +Element 1 - 2 - 2
6. +Element 1 - 2 - 3
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](image)

**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.

<table>
<thead>
<tr>
<th>Widget Name</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIDGET_SLIDER_HORIZONTAL</td>
<td>3</td>
<td>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</td>
</tr>
<tr>
<td>WIDGET_SLIDER_VERTICAL</td>
<td>4</td>
<td>Displays a vertical slider control. This is the default control for range-bounded integer and floating point array parameters</td>
</tr>
<tr>
<td>WIDGET_SPINNER</td>
<td>5</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_TEXTBOX</td>
<td>6</td>
<td>Displays a numeric entry field. This is the default for unbounded FLOAT and INT32 parameters.</td>
</tr>
<tr>
<td>WIDGET_IP_ADDRESS</td>
<td>14</td>
<td>Displays an IP Address entry field. Only works with unconstrained INT32 parameters.</td>
</tr>
<tr>
<td>WIDGET_PROGRESS_BAR</td>
<td>17</td>
<td>Displays a read-only progress bar control.</td>
</tr>
<tr>
<td>WIDGET_AUDIO_METER</td>
<td>19</td>
<td>Displays a read-only audio level meter control with green, yellow, and red markers.</td>
</tr>
<tr>
<td>WIDGET_TIMER</td>
<td>21</td>
<td>Displays a label that counts down from the parameter value to 0 when double-clicked.</td>
</tr>
<tr>
<td>WIDGET_COLOR_CHOOSER</td>
<td>23</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_SLIDER_HORIZONTAL_NO_LABEL</td>
<td>24</td>
<td>Displays a horizontal slider control with no label</td>
</tr>
<tr>
<td>WIDGET_SLIDER_VERTICAL_NO_LABEL</td>
<td>25</td>
<td>Displays a vertical slider control with no label</td>
</tr>
<tr>
<td>WIDGET_VERTICAL_FADER</td>
<td>26</td>
<td>Displays a vertical slider that looks like a fader bar</td>
</tr>
<tr>
<td>Widget Name</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WIDGET_TOUCH_WHEEL</td>
<td>27</td>
<td>Displays a touch wheel control</td>
</tr>
<tr>
<td>WIDGET_HEX_SPINNER</td>
<td>28</td>
<td>Displays a spinner (entry field plus up/down arrows). Display the value in Base 16.</td>
</tr>
<tr>
<td>WIDGET_ABSOLUTE_POSITIONER</td>
<td>29</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_ABSOLUTE_CROSSHAIR</td>
<td>30</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_JOY_STICK</td>
<td>34</td>
<td>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.</td>
</tr>
<tr>
<td>WIDGET_COLOR_CHOOSER_POPUP</td>
<td>33</td>
<td>Display a combo box control showing the ‘current’ colour. On click, show the colour chooser. If “Live” is toggled 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.</td>
</tr>
<tr>
<td>WIDGET_GRAPH</td>
<td>256</td>
<td>Displays a plot graph of a parameter’s value over time.</td>
</tr>
<tr>
<td>WIDGET_EQ_GRAPH</td>
<td>46</td>
<td>Displays an EQ Graph that provides a visual representation of how bands effect frequencies across a given range.</td>
</tr>
</tbody>
</table>

**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](image-url)
**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](image)

**WIDGET_SLIDER_HORIZONTAL_NO_LABEL (24)**

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 30 - WIDGET_SLIDER_HORIZONTAL_NO_LABEL hint](image)

**WIDGET_SLIDER_VERTICAL_NO_LABEL (25)**

![Figure 30 - WIDGET_SLIDER_HORIZONTAL_NO_LABEL hint](image)

**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.
**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.

![WIDGET_TOUCH_WHEEL](image)

**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%.

![WIDGET_PROGRESS_BAR](image)

**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.

![WIDGET_SPINNER](image)

**Notes**

The range of the parameter: \( \text{abs}(\text{max} - \text{min}) \times \text{precision} \) 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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.keyboard</td>
<td>Integer</td>
<td>disabled</td>
<td>- Disables the soft keyboard or number pad to enter characters when using a touchscreen.</td>
</tr>
</tbody>
</table>
**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](image)

**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](image)

**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](image)

**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:
- \( \text{min}=-600, \text{max}=0, \text{step}=1 \) (count from 10:00 to 0:00 showing each second).
- \( \text{min}=0, \text{max}=600, \text{step}=1 \) (count from 0:00 to 10:00 showing each second)
- \( \text{min}=0, \text{max}=1000, \text{step}=1000 \) (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.

**WIDGET_HEX_SPINNER (28)**

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

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.

**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.

![Crosshair](image)

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.

![Joystick](image)

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.

![Color Chooser](image)
**WIDGET_COLOR_CHOOSER_POPUP(33)**

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

- If “Live” is toggled 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)**

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

![WIDGET_GRAPH graph example](image)

*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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.time</td>
<td>Integer</td>
<td></td>
<td>Sets the timescale of the plot. If set to 0, the timescale will adapt to display entire change history.</td>
</tr>
<tr>
<td>w.grid</td>
<td>String</td>
<td></td>
<td>Sets the color of the gridlines</td>
</tr>
<tr>
<td>w.plotfg</td>
<td>String</td>
<td></td>
<td>Sets the color of the plot foreground</td>
</tr>
<tr>
<td>w.plotbg</td>
<td>String</td>
<td></td>
<td>Sets the color of the plot background</td>
</tr>
<tr>
<td>w.hidelegend</td>
<td>Boolean</td>
<td>true</td>
<td>true – Legend is not shown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
<td>false – Legend is shown</td>
</tr>
<tr>
<td>w.hidex</td>
<td>Boolean</td>
<td>true</td>
<td>true – X-axis scale is not shown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
<td>false – X-axis scale is shown</td>
</tr>
<tr>
<td>w.hidey</td>
<td>Boolean</td>
<td>true</td>
<td>true – Y-axis scale is not shown</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
<td>false – Y-axis scale is shown</td>
</tr>
<tr>
<td>w.autoadvance</td>
<td>Booleloan</td>
<td>true</td>
<td>graph will auto-update every 1 second</td>
</tr>
<tr>
<td></td>
<td></td>
<td>false</td>
<td>graph will only update upon parameter change.</td>
</tr>
</tbody>
</table>
**WIDGET_EQ_GRAPH (46)**

The EQ graph widget provides 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.

![EQ Graph Example](image)

*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**

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.linecolor</td>
<td># [RGB Value]</td>
<td></td>
<td>Sets the color of the graph point to point line.</td>
</tr>
<tr>
<td>w.filters</td>
<td>[ String Array]</td>
<td></td>
<td>Sets the filter for each band, where the possible values are lowshelf, peak or highshelf. One filter per point.</td>
</tr>
<tr>
<td>w.pointnames</td>
<td>[String Array]</td>
<td></td>
<td>Sets the name on each graph point, for example:</td>
</tr>
<tr>
<td>w.colorselected</td>
<td># [RGB Value Array]</td>
<td>1, 2, 3, 4</td>
<td>Sets the color for each point when it’s selected. For example:</td>
</tr>
<tr>
<td>w.colorunselected</td>
<td># [RGB Value Array]</td>
<td></td>
<td>Sets the color for each point when it’s not selected. For example:</td>
</tr>
<tr>
<td>w.linethickness</td>
<td>[Integer]</td>
<td>#fffd66,#c27ba0,#6d9eeb,#93c47d</td>
<td></td>
</tr>
<tr>
<td>w.graphfontsize</td>
<td>[Integer]</td>
<td></td>
<td>Sets the color for each point when it’s not selected. For example:</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------</td>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>w.pointfontsize</td>
<td>[Integer]</td>
<td>#ffd966,#c27ba0,#6d9eeb,#93c47d</td>
<td>Sets the font size of the text used on the graph title, axis labels and axis entries.</td>
</tr>
<tr>
<td>w.pointwidth</td>
<td>[Integer]</td>
<td></td>
<td>Sets the thickness of the graph point to point line.</td>
</tr>
<tr>
<td>w.pointheight</td>
<td>[Integer]</td>
<td></td>
<td>Sets the line marks on the x axis. For example: 20,50,100,200,500,1000,2000,5000,10000</td>
</tr>
<tr>
<td>w.xaxis</td>
<td>[String]</td>
<td></td>
<td>Sets the font size for the point names.</td>
</tr>
<tr>
<td>w.yaxis</td>
<td>[String]</td>
<td></td>
<td>Sets the width of the points.</td>
</tr>
<tr>
<td>w.xaxisentries</td>
<td>[String Array]</td>
<td></td>
<td>Sets the x axis label.</td>
</tr>
<tr>
<td>w.yaxisentries</td>
<td>[String Array]</td>
<td></td>
<td>Sets the y axis label.</td>
</tr>
<tr>
<td>w.pointamount</td>
<td>[Integer]</td>
<td></td>
<td>Sets the y axis label.</td>
</tr>
<tr>
<td>w.frequencyshift</td>
<td>[Integer]</td>
<td></td>
<td>Sets the line marks on the x axis. For example: 20,50,100,200,500,1000,2000,5000,10000</td>
</tr>
</tbody>
</table>

**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.

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

![Network Name TestCard](image)

*Figure 47 - WIDGET_TEXT_ENTRY hint*

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.keyboard</td>
<td>Integer</td>
<td>disabled</td>
<td>Disables the soft keyboard or number pad to enter characters when using a touchscreen.</td>
</tr>
</tbody>
</table>

**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.

![Extra Menu Login Password](image)

*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](image)

**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>`](image)

**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](image)

**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.
**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.

**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.
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.

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>w.localselection</td>
<td>Boolean</td>
<td>false</td>
<td>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.</td>
</tr>
<tr>
<td>w.scrollselection</td>
<td>Boolean</td>
<td>true</td>
<td>true — auto scroll to the selected row false — do not scroll to the selected row</td>
</tr>
<tr>
<td>w.reorder</td>
<td>Boolean</td>
<td>false</td>
<td>true — allow drag to reorder values false — do not allow drag to reorder values</td>
</tr>
<tr>
<td>w.rowstyleparam</td>
<td>String</td>
<td>none</td>
<td>OID of string array parameter providing style information (background, foreground, font, font size, etc.) for each row.</td>
</tr>
<tr>
<td>w.selectionparam</td>
<td>String</td>
<td>none</td>
<td>OID of integer parameter that will be populated with the index of a selected row.</td>
</tr>
<tr>
<td>w.rowaccessparam</td>
<td>String</td>
<td>none</td>
<td>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.</td>
</tr>
<tr>
<td>w.rowheight</td>
<td>Number</td>
<td>automatic</td>
<td>Sets the row height. Specified in pixels</td>
</tr>
<tr>
<td>w.colwidth.(n)</td>
<td>Number</td>
<td>automatic</td>
<td>Sets the width of the (n)th column. First column index is 0.</td>
</tr>
<tr>
<td>w.colminwidth.(n)</td>
<td>Number</td>
<td>automatic</td>
<td>Sets the minimum width of the (n)th column. First column index is 0.</td>
</tr>
<tr>
<td>w.hscroll</td>
<td>Boolean</td>
<td>false</td>
<td>true — show horizontal scrollbar false — do not show horizontal scrollbar</td>
</tr>
<tr>
<td>w.alwaysscroll</td>
<td>Boolean</td>
<td>false</td>
<td>true — vertical scrollbar always shown false — vertical scrollbar only shown only when required</td>
</tr>
<tr>
<td>w.hgrid</td>
<td>Boolean</td>
<td>true</td>
<td>true — display horizontal grid lines false — do not display horizontal grid lines</td>
</tr>
<tr>
<td>w.vgrid</td>
<td>Boolean</td>
<td>true</td>
<td>true — display vertical grid lines false — do not display vertical grid lines</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Data Type Name</th>
<th>OGP type</th>
<th>Data Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INT16</td>
<td>2</td>
<td>2</td>
<td>16-bit signed integer (INT16)</td>
</tr>
<tr>
<td>INT32</td>
<td>4</td>
<td>4</td>
<td>32-bit signed integer (INT32)</td>
</tr>
<tr>
<td>FLOAT32</td>
<td>6</td>
<td>4</td>
<td>32-bit IEEE single-precision floating point number</td>
</tr>
<tr>
<td>STRING</td>
<td>7</td>
<td>variable</td>
<td>null-terminated UTF-8 string</td>
</tr>
<tr>
<td>INT16_ARRAY</td>
<td>12</td>
<td>2 * len</td>
<td>array of 16-bit integers</td>
</tr>
<tr>
<td>INT32_ARRAY</td>
<td>14</td>
<td>4 * len</td>
<td>array of 32-bit integers</td>
</tr>
<tr>
<td>FLOAT32_ARRAY</td>
<td>16</td>
<td>4 * len</td>
<td>array of 32-bit floats</td>
</tr>
<tr>
<td>STRING_ARRAY</td>
<td>17</td>
<td>variable</td>
<td>null-terminated UTF-8 strings</td>
</tr>
<tr>
<td>STRUCT</td>
<td>n/a</td>
<td>variable</td>
<td>User-defined data structure. (DashBoard 7.0+)</td>
</tr>
<tr>
<td>STRUCT_ARRAY</td>
<td>n/a</td>
<td>variable</td>
<td>Array of User-defined data structures. (DashBoard 7.0+)</td>
</tr>
<tr>
<td>BINARY_PARAM</td>
<td>18</td>
<td>variable</td>
<td>array of binary data of type unknown to DashBoard.</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Response</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>UINT8</td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>UINT16</td>
<td>0</td>
<td>65535</td>
</tr>
<tr>
<td>INT16</td>
<td>-32,768</td>
<td>32,767</td>
</tr>
<tr>
<td>INT32</td>
<td>-2,147,483,648</td>
<td>2,147,483,647</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>objtype</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Constraint</td>
</tr>
<tr>
<td>2</td>
<td>File</td>
</tr>
<tr>
<td>3</td>
<td>Image</td>
</tr>
<tr>
<td>4</td>
<td>OGLML or XML document</td>
</tr>
</tbody>
</table>

**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
<?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 container>
  ...
</ui container>

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](http://myhost/mydocument.xml) (include the entire document at the given URL)
- [https://10.0.100.1/document.xml#myid](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.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x00</td>
</tr>
<tr>
<td>content</td>
<td>0</td>
<td>No content is provided in this case</td>
</tr>
</tbody>
</table>

#### Descriptor provided by external object (0x01)

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x01</td>
</tr>
<tr>
<td>content</td>
<td>2 (uint16)</td>
<td>The external object ID of the object containing the OGLML Descriptor</td>
</tr>
</tbody>
</table>

#### OGLML Document provided by URL (0x02)

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x02</td>
</tr>
<tr>
<td>urllen</td>
<td>1 (uint8)</td>
<td>The length of the URL to follow including the null terminator</td>
</tr>
<tr>
<td>url</td>
<td>urllen</td>
<td>The null-terminated URL of the external object. This must begin with &quot;http://&quot; or &quot;https://&quot;. 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.</td>
</tr>
</tbody>
</table>

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

The OGLML File document immediately follows the descriptor type field.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x03</td>
</tr>
<tr>
<td>content</td>
<td>*</td>
<td>OGLML XML File Content</td>
</tr>
</tbody>
</table>

#### 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).

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x04</td>
</tr>
<tr>
<td>content</td>
<td>*</td>
<td>GZipped OGLML XML File Content</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>Field</th>
<th>length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>desctype</td>
<td>1 (uint8)</td>
<td>0x05</td>
</tr>
<tr>
<td>content</td>
<td>*</td>
<td>Deflate OGLML XML File Content</td>
</tr>
</tbody>
</table>

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>
  </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:

```xml
<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>
```
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.

```xml
<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>
  </config>
</widgetdescriptor>
```
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.

The widget also defines a custom configuration panel, which is presented within PanelBuilder’s “Edit Component” 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
<?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="0" 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>
  </widgetdescriptor>
</widgets>
```
<!-- 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>
</oglml>
<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 < 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>

<.statusText 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 tasktype="ogscript">addDigit('CLR');</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.Number', 0, '0'));
      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="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>
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 widget descriptor `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
**Important:** Unlike 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.

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. See the example below:

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.

![CUSTOMER AD 1](image)

00:06.18

TakeID: 10000
Framebuffer: 2
Layer: 2

**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**.

![Hardware Setup](image)

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.

   ![Insert into AIB Component](image1)

   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:

   ![Component Editor](image2)

   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):

![Default Widget Example](image)

**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.

![Countdown Timer Data Example](image)
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**

```xml
<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:

```xml
<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:
          ' + 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:

```xml
<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>
  </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.

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

```html
// 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:

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

Is replaced with this:

```html
<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:

```xml
<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:
    ' + pretty(this));
</api>

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

```xml
<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()`:

```javascript
// 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:

```javascript
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 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;
    }
})();
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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>containertype</td>
<td>bottom</td>
<td></td>
<td>Adjust the border and shading of the</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>----------------</td>
<td>-------------------------------------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>inset</td>
<td>etched</td>
<td></td>
<td>component. See the examples below.</td>
</tr>
<tr>
<td></td>
<td>raised</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>lowered</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tabpage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scroll</td>
<td>should not be used with “browser” tag. Nesting within another “scrolls” element is not recommended</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>true</td>
<td>Scrolling for the “menu” tag will always be true.</td>
<td>Provides both horizontal and vertical scrollbars.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td>Provides no scrollbars.</td>
</tr>
<tr>
<td></td>
<td>horizontal</td>
<td>Provides only horizontal scrolling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>vertical</td>
<td>Provides only vertical scrolling</td>
<td></td>
</tr>
<tr>
<td></td>
<td>always</td>
<td>Forces horizontal and vertical scrollbars to always be visible.</td>
<td></td>
</tr>
<tr>
<td>contexttype</td>
<td>opengear</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>objectid</td>
<td>node-id</td>
<td>node-id of the source of parameters within container. The objectid is passed onto child elements and containers.</td>
<td></td>
</tr>
<tr>
<td>objecttype</td>
<td></td>
<td>Type of device when communicating with an openGear or DashBoard Connect device.</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>String</td>
<td>Must be unique within all OGLML files displayed by a device.</td>
<td>Used to uniquely identify/reference an element in the OGLML file.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>IDs must only use A-Z, a-z, 0-9, “-“, and “_” characters.</td>
<td></td>
</tr>
<tr>
<td>subscriptions</td>
<td>Boolean</td>
<td>Required for devices that support subscriptions protocol.</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>This attribute can only be added to Layout/Container tags.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>If you are using more than one</td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------</td>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>width</td>
<td>Positive integer</td>
<td>Required for browser tags</td>
<td>Specifies the preferred width (in pixels) for a component. May be ignored by DashBoard depending on the component.</td>
</tr>
<tr>
<td>height</td>
<td>Positive integer</td>
<td>Required for browser tags</td>
<td>Specifies the preferred height (in pixels) for a component. May be ignored by DashBoard depending on the component.</td>
</tr>
<tr>
<td>style</td>
<td>style hints</td>
<td>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.</td>
<td>openGear Style Hints are used to specify the background color, foreground color, icon, and border for certain components. Refer to Style Hints.</td>
</tr>
</tbody>
</table>
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**

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

**Example of a Subscriptions Panel with Multiple Elements from a Device**

```xml
<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**

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

```xml
</context>
```
Example of a Subscriptions Panel with Two Device Contexts

```xml
<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:

```html
<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 (`&lt; &gt;`). Specifying hints within the constraint value allows different styles to be applied to each choice.

```html
<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:

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

<table>
<thead>
<tr>
<th>Style Hint</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>style:style-id</td>
<td>String</td>
<td>The style with the provided ID must be defined in an OGLML document at a higher scope than where it is referenced.</td>
<td>Apply the style hints of the style defined in a different set of style hints. See style tag documentation for more information.</td>
</tr>
</tbody>
</table>
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).

```xml
<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.

<table>
<thead>
<tr>
<th>Style Hint</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#color-value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td></td>
<td>Set the background color of the component.</td>
</tr>
<tr>
<td>bg#color-value</td>
<td></td>
<td></td>
<td>Colors may be specified as RGB, RGBA or one of the &lt;pre-defined color constants. R, G, B and A are specified as 2-digit hex values.</td>
</tr>
<tr>
<td>fg#color-value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td></td>
<td>Set the foreground color of the component.</td>
</tr>
<tr>
<td>bdr#color-value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td></td>
<td>Create a line border around the component with the specified color.</td>
</tr>
<tr>
<td>o#color-value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td></td>
<td>Create an outline around the text within a component with the specified color.</td>
</tr>
<tr>
<td>grid#color-value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td>Applies to table container only.</td>
<td>Specifies the color of the gridlines for a table container.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Colors may be specified as RGB, RGBA or one of the &lt;pre-defined color constants. R, G, B and A are specified as 2-digit hex values.</td>
</tr>
</tbody>
</table>
Example
The following style tag creates a label using the predefined background #panelbg and the foreground (text) in orange.

```html
<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:

![Predefined color constants](image)

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:
<table>
<thead>
<tr>
<th>Color Constant</th>
<th>Color Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#panelbg</td>
<td>[ ]</td>
<td>Panel background color.</td>
</tr>
<tr>
<td>#panelfg</td>
<td>[ ]</td>
<td>Panel foreground color for a basic control or button.</td>
</tr>
<tr>
<td>#selectbg</td>
<td>[ ]</td>
<td>Background color for a toggle button that is selected.</td>
</tr>
<tr>
<td>#selectedmuted</td>
<td>[ ]</td>
<td>Color for a mute button that is selected.</td>
</tr>
<tr>
<td>#buttonbg</td>
<td>[ ]</td>
<td>Background color for a button.</td>
</tr>
<tr>
<td>#tableheader</td>
<td>[ ]</td>
<td>Color for a table header.</td>
</tr>
<tr>
<td>#tablezebra</td>
<td>[ ]</td>
<td>Secondary color for table rows.</td>
</tr>
<tr>
<td>#readonlyborder</td>
<td>[ ]</td>
<td>Color for a read-only border.</td>
</tr>
<tr>
<td>#listbg</td>
<td>[ ]</td>
<td>Background color for a list.</td>
</tr>
<tr>
<td>#tabbg</td>
<td>[ ]</td>
<td>Background color for a selected tab.</td>
</tr>
<tr>
<td>#textbg</td>
<td>[ ]</td>
<td>Color for background for text.</td>
</tr>
<tr>
<td>#lightdivider</td>
<td>[ ]</td>
<td>Color of a light divider.</td>
</tr>
<tr>
<td>#darkdivider</td>
<td>[ ]</td>
<td>Color of a dark divider.</td>
</tr>
<tr>
<td>#modaloverlay</td>
<td>[ ]</td>
<td>Color for a modal overlay.</td>
</tr>
<tr>
<td>#timerfg</td>
<td>[ ]</td>
<td>Foreground color for a timer.</td>
</tr>
<tr>
<td>#red</td>
<td>[ ]</td>
<td>Red.</td>
</tr>
<tr>
<td>#orange</td>
<td>[ ]</td>
<td>Orange.</td>
</tr>
<tr>
<td>#yellow</td>
<td>[ ]</td>
<td>Yellow.</td>
</tr>
<tr>
<td>#green</td>
<td>[ ]</td>
<td>Green.</td>
</tr>
<tr>
<td>#teal</td>
<td>[ ]</td>
<td>Teal.</td>
</tr>
<tr>
<td>#blue</td>
<td>[ ]</td>
<td>Blue.</td>
</tr>
<tr>
<td>#purple</td>
<td>[ ]</td>
<td>Purple.</td>
</tr>
</tbody>
</table>
**Color Constant** | **Color Sample** | **Description**
---|---|---
`#pink` | ![Pink](image) | Pink.
`#transparent` | ![No fill](image) | 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.

<table>
<thead>
<tr>
<th><strong>Style Hint</strong></th>
<th><strong>Values</strong></th>
<th><strong>Restrictions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>bdr: <code>border-style</code></td>
<td>none</td>
<td></td>
<td>Removes the border from the component.</td>
</tr>
<tr>
<td></td>
<td>etched</td>
<td></td>
<td>Create an etched border around the component.</td>
</tr>
<tr>
<td></td>
<td>shadow</td>
<td></td>
<td>Creates a drop shadow under the component.</td>
</tr>
<tr>
<td>bdr# <code>color-value</code></td>
<td>Sets border color; see Component Color Section.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>grid# <code>color-value</code></td>
<td>Sets grid color in a table; see Component Color Section.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

The following image illustrates the border style hint:

![Border Style Examples](image)

*Figure 65 - Border style*

**Text/Font Styles**

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

<table>
<thead>
<tr>
<th><strong>Style Hint</strong></th>
<th><strong>Values</strong></th>
<th><strong>Restrictions</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
</table>
| 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.
<p>|  |  |  | See examples below. |</p>
<table>
<thead>
<tr>
<th>Style Hint</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>font:font-type</code></td>
<td><code>default</code></td>
<td></td>
<td>Set the control font to the default font, a</td>
</tr>
<tr>
<td></td>
<td><code>bold</code></td>
<td></td>
<td>bold font, or a mono-spaced font.</td>
</tr>
<tr>
<td></td>
<td><code>mono</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>txt-align:alignment</code></td>
<td><code>center</code></td>
<td></td>
<td>Controls the position of text within a</td>
</tr>
<tr>
<td></td>
<td><code>north</code></td>
<td></td>
<td>button or label control.</td>
</tr>
<tr>
<td></td>
<td><code>northeast</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>east</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>southeast</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>south</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>southwest</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>west</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><code>northwest</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>fg#color-value</code></td>
<td>Sets text foreground color; see Component Color Section.</td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>o#color-value</code></td>
<td>Sets text outline color; see Component Color Section.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

The following image illustrates the size style

![Figure 66 - size style attribute](image)

The following image illustrates the font style:

![Figure 67 - font style attribute](image)

**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.

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

**Example**

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

![Background and Hover Icon](image)

**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.

<table>
<thead>
<tr>
<th>Style Hint</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tt:tool-tip-string</td>
<td>String</td>
<td>May only be applied to label and buttons.</td>
<td>Set the tooltip of the component to the specified String. A &quot;;&quot; can be inserted into the string by inserting the escape sequence &quot;;&quot;.</td>
</tr>
</tbody>
</table>

**Example**

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

![Tooltip](image)

**Figure 69 – Tooltip Style**

### Inset Style

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

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

<table>
<thead>
<tr>
<th>Style Hint</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>b-u:image-url</td>
<td>URL String</td>
<td>Must be a fully qualified URL.</td>
<td>Set the background image of the component.</td>
</tr>
<tr>
<td>bg-fill:value</td>
<td>none</td>
<td>Do not scale the image.</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td>Stretch the image to the width/height of the control.</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>horizontal</td>
<td>Scale the image to the width of the control (maintain aspect ratio).</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>vertical</td>
<td>Scale the image to the height of the control (maintain aspect ratio).</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>fit</td>
<td>Scale the image to the largest size that will fit inside of the control (maintain aspect ratio).</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>crop</td>
<td>Scale the image to fill the control maintaining the aspect ratio. Crop the image to remove the parts that don’t fit.</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>tile</td>
<td>Tile the image (starting at the upper left) to fill the background of the control.</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
<tr>
<td></td>
<td>paint9</td>
<td>Divide the image into 9 areas (defined with Background Insets) to define fixed corners, vertically or horizontally stretched sides, and a stretched center.</td>
<td>Facilities such as buttons and radio buttons can be configured for each choice.</td>
</tr>
</tbody>
</table>

### 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

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

<table>
<thead>
<tr>
<th>Hint Modifier</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>true-style-hint</code></td>
<td>Valid style hint</td>
<td>Applies the style hint only when the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>choice is true</td>
<td></td>
</tr>
<tr>
<td><code>false-style-hint</code></td>
<td>Valid style hint</td>
<td>Applies the style hint only when the</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>choice is false</td>
<td></td>
</tr>
</tbody>
</table>

Examples

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

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

![Figure 70 - Toggle Button Style Modifier](image)

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

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

![Figure 71 – Radio Button Style Modifier](image)

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:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>abs</code></td>
<td>Allows elements to be placed in absolute positions</td>
</tr>
<tr>
<td><code>borderlayout</code></td>
<td>Creates a border layout that maintains proportions of components</td>
</tr>
<tr>
<td></td>
<td>anchored to the border edges or center when resized, and offers the</td>
</tr>
<tr>
<td></td>
<td>option to set one component to grow in relation to the other components</td>
</tr>
<tr>
<td><code>flow</code></td>
<td>Aligns elements in a horizontal row</td>
</tr>
<tr>
<td><code>pager</code></td>
<td>Creates a pager control component that is customizable using script</td>
</tr>
<tr>
<td><code>popup</code></td>
<td>Presents child elements in a popup window</td>
</tr>
<tr>
<td><code>simplegrid</code></td>
<td>Creates a grid of fixed-size rows and columns</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>split</td>
<td>Creates a draggable split screen with 2 components</td>
</tr>
<tr>
<td>tab</td>
<td>Creates a tabbed page</td>
</tr>
<tr>
<td>table</td>
<td>Creates a grid of rows and columns</td>
</tr>
</tbody>
</table>

### 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

```html
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualwidth</td>
<td>Integer</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>virtualheight</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
<tr>
<td>subscriptions</td>
<td>String</td>
<td></td>
<td>When set to subscriptions=&quot;true&quot;, 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.</td>
</tr>
</tbody>
</table>

#### Child Component Attributes

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Integer</td>
<td></td>
<td>Defines the distance between the left edge of the <code>abs</code> and the component. When combined with right it will force the component to fill the available area.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>right</td>
<td>Integer</td>
<td></td>
<td>Defines the distance between the right edge of the <code>abs</code> and the control. When combined with <code>left</code>, it will force the component to fill the available area.</td>
</tr>
<tr>
<td>top</td>
<td>Integer</td>
<td></td>
<td>Defines the distance between the top edge of the <code>abs</code> and the control. When combined with <code>bottom</code>, it will force the component to fill the available area.</td>
</tr>
<tr>
<td>bottom</td>
<td>Integer</td>
<td></td>
<td>Defines the distance between the bottom edge of the <code>abs</code> and the control. When combined with <code>top</code>, it will force the component to fill the available area.</td>
</tr>
<tr>
<td>width</td>
<td>Integer</td>
<td>Ignored if both <code>left</code> and <code>right</code> are specified.</td>
<td>Defines the width of the control. If undefined, the control's calculated preferred size will be used.</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>Ignored if both <code>top</code> and <code>bottom</code> are specified.</td>
<td>Defines the height of the control. If undefined, the control's calculated preferred size will be used.</td>
</tr>
</tbody>
</table>

**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:

\[
\text{component display width} = \text{component width} \times \left( \frac{\text{abs width}}{\text{abs virtualwidth}} \right)
\]

\[
\text{component display height} = \text{component height} \times \left( \frac{\text{abs height}}{\text{abs virtualheight}} \right)
\]

**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):

```xml
<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:

```xml
<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"/>
</abs>
```
In the following example, the 2x2 grid will be scaled to half its original size. All buttons will appear 50x50 pixels in size:

```html
<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

```xml
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
</table>
| grow      | String | In the GUI, the growth quadrant 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchor</td>
<td>String</td>
<td>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.</td>
<td>Defines the border side which the component will be anchored to.</td>
</tr>
<tr>
<td>width</td>
<td>Integer</td>
<td>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 &lt;abs/&gt; containers the width is required for the component to appear on the canvas.</td>
<td>Defines the width of the component. This is impacted by the growth attribute.</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>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 &lt;abs/&gt; containers the height is required for the component to appear on the canvas.</td>
<td>Defines the height of the component. This is impacted by the growth attribute.</td>
</tr>
</tbody>
</table>

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:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>anchor</td>
<td>center</td>
<td>east west</td>
<td>Defines the alignment of the controls</td>
</tr>
</tbody>
</table>

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.

```xml
<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. Popups 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**

```xml
<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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>The name to display on the button to trigger the popup.</td>
</tr>
<tr>
<td>group</td>
<td>String</td>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>Integer</td>
<td>Required</td>
<td>Width of the container inside the popup</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>Required</td>
<td>Height of the container inside the popup</td>
</tr>
</tbody>
</table>

**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.

```xml
<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](image)

**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.

![Page 1](image)

**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:
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>virtualwidth</td>
<td>Integer</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>virtualheight</td>
<td>Integer</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

The following example creates a horizontal pager control.

```xml
<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="asdasdfasdfs" style="txt-align:west" top="58" width="161"/>
      </abs>
      <abs id="page-2">
        <button buttontype="push" height="65" left="185" name="asdfsdfasdfs" top="50" width="182"/>
      </abs>
      <abs id="page-3"/>
      <abs id="page-4"/>
      <abs id="page-5"/>
    </tab>
```
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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>rows</td>
<td>Integer</td>
<td></td>
<td>Specifies the number of rows in the grid</td>
</tr>
<tr>
<td>cols</td>
<td>Integer</td>
<td></td>
<td>Specified the number of columns in the grid</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>orientation</td>
<td>horizontal</td>
<td></td>
<td>The first component will be on the left and the second component will be on the right.</td>
</tr>
<tr>
<td></td>
<td>vertical</td>
<td></td>
<td>The first component will be on the top and the second component will be on the bottom.</td>
</tr>
</tbody>
</table>

Default values shown in bold.

Child Component Attributes

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>weight</td>
<td>Double value between +0.0 and 1.0</td>
<td>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.</td>
<td></td>
</tr>
<tr>
<td>minw</td>
<td>Positive integer</td>
<td>The minimum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
<td></td>
</tr>
<tr>
<td>minh</td>
<td>Positive integer</td>
<td>The minimum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
<td></td>
</tr>
<tr>
<td>maxw</td>
<td>Positive integer</td>
<td>The maximum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
<td></td>
</tr>
<tr>
<td>maxh</td>
<td>Positive integer</td>
<td>The maximum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
<td></td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tabposition</td>
<td>north&lt;br/&gt;east&lt;br/&gt;south&lt;br/&gt;west&lt;br/&gt;none</td>
<td>How the tabs are rendered within their quadrant is determined by the look and feel.</td>
<td>Specifies the placement of the tabs.</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
<td>Tabs will be hidden and the visible component must be controlled through the “reveal” tag or the OGP REVEAL_ELEMENT trap.</td>
</tr>
<tr>
<td>tablayout</td>
<td>scroll&lt;br/&gt;stack</td>
<td>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”).</td>
<td></td>
</tr>
<tr>
<td>tabheight</td>
<td>Integer</td>
<td>Tab will not resize below minimum size to render tab label text</td>
<td>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.</td>
</tr>
<tr>
<td>onchange</td>
<td>ogScript String</td>
<td>The provided snippet of ogScript is triggered when the selected tab changes.</td>
<td>Current tab index is: this.getSelectedIndex()</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Current tab name is: this.getTitleAt(this.getSelectedIndex())</td>
<td></td>
</tr>
</tbody>
</table>

Default values shown in bold.
Child Component Attributes

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td>This attribute is used in elements contained within a tab tag.</td>
<td>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.</td>
</tr>
<tr>
<td>selected</td>
<td>default forced none</td>
<td>This attribute is used in elements contained within a tab tag.</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

\[
\begin{array}{c}
\text{<table container attributes>}
\text{<tr>}
\quad \text{<component child component attributes> } \text{</component>}
\quad \text{<component child component attributes> } \text{</component>}
\quad \ldots
\text{</tr>}
\text{<tr>}
\quad \text{<component child component attributes> } \text{</component>}
\quad \text{<component child component attributes> } \text{</component>}
\quad \ldots
\text{</tr>}
\text{. . .}
\text{</table>}
\end{array}
\]

Child Tags

<table>
<thead>
<tr>
<th>Tag</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tr&gt;</td>
<td></td>
<td></td>
<td>Encapsulates a row.</td>
</tr>
<tr>
<td>&lt;component&gt;</td>
<td>Any valid component tag</td>
<td></td>
<td>Defines the component for a table cell. Must be a child of a \text{tr} tag.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fill</td>
<td></td>
<td></td>
<td>Controls how the component inside of a table cell fills the cell itself.</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
<td>Uses the component’s natural width and height and floats it inside of the cell.</td>
</tr>
<tr>
<td></td>
<td>Horizontal</td>
<td></td>
<td>Uses the component’s natural height but fills the horizontal space.</td>
</tr>
<tr>
<td></td>
<td>Vertical</td>
<td></td>
<td>Uses the component’s natural width but fills the vertical space.</td>
</tr>
<tr>
<td></td>
<td>both</td>
<td></td>
<td>Ignores the component’s natural width and fills the entire cell.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------------------------------</td>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>anchor</td>
<td>center, north, northeast, east, south, southeast, southwest, west, northwest</td>
<td>Cells must not collide</td>
<td>If the fill is set to anything other than both, this controls where the component is attached to the cell.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><img src="compass.png" alt="Compass" /></td>
</tr>
<tr>
<td>rowspan</td>
<td>Positive integer</td>
<td>Cells must not collide</td>
<td>The number of rows spanned by a cell.</td>
</tr>
<tr>
<td>colspan</td>
<td>Positive integer</td>
<td>Cells must not collide</td>
<td>The number of columns spanned by a cell.</td>
</tr>
<tr>
<td>insets</td>
<td>4 positive integers separated by commas, e.g. “5,5,5,5”</td>
<td>Cells must not collide</td>
<td>Specifies padding around the component. The 4 numbers represent the top, left, bottom, and right padding. The insets are specified in pixels.</td>
</tr>
<tr>
<td>weightx</td>
<td>Double value between +0.0 and 1.0</td>
<td></td>
<td>Specifies how to distribute extra horizontal space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>weighty</td>
<td>Double value between +0.0 and 1.0</td>
<td></td>
<td>Specifies how to distribute extra vertical space.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>orientation</td>
<td>horizontal, vertical</td>
<td>Only applies to element tags that return multiple components.</td>
<td>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).</td>
</tr>
<tr>
<td>minw</td>
<td>Positive integer</td>
<td></td>
<td>The minimum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
</tr>
<tr>
<td>minh</td>
<td>Positive integer</td>
<td></td>
<td>The minimum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
</tr>
<tr>
<td>maxw</td>
<td>Positive integer</td>
<td></td>
<td>The maximum width of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
</tr>
<tr>
<td>maxh</td>
<td>Positive integer</td>
<td></td>
<td>The maximum height of the component in pixels. This is considered a hint and may or may not be honored by DashBoard.</td>
</tr>
<tr>
<td>placeholders</td>
<td>Positive integer</td>
<td>Default 0</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>A value of 0 means that the tag is ignored if no elements were returned (or the tag is undefined).</td>
</tr>
<tr>
<td>maxperrow</td>
<td>Positive integer</td>
<td>Default -1</td>
<td>A value &gt; 0 defines the maximum number of elements in a row. Additional elements will be placed on the next row.</td>
</tr>
</tbody>
</table>

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](http://docs.oracle.com/javase/8/docs/api/java/awt/GridBagLayout.html)

**Example**

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

```xml
<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>
  </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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>editlock</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>encrypt</strong></td>
<td>Encrypts a panel to protect the source code.</td>
</tr>
<tr>
<td><strong>gridsize</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>keepalive</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 encrypt 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>`.

**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**

```xml
<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

```xml
<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:

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>reveal</td>
<td>Brings hidden tab pages to the front</td>
</tr>
<tr>
<td>drawer</td>
<td>Creates a container that allows you to add drawer tabs to maximize panel space, by organizing additional content in hidden drawer tabs.</td>
</tr>
<tr>
<td>ext</td>
<td>Opens the editor for a specified node in the DashBoard Tree.</td>
</tr>
<tr>
<td>exit</td>
<td>Creates an exit button that, when clicked, causes DashBoard to close the current panel, window, or application.</td>
</tr>
<tr>
<td>help</td>
<td>Creates a help pop-up button which can display a custom help title and</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>image</td>
<td>Displays a static image</td>
</tr>
<tr>
<td>label</td>
<td>Creates a static text label</td>
</tr>
<tr>
<td>button</td>
<td>Creates a button</td>
</tr>
<tr>
<td>browser</td>
<td>Creates a web browser window</td>
</tr>
<tr>
<td>blank</td>
<td>Placeholder, used to leave a blank cell in simplegrid, table and flow containers</td>
</tr>
<tr>
<td>lock</td>
<td>Allows DashBoard client screen to be locked</td>
</tr>
<tr>
<td>memory</td>
<td>The memory manager widget allows you to add a memory status indicator bar to monitor the current memory usage of the DashBoard application.</td>
</tr>
<tr>
<td>widget</td>
<td>Creates an instance of a custom widget</td>
</tr>
<tr>
<td>wizard</td>
<td>You can create a basic wizard, that is also customizable using script.</td>
</tr>
</tbody>
</table>

**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:

![Ultritouch Panel with Drawers](image)

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**

```xml
<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>
    </abs>
    <drawer height="380" left="163" tabfill="none" top="141" width="538">
        <abs anchor="east" height="48" id="north1" name="north1" width="249"/>
    </drawer>
</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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>The name to display on the button.</td>
</tr>
<tr>
<td>targetid</td>
<td>list of Strings separated by commas or semicolons.</td>
<td>Each string in the value must refer to the id of another component.</td>
<td>Specifies the element ids to show.</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>The name to display on the button.</td>
</tr>
<tr>
<td>objectid</td>
<td>String</td>
<td></td>
<td>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 frame.</td>
</tr>
</tbody>
</table>
### 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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>The name to display on the button.</td>
</tr>
<tr>
<td>level</td>
<td>String</td>
<td></td>
<td>The level can be set to one of the following: panel, window or application. Where:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• panel closes the current panel.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• window closes the current window.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Application closes the DashBoard application.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Note: Setting level to window can also result in exiting the DashBoard application if only one window is open when the button is pressed.</td>
</tr>
<tr>
<td>objectid</td>
<td>String</td>
<td></td>
<td>Specifies the id of the components to show.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>DashBoard provides a few shortcuts to reference elements under the device</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>view.</td>
<td></td>
<td></td>
<td>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 &quot;FileNavigator,&quot; the objectid specifies a path and filename of a resource in the DashBoard file navigator, rather than the tree view.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>button</th>
<th>label</th>
<th>Optional</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;button&quot; = display as a button</td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;label&quot; = display as a clickable label</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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>
</html>
</help>
```

**Control Attributes**

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

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

Default values shown in **bold**.
Examples

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

```html
<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

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

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>URL String</td>
<td>Required. Must be a fully qualified URL.</td>
<td>Set the background image of the component.</td>
</tr>
<tr>
<td>height</td>
<td>Integers</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>top</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bottom</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------</td>
<td>-------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>width</td>
<td>Integers</td>
<td></td>
<td>It <em>left</em> and <em>right</em> are both specified, or <em>width</em> is specified, the image will be stretched to the width specified. Otherwise, the image's native width is used.</td>
</tr>
<tr>
<td>left</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>right</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Examples**

The following example places an image at its native size:

```xml
<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.

```xml
<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**

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

**Attributes**

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

Default values shown in **bold**.

**Examples**

```xml
<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

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

Attributes

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

Default values shown in **bold**.

Examples

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

```html
<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](image)

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

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

Attributes
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>url</td>
<td>URL String</td>
<td>Required. Must be a fully qualified URL.</td>
<td>The URL to use for the provided browser.</td>
</tr>
<tr>
<td>width</td>
<td>Positive integer</td>
<td>Required</td>
<td>The width (in pixels) of the browser</td>
</tr>
<tr>
<td>height</td>
<td>Positive integer</td>
<td>Required</td>
<td>The height (in pixels) of the browser</td>
</tr>
</tbody>
</table>

**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**

```xml
<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**

```xml
<blank attributes />
```

**Attributes**

See [General Attributes](#).

**Example**

```xml
<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>
```
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

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

Attributes

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Text to display on the button. Text will be rendered beside the lock icon.</td>
</tr>
</tbody>
</table>

Example

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

When locked, the DashBoard UI will be darkened, with an unlock 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
A memory manager widget appears in the specified area.

![Memory Manager Widget](image)

**Figure 79 – Lock Screen Widget**

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td></td>
<td>Widget identifier.</td>
</tr>
<tr>
<td>height</td>
<td>String</td>
<td></td>
<td>Height of the memory manager widget.</td>
</tr>
<tr>
<td>width</td>
<td>String</td>
<td></td>
<td>Width of the memory manager widget.</td>
</tr>
<tr>
<td>left</td>
<td>String</td>
<td></td>
<td>Offsets the memory manager widget a select number of pixels from the left side margins of the panel.</td>
</tr>
<tr>
<td>right</td>
<td>String</td>
<td></td>
<td>Offsets the memory manager widget a select number of pixels from the right side margins of the panel.</td>
</tr>
<tr>
<td>top</td>
<td>String</td>
<td></td>
<td>Offsets the memory manager widget a select number of pixels from the top margins of the panel.</td>
</tr>
<tr>
<td>bottom</td>
<td>String</td>
<td></td>
<td>Offsets the memory manager widget a select number of pixels from the bottom margins of the panel.</td>
</tr>
</tbody>
</table>

**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**

```xml
<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:

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>widgetid</td>
<td>String</td>
<td>Must match the <code>id</code> of a declared widgetdescriptor.</td>
<td>Widget identifier.</td>
</tr>
<tr>
<td>baseOID</td>
<td>String</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>
See Also

- `widgetdescriptor`
- `config`
- `param`

Examples

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

```xml
<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”:

```xml
<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.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>api</td>
<td>Provides a location for global ogScript code.</td>
</tr>
<tr>
<td>context (device context)</td>
<td>A device context is a data structure that contains information about the attributes of a device data source.</td>
</tr>
<tr>
<td>meta</td>
<td>This is a convenient parent tag for all non-UI tags.</td>
</tr>
<tr>
<td>subscription</td>
<td>This tag indicates the list of subscription oids that the panel wishes to receive parameter updates from the OGP JSON device. <strong>Note</strong>: The device source must support subscriptions protocol.</td>
</tr>
<tr>
<td>widgets</td>
<td>This is a container for user-defined widget descriptors.</td>
</tr>
<tr>
<td>widgetdescriptor</td>
<td>Defines a custom widget.</td>
</tr>
<tr>
<td>lookup</td>
<td>A lookup defines constants to be substituted inside of other tag attributes or used in ogScript blocks.</td>
</tr>
<tr>
<td>style</td>
<td>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.</td>
</tr>
<tr>
<td>color</td>
<td>Defines or overrides a color constant for use within style hints.</td>
</tr>
<tr>
<td>ogscript</td>
<td>Defines an ogScript code snippet to handle an event on a UI element or parameter.</td>
</tr>
<tr>
<td>constraint</td>
<td>Defines the constraint of a parameter.</td>
</tr>
<tr>
<td>Tag</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>params</td>
<td>The parent container for parameters defined within the OGLML document.</td>
</tr>
<tr>
<td>timer</td>
<td>The timer tag fires events at regular intervals.</td>
</tr>
<tr>
<td>listener</td>
<td>The listener tag allows an OGLML page to process network communications using protocols not already available.</td>
</tr>
<tr>
<td>task</td>
<td>Defines a block of ogScript to be run when an event happens in the system.</td>
</tr>
<tr>
<td>timertask</td>
<td>Defines a block of ogScript to be run when a timer goes off.</td>
</tr>
<tr>
<td>include</td>
<td>This tag allows an OGLML document to be assembled from several individual XML files or fragments.</td>
</tr>
</tbody>
</table>

### 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**

```xml
<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**

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

**Syntax for Panels that Support Subscriptions**

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

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

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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oids</td>
<td>String of comma separated OIDs</td>
<td>*Required to support OGP devices that support the subscription protocol.</td>
<td>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. <strong>Note:</strong> This tag can only be added to a CustomPanel that indicates support for <code>subscriptions=&quot;true&quot;</code> in the context or top level attributes of the panel. It is recommended to nest the subscription oid list within a <code>meta</code> tag. See the details below for more information wildcards.</td>
</tr>
</tbody>
</table>

**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**

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

**Attributes**

None.

### widgets

This is a container for user-defined widget descriptors.

**Syntax**

```xml
<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**

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

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Must be unique</td>
<td>Widget identifier.</td>
</tr>
<tr>
<td>structtype</td>
<td>String</td>
<td></td>
<td>Specifies a dependency of the widget upon a global struct parameter with matching <code>structtype</code>. 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 <code>structtype</code>.</td>
</tr>
<tr>
<td>baseurl</td>
<td>String</td>
<td>Must be a valid, fully qualified URL.</td>
<td>When specified, the widget descriptor will be fetched from a document specified by the URL, rather than inline.</td>
</tr>
</tbody>
</table>

## 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.

```xml
<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:

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

The widget is then displayed with the following:

```xml
<widget widgetid="alarmgrid" top="100" left="100"/>
```
The following example displays the widget, overriding the value of parameter \texttt{str2} with the value “New String Value”:

\begin{verbatim}
<widget left="100" top="300" widgetid="alarmgrid">
  <config>
    <params>
      <param oid="str2" value="New String Value"/>
    </params>
  </config>
</widget>
\end{verbatim}

**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 \texttt{api} tag.

**Syntax**

\begin{verbatim}
<lookup id="id-string" scope="scope">
  <entry key="key">value</entry>
  . . .
</lookup>
\end{verbatim}

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>scope</td>
<td>private public window</td>
<td>If “private”, the lookup must define the \texttt{id} attribute.</td>
<td>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 \texttt{id}.</td>
</tr>
<tr>
<td>id</td>
<td>string</td>
<td>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 \texttt{%const['id']['key']}%</td>
<td></td>
</tr>
<tr>
<td>code</td>
<td>true false</td>
<td>Must be set true if the lookup \texttt{value} contains executable script.</td>
<td></td>
</tr>
<tr>
<td>multiline</td>
<td>true false</td>
<td>Must be set true if lookup \texttt{value} contains multi-line strings.</td>
<td></td>
</tr>
</tbody>
</table>

Default values shown in **bold**.
Example

The following tag creates a public lookup

```ogscript
<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.

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

The following tag creates a private scope lookup

```ogscript
<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.

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

The following tag creates a block of code lookup:

```ogscript
<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
<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**

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

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td>Must not contain a semicolon</td>
<td>The ID to use when referencing the style.</td>
</tr>
<tr>
<td>value</td>
<td>string</td>
<td>Must not contain any circular references to itself.</td>
<td>Value contains a style hint string following the same format used in the <code>style</code> attribute of other tags.</td>
</tr>
</tbody>
</table>

**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`).

```xml
<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](image)

**color**

Defines or overrides a color constant for use within style hints.

**Syntax**

```xml
<color id="color-name" value="color-value"/>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>string</td>
<td></td>
<td>The ID to use when referencing the color.</td>
</tr>
<tr>
<td>value</td>
<td>#RRGGBB or #color-constant or #RRGGBBAA</td>
<td>Value contains a style hint string following the same format used in the <code>color</code> style attribute.</td>
<td></td>
</tr>
</tbody>
</table>
Example
The following example defines a color constant \texttt{ColorBlue} and applies it to the background of a button widget.

\begin{verbatim}
<color id="VibrantBlue" value="#0000FF"/>
<button name="Blue Button" style="bg#VibrantBlue"/>
\end{verbatim}

\texttt{ogscript}

Defines an ogScript code snippet to handle an event on a UI element or parameter.

\textbf{Syntax}

\begin{verbatim}
<ogscript handles="\texttt{eventType}">
\texttt{ogScript code}
</ogscript>
\end{verbatim}

\textbf{Attributes}

\begin{tabular}{|c|c|c|c|}
\hline
\textbf{Attribute} & \textbf{Values} & \textbf{Restrictions} & \textbf{Description} \\
\hline
use & online & Script will only run on a real device & \\
offlne & offline & Script will only run on a file-based device & \\
both & both & Script will run always & \\
\hline
targetid & string & The ID of the UI element to target. & \\
\hline
handles & Multiple "handles" arguments can be supplied, separated by commas. & The type of event that triggers the script. & \\
\hline
attributechange & Can be used to trigger scripts when selected NK device is changed or monitor status of FTP download/upload: & & \\
\hline
\begin{verbatim}
<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.getTotalBytesTransfered() / 1024) + ' kb');
}</ogscript>
\end{verbatim} & & & \\
\hline
dragvalue & Must specify something to return (generally a string or a number) when the element is dragged. & & \\
\hline
onaction & Triggered when a button is pressed. & & \\
\hline
\end{tabular}
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>onchange</td>
<td>Only supported by tabs and parameters.</td>
<td></td>
<td>Triggers script when parameter or tab is changed.</td>
</tr>
<tr>
<td>onclick</td>
<td></td>
<td></td>
<td>Triggers script when element is clicked.</td>
</tr>
<tr>
<td>onclose</td>
<td></td>
<td></td>
<td>Triggered when the panel has been closed (can be used for cleaning-up).</td>
</tr>
<tr>
<td>oncontextmenu</td>
<td></td>
<td></td>
<td>Triggers script when the element is right-clicked, or tapped and held.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>To create a context menu, define an array of menu options, each associated with a segment of ogScript.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For more information, see <a href="#">Example of a Context menu</a> on page 146.</td>
</tr>
<tr>
<td>ondrag</td>
<td></td>
<td></td>
<td>Triggers script when the element is dragged</td>
</tr>
<tr>
<td>ondrop</td>
<td></td>
<td></td>
<td>Triggers script when another component is dropped on the component.</td>
</tr>
<tr>
<td>onkeypress</td>
<td></td>
<td></td>
<td>Triggers script when the component has focus and a keyboard key is pressed.</td>
</tr>
<tr>
<td>onkeydown</td>
<td></td>
<td></td>
<td>Triggers script when the component has focus and a keyboard key is released.</td>
</tr>
<tr>
<td>onlassoout</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is no longer inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoover</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoout</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is no longer inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoover</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoout</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is no longer inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoover</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoout</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is no longer inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onlassoover</td>
<td></td>
<td></td>
<td>Triggers script when a lassostart operation has started and the component with the selected ID is inside of its bounding rectangle.</td>
</tr>
<tr>
<td>onload</td>
<td></td>
<td></td>
<td>Triggered when the panel has finished loading or is reloaded</td>
</tr>
<tr>
<td>onmousedown</td>
<td></td>
<td></td>
<td>Triggers script at onmouse click down event.</td>
</tr>
<tr>
<td>onmouseenter</td>
<td></td>
<td></td>
<td>Triggers script when the pointer moves over the component.</td>
</tr>
<tr>
<td>onmouseleave</td>
<td></td>
<td></td>
<td>Triggers script when the pointer leaves the component.</td>
</tr>
<tr>
<td>onmouseup</td>
<td></td>
<td></td>
<td>Triggers script on mouse click up event.</td>
</tr>
<tr>
<td>onmousemove</td>
<td></td>
<td></td>
<td>Triggers script when mouse moves over component</td>
</tr>
<tr>
<td>onmouseup</td>
<td></td>
<td></td>
<td>Triggers script when the mouse is released after having been pressed while pointing to the component.</td>
</tr>
</tbody>
</table>
### Attribute Table

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>onresize</td>
<td></td>
<td>Triggers script when the component is resized.</td>
</tr>
<tr>
<td>oid</td>
<td>Positive integer</td>
<td>Must be a defined OID. Only applies to &quot;onchange&quot;</td>
<td>The OID of the parameter to target.</td>
</tr>
<tr>
<td>element</td>
<td>List of array indices separated by commas</td>
<td>All array elements referenced must exist in the parameter value. Only applies to &quot;onchange&quot;</td>
<td>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 &quot;0&quot; for a non-array parameter.</td>
</tr>
<tr>
<td>script</td>
<td>ogScript</td>
<td>Can also be the text content of the &lt;ogscript&gt; tag.</td>
<td>The script to run when triggered by any of the events listed in &quot;handles&quot;.</td>
</tr>
</tbody>
</table>

**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:

```html
<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:

```html
<constraint constrainttype="ctype" constraint="cvalue" id="constraint-id" />
```

Constraints defined within a *param* tag with inline constraint values:

```html
<param constrainttype="ctype" constraint="cvalue" param-attributes />
```
Choice and Alarm Constraints:

```xml
<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:

```xml
<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

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Constraint Type</th>
<th>Param Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unconstrained</td>
<td>INT_NULL</td>
<td>INT16_PARAM, INT16 ARRAY, INT32_PARAM, INT32 ARRAY</td>
</tr>
<tr>
<td></td>
<td>FLOAT_NULL</td>
<td>FLOAT16_PARAM, FLOAT ARRAY</td>
</tr>
<tr>
<td>Range Constraint</td>
<td>INT_RANGE</td>
<td>INT16_PARAM, INT16 ARRAY, INT32_PARAM, INT32 ARRAY</td>
</tr>
<tr>
<td></td>
<td>INT_STEP_RANGE</td>
<td>INT16_PARAM, INT16 ARRAY, INT32_PARAM, INT32 ARRAY</td>
</tr>
<tr>
<td></td>
<td>FLOAT_RANGE</td>
<td>FLOAT16_PARAM, FLOAT ARRAY</td>
</tr>
<tr>
<td></td>
<td>FLOAT_STEP_RANGE</td>
<td>FLOAT16_PARAM, FLOAT ARRAY</td>
</tr>
<tr>
<td>Integer Choice Constraint</td>
<td>INT_CHOICE</td>
<td>INT16_PARAM, INT16 ARRAY, INT32_PARAM, INT32 ARRAY</td>
</tr>
<tr>
<td>String Choice Constraint</td>
<td>STRING_CHOICE</td>
<td>STRING16_PARAM, STRING ARRAY</td>
</tr>
<tr>
<td>Alarm Table</td>
<td>ALARM_TABLE</td>
<td>INT16_PARAM, INT32_PARAM</td>
</tr>
<tr>
<td>Constraint Reference</td>
<td>ID_REFERENCE</td>
<td>All</td>
</tr>
<tr>
<td>Structure</td>
<td>STRUCT</td>
<td>STRUCT16_PARAM, STRUCT ARRAY</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrainttype</td>
<td>INT_NULL</td>
<td><strong>param type:</strong></td>
<td>Parameter is unconstrained</td>
</tr>
<tr>
<td></td>
<td>FLOAT_NULL</td>
<td><strong>param type:</strong></td>
<td></td>
</tr>
</tbody>
</table>

**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

```xml
<constraint id="constraint-id" constrainttype="constraint-type">
  ...
</constraint>
```

```xml
<param constrainttype="ID_REFERENCE" constraint="constraint-id" attributes />
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Unique identifier for this constraint</td>
<td></td>
</tr>
<tr>
<td>constrainttype</td>
<td>Any valid</td>
<td>Param type must be compatible with the</td>
<td>Unique identifier for this constraint</td>
</tr>
<tr>
<td></td>
<td>constraint type</td>
<td>referenced constraint.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>See Constraint Types for valid constraint types.</td>
</tr>
</tbody>
</table>

Examples

The following example creates a global constraint VideoFormat. Params 0x501, 0x502 and 0x503 are all constrained using this constraint definition.

```xml
<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>
```

```xml
<param constrainttype="IDREFERENCE" constraint="VideoFormat" name="Vid1" oid="0x501" type="INT16_PARAM"/>
```

```xml
<param constrainttype="IDREFERENCE" constraint="VideoFormat" name="Vid2" oid="0x502" type="INT16_PARAM"/>
```

```xml
<param constrainttype="IDREFERENCE" 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:

```xml
<param constraint="min;max;" constrainttype="constraint-type" attributes />
```
Min / Max Constraint with Display-Min and Display-Max:

```xml
<param constraint="min;max;display-min;display-max;"
    constrainttype="constraint-type" attributes/>
```

Min / Max Step Constraint:

```xml
<param constraint="min;max;step"
    constrainttype="constraint-type"
    attributes/>
```

Min / Max Step Constraint with Display-Min and Display-Max:

```xml
<param constraint="min;max;display-min;display-max;step"
    constrainttype="constraint-type" attributes/>
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>constrainttype</td>
<td>INT_RANGE, INT_STEP_RANGE, FLOAT_RANGE, FLOAT_STEP_RANGE</td>
<td>param type: INT16_PARAM, INT16_ARRAY, INT32_PARAM, INT32_ARRAY, FLOAT_PARAM, FLOAT_ARRAY</td>
<td>Type of constraint</td>
</tr>
<tr>
<td>constraint</td>
<td>min</td>
<td>Required</td>
<td>Minimum value to which a parameter can be set</td>
</tr>
<tr>
<td></td>
<td>max</td>
<td>Required</td>
<td>Minimum value to which a parameter can be set</td>
</tr>
<tr>
<td></td>
<td>display-min</td>
<td>Optional; must be used with display_max</td>
<td>The displayed value of the parameter when the parameter has a value of min. The default value is min.</td>
</tr>
<tr>
<td></td>
<td>display-max</td>
<td>Optional; must be used with display_min</td>
<td>The displayed value of the parameter when the parameter has a value of max. The default value is max.</td>
</tr>
<tr>
<td></td>
<td>step</td>
<td>xxx_STEP_RANGE only</td>
<td>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.</td>
</tr>
</tbody>
</table>

Examples

The following example constrains a FLOAT_PARAM to [0,100]:

```xml
<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:

```xml
<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**

```xml
<param constrainttype="INT_CHOICE" type="param-type" attributes >
   <constraint key="choice1-key">choice1-value</constraint>
   <constraint key="choice2-key">choice2-value</constraint>
   . . .
</param>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>INT16_PARAM</td>
<td>Parameter must be integer type.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT32_PARAM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT16_ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>INT32_ARRAY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>key</td>
<td>Integer</td>
<td>Numeric assignment of current</td>
<td>Numeric assignment of current enumerated choice.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>enumerated choice.</td>
<td></td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td></td>
<td>Text name for the current enumerated choice.</td>
</tr>
</tbody>
</table>

**Examples**

The following constraint provides an enumerated choice:

```xml
<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**

```xml
<param constrainttype="STRING_CHOICE" type="param-type" attributes >
   <constraint>value</constraint>
   <constraint>value</constraint>
   . . .
</param>
```
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>STRING_PARAM</td>
<td></td>
<td>Parameter must be string type.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td></td>
<td>Available strings for drop-down widget</td>
</tr>
</tbody>
</table>

### Examples

The following constraint provides five string options for a String parameter.

```xml
<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](image)

**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**

```xml
<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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
</table>
| type      | INT16_PARAM
           | INT32_PARAM   | Parameter must be integer type.    |
| key       | Integer          | INT16: 0..15  | The bit position for the alarm (0 is LSB). |
|           |                  | INT32: 0..31   |                                     |
| 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:

```xml
<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**

```xml
<param constrainttype="STRUCT" structtype="struct-type" templateoid="template-oid" type="STRUCT" param-attributes>

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>String</td>
<td>Required</td>
<td>Set to &quot;STRUCT&quot;</td>
</tr>
<tr>
<td>templateoid</td>
<td>String</td>
<td></td>
<td>Specifies a template OID to pre-populate the structure. All parameters, constraints and widgets for the sub-OIDs are copied from the template.</td>
</tr>
<tr>
<td>structtype</td>
<td>String</td>
<td>Must be unique</td>
<td>Defines the structure type. Used by PanelBuilder to type-check custom widgets against defined struct parameters.</td>
</tr>
</tbody>
</table>

**Examples**

The following code is an example of a struct definition.

```xml
<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; 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; 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; 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 Benoiff &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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Optional</td>
<td>The ID used to reference this timer. Required for ogScript, child timers, or external &lt;timertask/&gt; tags to interact with the timer.</td>
</tr>
<tr>
<td>source</td>
<td>String</td>
<td>Optional</td>
<td>If used, the timer being defined will be a child of the timer with the given ID.</td>
</tr>
<tr>
<td>rate</td>
<td>Long</td>
<td>Not applicable if &quot;source&quot; is set.</td>
<td>The rate (in milliseconds) at which the timer fires.</td>
</tr>
<tr>
<td>delay</td>
<td>Long</td>
<td>Not applicable if &quot;source&quot; is set.</td>
<td>The delay (in milliseconds) before the timer initially fires.</td>
</tr>
<tr>
<td>pattern</td>
<td>String</td>
<td></td>
<td>The display pattern for the timer’s current time: <a href="https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html">https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html</a></td>
</tr>
<tr>
<td>start</td>
<td>Long or time in format of “pattern”</td>
<td>The start value of the timer. If start &gt; stop, timer counts down. If start is undefined, the timer is ‘clock mode’</td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>Long or time in format of “pattern”</td>
<td>The start value of the timer. If start &gt; stop, timer counts down. If start is undefined, the timer is ‘clock mode’</td>
<td></td>
</tr>
<tr>
<td>autostart</td>
<td>True or false</td>
<td>Default value is true if ‘clock mode’ is used.</td>
<td>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.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connecthost</td>
<td>String</td>
<td>Cannot be used if listenport is defined.</td>
<td>The hostname of the remote host to connect to.</td>
</tr>
<tr>
<td>connectport</td>
<td>Integer</td>
<td>Cannot be used if listenport is defined.</td>
<td>The port to connect to on the remote host.</td>
</tr>
<tr>
<td>listenport</td>
<td>Integer</td>
<td>Cannot be used if connectport/connect host are defined.</td>
<td>The local port to listen on for new connections.</td>
</tr>
<tr>
<td>delimitertype</td>
<td>newline bytes</td>
<td>Required</td>
<td>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 “delimiter” 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.</td>
</tr>
<tr>
<td></td>
<td>fixedlen varlen</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>string</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>delimiter</td>
<td>May be required depending on value of delimitertype</td>
<td>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” varlen: 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”.</td>
<td></td>
</tr>
<tr>
<td>delimiter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>syncword</td>
<td>Optional</td>
<td></td>
<td>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”</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>--------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>blockingpause</td>
<td>true</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td></td>
</tr>
<tr>
<td>buttontype</td>
<td>toggle</td>
<td></td>
<td>If no button is defined, the listener is automatically started. If a button is defined, this allows the user to toggle the listener on/off.</td>
</tr>
<tr>
<td></td>
<td>none</td>
<td></td>
<td></td>
</tr>
<tr>
<td>autostart</td>
<td>true</td>
<td></td>
<td>Whether or not the listener should be automatically started. This is always true if no buttontype has been defined.</td>
</tr>
<tr>
<td></td>
<td>false</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Example**

```xml
<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**

```xml
<component>
    <task tasktype="task-type">ogScript-code</task>
</component>
```
Attributes

<table>
<thead>
<tr>
<th>Tag</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasktype</td>
<td>*ogscript robot vdcp rostalk opparamset timercontrol</td>
<td></td>
<td>This attribute tells the editor user interface what type of task is contained in the tag body. Manually-edited tasks should simply use ogscript.</td>
</tr>
</tbody>
</table>

**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**

```xml
<container>
  <timer id="timer-id"/>
  <container>
    <container>
      <timertask tasktype="task-type" timerid="timer-id">
        ogScript Code
      </timertask>
    </container>
  </container>
</container>
```

**Attributes**

<table>
<thead>
<tr>
<th>Tag</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tasktype</td>
<td>*ogscript robot vdcp rostalk opparamset timercontrol</td>
<td></td>
<td>This attribute tells the editor user interface what type of task is contained in the tag body. Manually-edited tasks should simply use ogscript.</td>
</tr>
<tr>
<td>timerid</td>
<td>String</td>
<td>Must match the id attribute of a timer tag accessible in this tasks's scope.</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*src</td>
<td>URL for http, https, or “eo”</td>
<td>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)</td>
<td></td>
</tr>
</tbody>
</table>

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
<?xml version="1.1" encoding="UTF-8"?>
<frame>
  <card>
    <params>
      <param/>
      <param/>
    . . .
    </params>
    <statusmenu>
      <menu>
        <param/>
        <param/>
      </menu>
    </statusmenu>
  </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.

![Diagram of OGP Device and DashBoard sending commands](Diagram.png)

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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>String</td>
<td>*Required</td>
<td>Command oid.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Not required</td>
<td>Command name.</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>*Required</td>
<td>Data type for the command.</td>
</tr>
<tr>
<td>readonly</td>
<td>Boolean</td>
<td>Not required</td>
<td>If set to true, the parameter is read-only.</td>
</tr>
<tr>
<td>widget</td>
<td>String</td>
<td>Not required</td>
<td>The widget used to display the data in DashBoard.</td>
</tr>
<tr>
<td>constraint</td>
<td>constraint Object</td>
<td>Not required</td>
<td>Parameter Constraint.</td>
</tr>
<tr>
<td>config</td>
<td>config Object</td>
<td>Not required</td>
<td>Extended parameter configuration.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>Not required</td>
<td>Value of the parameter. Defines an argument that can be passed to the command.</td>
</tr>
</tbody>
</table>

**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 –Visual 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"
      }
    ]
  }
}
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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>String</td>
<td>*Required</td>
<td>Command oid.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Not required</td>
<td>Command name.</td>
</tr>
<tr>
<td>type</td>
<td>String</td>
<td>*Required</td>
<td>Data type for the command.</td>
</tr>
<tr>
<td>readonly</td>
<td>Boolean</td>
<td>Not required</td>
<td>If set to true, the parameter is read-only.</td>
</tr>
<tr>
<td>widget</td>
<td>String</td>
<td>Not required</td>
<td>The widget used to display the data in DashBoard.</td>
</tr>
<tr>
<td>constraint</td>
<td>constraint Object</td>
<td>Not required</td>
<td>Parameter Constraint.</td>
</tr>
<tr>
<td>config</td>
<td>config Object</td>
<td>Not required</td>
<td>Extended parameter configuration.</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td>Not required</td>
<td>Value of the parameter. Defines an argument that can be passed to the command.</td>
</tr>
</tbody>
</table>

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**

```xml
<param>
  <config key="key">value</config>
  <config key="key">value</config>
  ...
</param>
```
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>String</td>
<td></td>
<td>Configuration parameter name</td>
</tr>
<tr>
<td>value</td>
<td>String</td>
<td></td>
<td>Configuration parameter value</td>
</tr>
</tbody>
</table>

Example

The following config object sets attributes of a graph widget:

```xml
<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

```xml
<constraint key="choice1-key">choice1-value</constraint>
```

Examples

The following constraint provides an enumerated choice:

```xml
<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:

```xml
<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**

```xml
<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">
  ...
</card>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autosave</td>
<td>true</td>
<td>false</td>
<td>If true, DashBoard will automatically save contents of the resources specified in the file from data store periodically.</td>
</tr>
<tr>
<td>online</td>
<td>true</td>
<td>false</td>
<td>Sets the device's online status. Normally should be set to <strong>true</strong>.</td>
</tr>
<tr>
<td>slot</td>
<td>Integer</td>
<td></td>
<td>Defines the slot-id for the device.</td>
</tr>
<tr>
<td>sourceframe</td>
<td>String</td>
<td></td>
<td><strong>node-id</strong> of the frame or device.</td>
</tr>
<tr>
<td>sourceframename</td>
<td>String</td>
<td></td>
<td>Name of the device. This is the top-level name shown in the DashBoard Tree</td>
</tr>
<tr>
<td>sourceid</td>
<td>String</td>
<td></td>
<td>The original node-id of the virtual device (used when saved as the offline configuration of a real device)</td>
</tr>
<tr>
<td>status</td>
<td>0</td>
<td>Not required for PanelBuilder</td>
<td>Status OK</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Status WARN</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>Status ERROR</td>
</tr>
<tr>
<td>statustext</td>
<td>String</td>
<td>Not required for PanelBuilder</td>
<td>Status text for the node.</td>
</tr>
<tr>
<td>version</td>
<td>String</td>
<td></td>
<td>Set to <strong>2.0</strong>.</td>
</tr>
</tbody>
</table>

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&gt;Slot 10&amp;lt;br&gt;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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>String</td>
<td></td>
<td>Display name of the frame</td>
</tr>
<tr>
<td>sourceid</td>
<td>String</td>
<td></td>
<td>The original node-id of the virtual device (used when saved as the offline configuration of a real device)</td>
</tr>
</tbody>
</table>

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.

Syntax

```
<menu menuid="menu-id" menustate="state" name="name" staticid="static-id">
  <param/>
  <param/>
  ...
</menu>
```
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuid</td>
<td>Integer</td>
<td>Required</td>
<td>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.</td>
</tr>
<tr>
<td>menustate</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>Menu is hidden</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Menu is displayed, but params are read-only</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>Menu is displayed and params are read/write (based upon individual parameter access permissions)</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Name of the menu. This name will appear in the menu tab.</td>
</tr>
<tr>
<td>staticid</td>
<td>Integer</td>
<td>Required</td>
<td>Unique numeric identifier for this menu. This value must be only set once and not changed.</td>
</tr>
</tbody>
</table>

### Example

The following example creates a menu called “Network Setup”.

```xml
<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

```xml
<menugroup menuid="id" name="menu-group-name">
  <menu/>
  <menu/>
  ...
</menugroup>
```

#### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuid</td>
<td>Integer</td>
<td>Required</td>
<td>Numeric ID for the menu group. This value must be only set once and not changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>menuid=0 corresponds to the openGear status menu</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>menuid=1 corresponds to the openGear configuration menu</td>
</tr>
</tbody>
</table>
### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Name of the menu group.</td>
</tr>
</tbody>
</table>

### Examples

The following example creates a menu group with two menus:

```xml
<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

```xml
<statusmenu menuid="id" name="menu-group-name">
  <menu/>
  <menu/>
  . . .
</statusmenu>
```

#### Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuid</td>
<td>Integer</td>
<td>Optional. Should be set to 0.</td>
<td>Numeric ID for the menu group. This value must be only set once and not changed. Defaults to 0.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Name of the menu group.</td>
</tr>
</tbody>
</table>
Example

The following example creates a status menu group with two menus:

```xml
<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

```xml
<configmenu menuid="id" name="menu-group-name">
  <menu/>
  <menu/>
  ...
</configmenu>
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>menuid</td>
<td>Integer</td>
<td>Optional. Should be set to 1.</td>
<td>Numeric ID for the menu group. This value must be only set once and not changed. Defaults to 1.</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td>Name of the menu group.</td>
<td></td>
</tr>
</tbody>
</table>

Example

The following example creates a status menu group with 2 menus:

```xml
<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"/>
  </menu>
</configmenu>
```
<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**

```xml
<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**

```xml
<param oid="oid" attributes/>

<param oid="oid" attributes>
  <constraint/>
  <constraint/>
  ...
  <config/>
  <config/>
  ...
</param>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>String</td>
<td>Required, except for subparams</td>
<td>The OID of the parameter (can be used to override an existing parameter).</td>
</tr>
<tr>
<td>suboid</td>
<td>String</td>
<td>Required for subparams</td>
<td>If the param declaration is a sub-param within a struct, the OID is specified in the suboid attribute.</td>
</tr>
<tr>
<td>access</td>
<td>0, 1</td>
<td></td>
<td>Parameter is read-only in DashBoard</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Parameter Name</td>
</tr>
<tr>
<td>Attribute</td>
<td>Values</td>
<td>Restrictions</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------</td>
<td>-------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>widget</td>
<td>Positive integer</td>
<td>Must be a valid widget hint</td>
<td>Defines the default widget hint for the param.</td>
</tr>
<tr>
<td>maxlength</td>
<td>Positive integer</td>
<td>Applies only to String/String Array parameters</td>
<td>The maximum length of any String element in the parameter.</td>
</tr>
<tr>
<td>precision</td>
<td>Positive integer</td>
<td></td>
<td>This field defines the number of digits following the decimal point displayed for printed numbers. It applies mainly to floating point numbers.</td>
</tr>
<tr>
<td>type</td>
<td>INT16</td>
<td></td>
<td>Param is 16-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>INT16_ARRAY</td>
<td></td>
<td>Param is an array of 16-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>INT32</td>
<td></td>
<td>Param is 32-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>INT32_ARRAY</td>
<td></td>
<td>Param is an array of 32-bit signed integer.</td>
</tr>
<tr>
<td></td>
<td>STRING</td>
<td></td>
<td>Param is a string.</td>
</tr>
<tr>
<td></td>
<td>STRING_ARRAY</td>
<td></td>
<td>Param is an array of strings.</td>
</tr>
<tr>
<td></td>
<td>FLOAT32</td>
<td></td>
<td>Param is a 32-bit (IEEE single) float.</td>
</tr>
<tr>
<td></td>
<td>FLOAT32_ARRAY</td>
<td></td>
<td>Param is an array of 32-bit (IEEE single) float.</td>
</tr>
<tr>
<td></td>
<td>STRUCT</td>
<td></td>
<td>Param is a struct.</td>
</tr>
<tr>
<td></td>
<td>STRUCT_ARRAY</td>
<td></td>
<td>Param is an array of struct.</td>
</tr>
<tr>
<td></td>
<td>BINARY_VALUE</td>
<td></td>
<td>Param is of unknown type.</td>
</tr>
<tr>
<td>constraint</td>
<td>Cvalue</td>
<td></td>
<td>See constraint tag for more details.</td>
</tr>
<tr>
<td>constrainttype</td>
<td>CType</td>
<td></td>
<td>See constraint tag for more details.</td>
</tr>
<tr>
<td>stateless</td>
<td>False</td>
<td></td>
<td>Parameters are saved to backing source.</td>
</tr>
<tr>
<td></td>
<td>True</td>
<td></td>
<td>Parameters are not saved.</td>
</tr>
<tr>
<td>value</td>
<td>Varies</td>
<td>Value type must be compatible with the specified type.</td>
<td>Specifies the initial value of the param. Arrays may be initialized by separating values with &quot;;&quot;.</td>
</tr>
<tr>
<td>config</td>
<td>Varies</td>
<td></td>
<td>Provides additional widget configuration parameters.</td>
</tr>
</tbody>
</table>

Default values shown in **bold**.

**Example**

```xml
<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 \texttt{STRUCT} param type. A \texttt{struct} contains a collection of parameters. \texttt{Structs} may not be nested. \texttt{Struct} must have a \texttt{constrainttype} of \texttt{STRUCT}. Members of the \texttt{struct} are declared through \texttt{subparam} tags within the \texttt{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 \texttt{templateoid} attribute.

Syntax

\[
\begin{array}{l}
<\text{param constrainttype}="\text{STRUCT}" \text{oid}="\text{oid}" \text{type}="\text{STRUCT}" \text{attributes}> \\
\text{<value>} \\
\quad <\text{subparam} \text{suboid}="\text{sub-oid}" \text{sub-param-attributes}/> \\
\quad <\text{subparam} \text{suboid}="\text{sub-oid}" \text{sub-param-attributes}/> \\
\quad \ldots \\
\text{</value>} \\
\text{</param>}
\end{array}
\]

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>oid</td>
<td>String</td>
<td>Required</td>
<td>The OID of the parameter (can be used to override an existing parameter).</td>
</tr>
<tr>
<td>access</td>
<td>0</td>
<td></td>
<td>Parameter is read-only in DashBoard</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Parameter is read-write in DashBoard</td>
</tr>
<tr>
<td>name</td>
<td>String</td>
<td></td>
<td>Parameter Name</td>
</tr>
<tr>
<td>widget</td>
<td>Positive integer</td>
<td>Must be a valid widget hint</td>
<td>Defines the default \texttt{widget hint} for the param.</td>
</tr>
<tr>
<td>type</td>
<td>STRUCT</td>
<td></td>
<td>Must be set to \texttt{STRUCT}.</td>
</tr>
<tr>
<td>structtype</td>
<td>String</td>
<td></td>
<td>Defines the structure type. Specifies a dependency of a widget upon a global struct parameter with matching \texttt{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 \texttt{structtype}.</td>
</tr>
<tr>
<td>templateoid</td>
<td>String</td>
<td></td>
<td>Specifies a template struct parameter to pre-populate the subparams.</td>
</tr>
<tr>
<td>constrainttype</td>
<td>STRUCT</td>
<td></td>
<td>Must be set to \texttt{STRUCT}</td>
</tr>
<tr>
<td>value</td>
<td></td>
<td></td>
<td>Container for \texttt{subparam} elements.</td>
</tr>
<tr>
<td>subparam</td>
<td>\texttt{param}</td>
<td>May not be a nested struct param</td>
<td>Member parameters, declared using the same syntax as a param declaration, with the exception that its \texttt{oid} is specified in the attribute \texttt{suboid}.</td>
</tr>
</tbody>
</table>

Default values shown in \textbf{bold}. 
Example

The following declares a struct parameter.

```xml
<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.

```xml
<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.

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**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mid</td>
<td>integer</td>
<td>Must be a defined top-level menu.</td>
<td><code>mid</code> of a defined menu. 0 = Status Menu 1 = Configuration Menu 2 = “Extra” Menu</td>
</tr>
</tbody>
</table>

**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:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mid</td>
<td>integer</td>
<td>Must be a defined OID Menu.</td>
<td>The static ID of the OID Menu to draw.</td>
</tr>
<tr>
<td>Popup</td>
<td>true</td>
<td><code>name</code> attribute must also be specified.</td>
<td>A button with the <code>name</code> attribute as its label is the component. When pressed, the menu will appear in a balloon dialog. For more information, see <a href="#">WIDGET_MENU_POPUP</a> on page 32.</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td></td>
<td>The menu is included as a single component.</td>
</tr>
<tr>
<td>Oglml</td>
<td>true</td>
<td></td>
<td>If the referenced menu has been overridden by an OGLML page, the OGLML version of the menu will be used.</td>
</tr>
<tr>
<td>false</td>
<td></td>
<td></td>
<td>The standard OGP menu without any OGLML will be used.</td>
</tr>
</tbody>
</table>
### Tabposition

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tabposition</td>
<td>north east south west</td>
<td>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.)</td>
<td>Specifies the placement of the tabs for any 3rd-level submenus.</td>
</tr>
</tbody>
</table>

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**

```xml
<param oid="oid" attributes/>

<param oid="oid" attributes>
  <constraint/>
  <constraint/>
  ...
  <config/>
  <config/>
  ...
</param>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Showlabel</td>
<td>true false</td>
<td></td>
<td>Display the parameter name as a label beside the parameter elements.</td>
</tr>
<tr>
<td>Oid</td>
<td>String</td>
<td>Must be a defined OID</td>
<td>The OID of the parameter to show.</td>
</tr>
<tr>
<td><code>*mid</code></td>
<td>String</td>
<td>Must be the static menu ID of a defined OID Menu.</td>
<td>This is used to determine the user rights for a parameter. The menu with the a staticid matching the specified <code>mid</code> 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.</td>
</tr>
</tbody>
</table>

If no `mid` is defined, the parameter is always rendered as though it is on a configuration menu with full read/write rights.
<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>List of array indices separated by commas</td>
<td>All array elements referenced must exist in the parameter value.</td>
<td>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 &quot;0&quot; or should not be provided for a non-array parameter.</td>
</tr>
<tr>
<td>Widget</td>
<td>Positive integer</td>
<td>The value must be a widget hint defined for the parameter's type</td>
<td>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.</td>
</tr>
<tr>
<td>Expand</td>
<td>true, false</td>
<td>Only applies to radio and toggle button parameters.</td>
<td>Return each radio or toggle button created by a choice constraint as a separate element.</td>
</tr>
<tr>
<td>Constrainttype</td>
<td>INT_CHOICE or eo://external-object-OID</td>
<td>Can only be applied to parameters that already use choice constraints.</td>
<td>Allows a device developer to override the choice constraint defined in the OGP Parameter Descriptor. The parameter must either contain the available choices in constraint tags inside of the param tag or an external object URL pointing to an external object that contains an integer choice constraint.</td>
</tr>
<tr>
<td>Onchange</td>
<td>ogScript String</td>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Relative</td>
<td>true, false</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

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 constraint type of INT16_CHOICE, INT32_CHOICE or STRING_CHOICE.

Syntax

```
<constraint key="choice1-key">choice1-value</constraint>
```

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Integer</td>
<td>Not required for STRING_CHOICE constraints</td>
<td>Numeric assignment of current enumerated choice.</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td></td>
<td>Text name for the current enumerated choice</td>
</tr>
</tbody>
</table>

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:

```html
<buttonbar/>
```

![Figure 84 - <buttonbar/> tag](image)

**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**

```html
<editor objectid="object-id" template="template-style" widgetroot="root-flag"/>
```

**Attributes**

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objectid</td>
<td>String</td>
<td></td>
<td>ID of the device node to insert</td>
</tr>
<tr>
<td>Template</td>
<td>summary</td>
<td></td>
<td>Inserts a summary panel for the device.</td>
</tr>
<tr>
<td>Widgetroot</td>
<td>Boolean</td>
<td></td>
<td>Everything inside of the editor must be kept together. Individual elements cannot be dragged out to other panels.</td>
</tr>
</tbody>
</table>

**Example**

The following inserts the full UI for device with id `00.0f.9b.00.00.26(Slot 0)MFC-8310`:

```html
<editor objectid="00.0f.9b.00.00.26\nSlot 0\nMFC-8310" widgetroot="true"/>
```

The following inserts a summary panel for the device:

```html
<editor objectid="00.0f.9b.00.00.26\nSlot 0\nMFC-8310" template="summary" widgetroot="true"/>
```

![Figure 85 – Summary Editor](image)
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](image)

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

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objected</td>
<td>String</td>
<td>Must be the node-ID of a node in DashBoard tree view</td>
<td>The node-id of the element in the tree to display. If the object has children, they are automatically shown under the node.</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td></td>
<td>The display name of the item.</td>
</tr>
</tbody>
</table>
Example

The node-id of a node in the Tree View may be obtained by right-clicking the node and selecting “View Connection Settings”.

![Connection Information](image)

Figure 87 – Connection Settings

The following code creates a `statuscombo` with 2 nodes:

```xml
<statuscombo grid="false" left="448" name="Favorite Cards" top="118">
  <treeElement name="Slot 3: ZTC-8399"
  objectid="172.16.9.31:5253<br>Slot 3<br>ZTC-8399"/>
  <treeElement name="Slot 5: SRA-8602"
  objectid="10.1.9.36:5253<br>Slot 5<br>SRA-8602"/>
</statuscombo>
```

The result appears in DashBoard as:

![Favorite Cards](image)

Figure 88 – statuscombo

When clicked, it expands as follows:

![Favorite Cards](image)

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:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%frame%</td>
<td>Expands to the node-id of the current frame</td>
</tr>
<tr>
<td>%device%</td>
<td>Expands to the node-id of the current device or card node</td>
</tr>
<tr>
<td>%slot%</td>
<td>Expands to the node-id of the specified slot within the current frame</td>
</tr>
<tr>
<td>%value%</td>
<td>Expands to a parameter’s value</td>
</tr>
<tr>
<td>%widget%</td>
<td>Expands to a widget’s id</td>
</tr>
<tr>
<td>%const%</td>
<td>Expands to a lookup value</td>
</tr>
<tr>
<td>%baseoid%</td>
<td>Expands to a widget’s baseOID</td>
</tr>
<tr>
<td>%fully-qualified-id%</td>
<td>Expands to the full element id hierarchy</td>
</tr>
<tr>
<td>%panel-path%</td>
<td>Expands to the path of the current CustomPanel</td>
</tr>
<tr>
<td>%app-path%</td>
<td>Expands to the DashBoard installation directory</td>
</tr>
<tr>
<td>%id%</td>
<td>Expands to the id of the current component</td>
</tr>
<tr>
<td>%eval[ogscript]%</td>
<td>Performs a regular expression expansion</td>
</tr>
</tbody>
</table>
%frame%
Expands to the node-id of the frame within the current context.

Syntax
%frame%

Example
<label name="frame node-id is %frame%"/>

```
frame node-id is 00.0f.9b.01.05.2c
```

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%"/>

```
device node-id is 00.0f.9b.01.05.2c<br>Slot 8<br>ZTC-8399
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>slot-number</td>
<td>Integer</td>
<td>Must be a valid slot number within the current frame</td>
<td>Slot number of the device whose node-id is to be returned.</td>
</tr>
</tbody>
</table>

Example
<label name="slot 2 node-id is %slot 2%"/>

```
slot 2 node-id is 00.0f.9b.01.05.2c<br>Slot 2<br>UDA-8705A
```

Figure 93 - %slot% macro
%value%

Expands to the value of a specified parameter.

**Syntax**

```
%value ['param-oid'][element]%
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>param-oid</td>
<td>String</td>
<td></td>
<td>The OID of the parameter whose value is returned</td>
</tr>
<tr>
<td>element</td>
<td>Integer</td>
<td></td>
<td>The array index to return. For non-array parameters this must be set to 0.</td>
</tr>
</tbody>
</table>

**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"/>
  </label>
</abs>
```

![Figure 94 - %value% macro](image)
%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:

```html
<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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
<th>Restrictions</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>String</td>
<td>Must be an id defined in a &lt;lookup&gt; tag</td>
<td>ID of the lookup tag.</td>
</tr>
<tr>
<td>key</td>
<td>String</td>
<td>Must be a valid key within the specified lookup.</td>
<td>Key within the lookup tag whose value will be returned.</td>
</tr>
</tbody>
</table>

**Example**

Given the following lookup:

```html
<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”.

```html
<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
<ogscript handles="onchange" oid="%baseoid%.eq" element="0">
  ogscript.debug('EQ has changed for %baseoid%:' + this.getValue());
</ogscript>
```

%f 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
%f fully-qualified-id%

Example

```xml
<abs id="abs1">
  <abs>
    <abs id="abs2">
      <label "The fully qualified ID is %fully-qualified-id%"/>
    </abs>
  </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

```xml
<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
%
%

Example
<label name="DashBoard is installed in %app-path%" />

![Figure 97 - %app-path% macro](image)

%id%

Expands to the id of the current context.

Syntax
%
%

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
%
%

Example
<label height="62" left="0" name="%eval[var text = ''; for (var i = 0; i < 10; i++) {text += i + ' ';} text.trim()]%" style="txt-align:center;" top="0" width="291">
  0123456789
</label>
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
- rostalk Object
- rostalkex Object
- robot Object
- vdp 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:
- rostalk Object
- robot Object
- vdep 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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execute</td>
<td>N/A</td>
<td>Boolean</td>
<td>Execute the multiSet. Returns true if execution was successful; otherwise false.</td>
</tr>
<tr>
<td>setAllValues</td>
<td>Object [OID], Object [] [Values]</td>
<td>N/A</td>
<td>Update all values of the parameter with the specified OID using the values from the object array.</td>
</tr>
<tr>
<td>setValue</td>
<td>Object [OID], Int [Index], Object [Value]</td>
<td>N/A</td>
<td>Update the specified index using the value object.</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertCommaSeparatedLevelsToMask</td>
<td>String [Levels], Boolean [SearchTags]</td>
<td>Long Levelmask</td>
<td>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'.</td>
</tr>
<tr>
<td>doSwitch</td>
<td>N/A</td>
<td>Boolean</td>
<td>Equivalent of calling: doSwitch(getActiveDst(), getActiveSrc(), getLevelMask());</td>
</tr>
<tr>
<td>doSwitchWithLabels</td>
<td>String [Destination], String [Source], String [Levels]</td>
<td>Boolean</td>
<td>Allows you to switch between levels by name.</td>
</tr>
<tr>
<td>getActiveDst</td>
<td>N/A</td>
<td>Int</td>
<td>Get the active dst number (0-indexed). Returns -1 if there is no active destination.</td>
</tr>
<tr>
<td>getActiveDstName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active dst (from the switchboard configuration). Returns null if there is no active destination.</td>
</tr>
<tr>
<td>getActiveIPS</td>
<td>N/A</td>
<td>String</td>
<td>Get the serial number of the active IPS</td>
</tr>
<tr>
<td>getActiveIPSName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active IPS</td>
</tr>
<tr>
<td>getActiveSrc</td>
<td>N/A</td>
<td>Int</td>
<td>Get the active src number (0-indexed). Returns -1 if there is no active source.</td>
</tr>
<tr>
<td>getActiveSrcName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active src (from the switchboard configuration). Returns null if there is no active source.</td>
</tr>
<tr>
<td>getActiveSystem</td>
<td>N/A</td>
<td>NKSystem</td>
<td>Get the currently active NKSystem.</td>
</tr>
<tr>
<td>getDstName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the destination name of the given source.</td>
</tr>
<tr>
<td>getLevelMask</td>
<td>N/A</td>
<td>Long</td>
<td>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.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------</td>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getLevelName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the level name of the given source.</td>
</tr>
<tr>
<td>getProtectStatus</td>
<td>String [Destination], String [Levels]</td>
<td>Boolean</td>
<td>Get the protect status of the destination level.</td>
</tr>
<tr>
<td>getSrcName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the source name.</td>
</tr>
<tr>
<td>getStatus</td>
<td>String [Destination], String [Level]</td>
<td>Int</td>
<td>Get the status of the given destination level.</td>
</tr>
<tr>
<td>isLevelActive</td>
<td>Int [Level Num]</td>
<td>Boolean</td>
<td>Is the current level active. Equivalent to asking: levelMask &amp; (1 &lt;&lt; levelNum) != 0;</td>
</tr>
<tr>
<td>isMCFlag</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is the Machine Control flag set.</td>
</tr>
<tr>
<td>isProtected</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is the active destination protected.</td>
</tr>
<tr>
<td>isProtected</td>
<td>Int [Destination], Long [Levels]</td>
<td>Boolean</td>
<td>True if the given destination is protected; otherwise false.</td>
</tr>
<tr>
<td>isProtectedByMe</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is the active destination protected by this virtual panel.</td>
</tr>
<tr>
<td>isSrcActive</td>
<td>Int [Src]</td>
<td>Boolean</td>
<td>Is the given source active on the active destination any level.</td>
</tr>
<tr>
<td>isSrcActive</td>
<td>Int [Dst], Int [Src], Long [Levels]</td>
<td>Boolean</td>
<td>Is the given source active on the given destination on the given level mask.</td>
</tr>
<tr>
<td>isVirtual</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is virtual routing in use (for switch commands and status requests).</td>
</tr>
<tr>
<td>setActiveDst</td>
<td>Int [Dst]</td>
<td>N/A</td>
<td>Set the active destination (0-indexed).</td>
</tr>
<tr>
<td>setActiveIPS</td>
<td>String [Serial]</td>
<td>Boolean</td>
<td>Set the IPS with the given serial number as the active IPS to receive commands and send status. Deactivate any currently active IPS.</td>
</tr>
<tr>
<td>setActiveSrc</td>
<td>Int [Src]</td>
<td>N/A</td>
<td>Set the active source (0-indexed).</td>
</tr>
<tr>
<td>setLevelActive</td>
<td>Int [Level Num], Boolean [Active]</td>
<td>Boolean</td>
<td>Set the given level as active.</td>
</tr>
<tr>
<td>setLevelMask</td>
<td>Long [Level Mask]</td>
<td>N/A</td>
<td>Set the complete level mask bitfield.</td>
</tr>
<tr>
<td>setMCFlag</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Set the Machine Control flag to true or false.</td>
</tr>
<tr>
<td>setProtected</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Request the router to protect the active destination.</td>
</tr>
<tr>
<td>setVirtual</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Set virtual routing on/off for switch commands and status requests.</td>
</tr>
</tbody>
</table>
**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>Function reference</td>
<td>Yes</td>
<td>Reference to the function to be executed. Can also be an anonymous function.</td>
</tr>
<tr>
<td>delay</td>
<td>Long</td>
<td>No</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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:

---

**asyncFTPLListFiles**

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()`
ogscript.asyncFTPListFiles(host, port, username, password, path, callback);

Syntax

ogscript.asyncFTPListFiles(host, port, username, password, path, callback);

ogscript.asyncFTPListFiles(host, port, username, password, path, fileName, callback);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host address</td>
</tr>
<tr>
<td>Port</td>
<td>Int</td>
<td>Yes</td>
<td>Host port</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Yes</td>
<td>Login username</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>Yes</td>
<td>Login password</td>
</tr>
<tr>
<td>Path</td>
<td>String</td>
<td>Yes</td>
<td>Source path</td>
</tr>
<tr>
<td>fileName</td>
<td>String</td>
<td>Optional</td>
<td>Source file name, can contain the &quot;.*&quot; wildcard.</td>
</tr>
<tr>
<td>callback</td>
<td>Function reference</td>
<td>Yes</td>
<td>Callback function. Invoked after FTPListFiles is complete. Callback is passed success, list of files, and exception</td>
</tr>
</tbody>
</table>

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:

```javascript
function outputResults(success, files, exception)
{
  if (!success)
  {
    ogscript.debug("NO SUCCESS");
    return;
  }
  else if (files != null)
  {
    /*
    * files[i].getName()       // returns java.util.Calendar
    * files[i].getTimestamp    // 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>String</td>
<td>Yes</td>
<td>Http url</td>
</tr>
<tr>
<td>Method</td>
<td>String</td>
<td>Yes</td>
<td>The method for the URL request, one of: GET POST HEAD OPTIONS PUT DELETE TRACE are legal, subject to protocol restrictions.</td>
</tr>
<tr>
<td>Content_Type</td>
<td>String</td>
<td>Yes</td>
<td>The content type of the request.</td>
</tr>
<tr>
<td>Data</td>
<td>Object</td>
<td>Yes</td>
<td>Data can be a string, byte array, XML, or JSON object</td>
</tr>
<tr>
<td>Callback</td>
<td>Function reference</td>
<td>Yes</td>
<td>Function to call after the request completes.</td>
</tr>
<tr>
<td>Include_Response_Code</td>
<td>Boolean</td>
<td>No</td>
<td>True to include response code; otherwise false.</td>
</tr>
</tbody>
</table>

Returns
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**

```javascript
ogscript.asyncPost (URL, HTTP Post Data, Callback Function);
ogscript.asyncPost (URL, HTTP Post Data, Callback Function, Include Response);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>String</td>
<td>Yes</td>
<td>URL to send a post.</td>
</tr>
<tr>
<td>HTTP Post Data</td>
<td>String</td>
<td>Yes</td>
<td>Post to send to the specified URL.</td>
</tr>
<tr>
<td>Callback Function</td>
<td>Function</td>
<td>Yes</td>
<td>Function to call after the post completes.</td>
</tr>
<tr>
<td>Include Response</td>
<td>Boolean</td>
<td>No</td>
<td>If true, result is a JSON Object</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>{ responseCode = HTTP RESPONSE CODE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contentType = HTTP MIME TYPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>url = URL Requested</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bytes= BYTES RECEIVED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>}</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Otherwise, it is content fetched over HTTP parsed as though it's a string (as before).</td>
</tr>
</tbody>
</table>

**Returns**

N/A

**Example**

Coming soon.

closePanel

Closes the DashBoard panel that the command was called from.

**Syntax**

```javascript
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>color</td>
<td>Int</td>
<td>Yes</td>
<td>Integer representation of RGB color (in decimal)</td>
</tr>
<tr>
<td>color</td>
<td>String</td>
<td>Yes</td>
<td>String representation of RGB color (in hex)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>String</td>
<td>Yes</td>
<td>Text to be copied to clipboard.</td>
</tr>
</tbody>
</table>

Returns
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>String</td>
<td>Yes</td>
<td>Desired ID for new thread</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Int</td>
<td>Yes</td>
<td>Length of the byte array (Must be greater than or equal to zero)</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File Object</td>
<td>Object</td>
<td>Yes</td>
<td>Destination file object</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID for new listener</td>
</tr>
<tr>
<td>Listener Settings</td>
<td>Object</td>
<td>Yes</td>
<td>Settings for new listener</td>
</tr>
<tr>
<td>Listener Task</td>
<td>Function reference</td>
<td>Yes</td>
<td>Task for new listener</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>Component ID to focus</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host address</td>
</tr>
<tr>
<td>Port</td>
<td>Int</td>
<td>Yes</td>
<td>Host port</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Yes</td>
<td>Login username</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>Yes</td>
<td>Login password</td>
</tr>
<tr>
<td>Destination Path</td>
<td>String</td>
<td>Yes</td>
<td>Data destination path</td>
</tr>
<tr>
<td>Destination Name</td>
<td>String</td>
<td>Yes</td>
<td>Data destination name</td>
</tr>
<tr>
<td>Binary</td>
<td>Boolean</td>
<td>Yes</td>
<td>True if data is binary (.jpg, .mp3), false if data is ascii (.txt, .html).</td>
</tr>
<tr>
<td>Data</td>
<td>Object</td>
<td>Yes</td>
<td>Data to be transferred</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host address</td>
</tr>
<tr>
<td>Port</td>
<td>Int</td>
<td>Yes</td>
<td>Host port</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Yes</td>
<td>Login username</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>Yes</td>
<td>Login password</td>
</tr>
<tr>
<td>Source Path</td>
<td>String</td>
<td>Yes</td>
<td>Source path</td>
</tr>
<tr>
<td>Source Name</td>
<td>String</td>
<td>Yes</td>
<td>Source name</td>
</tr>
<tr>
<td>Binary</td>
<td>Boolean</td>
<td>Yes</td>
<td>True if data is binary (.jpg, .mp3), false if data is ascii (.txt, .html).</td>
</tr>
<tr>
<td>Destination File</td>
<td>Object</td>
<td>Yes</td>
<td>Destination file object</td>
</tr>
</tbody>
</table>

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);
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host address</td>
</tr>
<tr>
<td>Port</td>
<td>Int</td>
<td>Yes</td>
<td>Host port</td>
</tr>
<tr>
<td>Username</td>
<td>String</td>
<td>Yes</td>
<td>Login username</td>
</tr>
<tr>
<td>Password</td>
<td>String</td>
<td>Yes</td>
<td>Login password</td>
</tr>
<tr>
<td>Source Path</td>
<td>String</td>
<td>Yes</td>
<td>Source path</td>
</tr>
<tr>
<td>File Name</td>
<td>String</td>
<td>No</td>
<td>Source file name</td>
</tr>
</tbody>
</table>

**Returns**

Returns an array of FTPFile objects.

FTPFile class is used to represent information about files stored on an FTP server.

**Example 1**

```javascript
// 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**

```javascript
ogscript.getApplicationPath();
```

**Parameters**

N/A

**Returns**

Returns a String representation of the path to the DashBoard installation location.

**Example 1**

```javascript
// Get and store dashboard installation loca
var dashboardLocation = ogscript.getApplicationPath();
```

**getAsyncExecById**

Finds and returns an asynchronous thread with a specified ID.

**Syntax**

```javascript
ogscript.getAsyncExecById(thread id);
```

**Parameters**
### Parameter

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of desired thread.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Browser ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of browser to look for.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filePath</td>
<td>String</td>
<td>Yes</td>
<td>Path to desired file</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filePath</td>
<td>String</td>
<td>Yes</td>
<td>Path to desired file</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Image ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of desired image</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>String</td>
<td>Yes</td>
<td>Relative path</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HSL Float Array</td>
<td>Float32_Array</td>
<td>Yes</td>
<td>Float array – first element is hue, second element is saturation, third element is lightness.</td>
</tr>
</tbody>
</table>

#### 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
### 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the timer to create and register in the ContextTimerManager.</td>
</tr>
<tr>
<td>Repeat</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — repeat the timer using the specified Delay and Repeat Delay; false — only run the timer once, do not repeat the timer.</td>
</tr>
<tr>
<td>Delay</td>
<td>Long</td>
<td>Yes</td>
<td>Number of milliseconds to wait before starting the timer.</td>
</tr>
<tr>
<td>Repeat Delay</td>
<td>Long</td>
<td>Yes</td>
<td>How frequently the associated function runs, in milliseconds.</td>
</tr>
<tr>
<td>Execute in Timer</td>
<td>Boolean</td>
<td>No</td>
<td>If true, task will execute in timer thread</td>
</tr>
<tr>
<td>Task</td>
<td>Function</td>
<td>Yes</td>
<td>ogScript function to run when the timer fires.</td>
</tr>
</tbody>
</table>

#### Returns

N/A

### Returns

Returns either string data or a JSON object.

#### Example 1

Coming soon.
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.

```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);
```

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON native object</td>
<td>NativeObject</td>
<td>Yes</td>
<td>The JSON to be converted to a String</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>String</td>
<td>Yes</td>
<td>The function to execute, including its parameters (if any).</td>
</tr>
<tr>
<td>Trigger ID</td>
<td>String</td>
<td>Yes</td>
<td>String that triggers the specified function to execute.</td>
</tr>
<tr>
<td>Trigger Name</td>
<td>String</td>
<td>Yes</td>
<td>Shows on the button in the web UI.</td>
</tr>
</tbody>
</table>

**Returns**

Returns an object that contains one function named `close`. When executed, `close` removes the function.

**Example**

```javascript
// 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:

```javascript
ogscript.function name(parameters);
```

For example:

```javascript
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addRemoteTrigger</td>
<td>String [Trigger] String [Function]</td>
<td>Returns an object that contains one function named <code>close</code>. When executed, <code>close</code> removes the function.</td>
<td></td>
</tr>
<tr>
<td>appendXML</td>
<td>String [Container ID] String [XML snippet]</td>
<td>N/A</td>
<td>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 <code>&lt;abs/&gt;</code> containers.</td>
</tr>
<tr>
<td>asyncExec</td>
<td>function</td>
<td>N/A</td>
<td>Executes a function outside of the UI current thread.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------</td>
</tr>
<tr>
<td>asyncFTP</td>
<td>post port username password destPath destName binary sourceFilePath callback</td>
<td>N/A</td>
<td>Sends a file to an FTP server.</td>
</tr>
<tr>
<td>asyncFTPGet</td>
<td>host port username password srcPath srcName binary destFilePath or null callback</td>
<td>N/A</td>
<td>Retrieves a file from FTP server.</td>
</tr>
<tr>
<td>asyncPost</td>
<td>String [URL] String [HTTP Post Data] Function [Callback Function] Boolean [include response]</td>
<td>N/A</td>
<td>Send an asynchronous post to the given URL.</td>
</tr>
<tr>
<td>cancelTimer</td>
<td>Timer ID</td>
<td>N/A</td>
<td>Cancel, stop and clean-up, a timer with the given ID.</td>
</tr>
<tr>
<td>copyByteArray</td>
<td>src offset length</td>
<td>byte array</td>
<td>Creates a full or partial copy of a byte array.</td>
</tr>
<tr>
<td>createByteArray</td>
<td>length</td>
<td>an empty byte array</td>
<td>Creates an empty byte array of a specified size.</td>
</tr>
<tr>
<td>createFileInput</td>
<td>String [File path]</td>
<td>FileInputStream (like MessageParser but with getSize(), close(), and isClosed())</td>
<td>Access a file as a byte array with the same capabilities as MessageParser to read raw bytes</td>
</tr>
<tr>
<td>createFileOutput</td>
<td>String [File path] Boolean [appendToExistingfile]</td>
<td>FileOutputStream, which is same as MessageBuilder with added functions for clean() (overwrite file), close(), getSize(), flush(), and isClosed()</td>
<td>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.</td>
</tr>
<tr>
<td>createMessageBuilder</td>
<td>N/A</td>
<td>Returns a MessageBuilder object used to build byte arrays (generally for creating network messages).</td>
<td>Creates a message builder, which enables you to construct a message.</td>
</tr>
<tr>
<td>createMessageParser</td>
<td>messageBytes</td>
<td>Returns a MessageParser object (generally used to parse the various pieces of messages received over the network).</td>
<td>Creates a message parser, which enables you to parse a message.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>debug</td>
<td>String [Message]</td>
<td>N/A</td>
<td>Write a string to the openGear Debug Information View.</td>
</tr>
<tr>
<td></td>
<td>Boolean [global]</td>
<td>N/A</td>
<td>Sends optional [state] data string, which can be read by the script.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When [global]' value is 'true', applies to all open panels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>When [global] is 'false', applies only to the current active panel.</td>
</tr>
<tr>
<td>getAllById</td>
<td>String [Object ID]</td>
<td>Object []</td>
<td>Get all Objects accessible in the current context that have the associated ID.</td>
</tr>
<tr>
<td>getAttribute</td>
<td>String [Attribute ID]</td>
<td>Object</td>
<td>Get an attribute registered in the context with the given ID.</td>
</tr>
<tr>
<td>getBuild</td>
<td>N/A</td>
<td>DashBoard version number (same value that appears in Help&gt;About DashBoard)</td>
<td>Gets the version of DashBoard running the panel.</td>
</tr>
<tr>
<td>getComponentsById</td>
<td>String [Object ID]</td>
<td>Component []</td>
<td>Get all Java Swing components accessible in the current context that have the associated ID.</td>
</tr>
<tr>
<td>getCurrentUser</td>
<td>N/A</td>
<td>String</td>
<td>Returns the username of the current DashBoard user.</td>
</tr>
<tr>
<td>getIncludeById</td>
<td>String [Include ID]</td>
<td>IncludeReloadableContainer</td>
<td>Returns the first include with the given ID.</td>
</tr>
<tr>
<td>getListenerById</td>
<td>ID</td>
<td>getListenerById returns an object representing the listener.</td>
<td>Starts or stops a listener. Can also check whether a listener is started.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>This object has three public methods you can call: start(), stop(), and isStarted().</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>The return depends on which of the three methods is used:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the start() method is used, return is true if the listener started successfully; otherwise false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the stop() method is used, return is true if the listener stopped successfully; otherwise false.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>If the isStarted() method is used, return is true if the listener is started; otherwise false.</td>
</tr>
<tr>
<td>getModificationDate</td>
<td>String [File Path]</td>
<td>Returns the time the specified file was last modified, in Unix Epoch time (also known as POSIX time), as a LONG value.</td>
<td>Retrieves the time the specified file was last modified.</td>
</tr>
<tr>
<td>getObject</td>
<td>String [Key]</td>
<td>String</td>
<td>Retrieves stored object</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-------------------------------------</td>
<td>------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getPosition</td>
<td>String [ID]</td>
<td>JAVA point object with point.x and point.y available.</td>
<td>Retrieves the horizontal (x) and vertical (y) position of the object, in pixels.</td>
</tr>
<tr>
<td>getPrivateString</td>
<td>String [Lookup ID]</td>
<td>String</td>
<td>Get a string defined in the lookup table with the specified lookup ID.</td>
</tr>
<tr>
<td>getScopedAttribute</td>
<td>String [Scope Name] String [Attribute ID]</td>
<td>Object</td>
<td>Get an attribute in the named scope that has the given ID. Scopes are often internally defined by DashBoard.</td>
</tr>
<tr>
<td>getSize</td>
<td>String [ID]</td>
<td>Dimension object with d.width and d.height available</td>
<td>Retrieves the width and height of the specified panel object.</td>
</tr>
<tr>
<td>getString</td>
<td>String [Key]</td>
<td>String</td>
<td>Get a string defined in the global lookup table.</td>
</tr>
<tr>
<td>getTimerManager</td>
<td>N/A</td>
<td>ContextTimerManager</td>
<td>Get the timer manager for the context to access timers and perform operations on selected timers. This function includes several methods.</td>
</tr>
<tr>
<td>hide</td>
<td>String [ID]</td>
<td>N/A</td>
<td>Hide the popup with the specified ID.</td>
</tr>
<tr>
<td>installTimer</td>
<td>String [Timer ID] Boolean [Repeat] Long [Delay] Long [Repeat Rate] Function [Task]</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>isTimerRunning</td>
<td>String [Timer ID]</td>
<td>Boolean</td>
<td>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.</td>
</tr>
<tr>
<td>parseXML</td>
<td>String</td>
<td>org.w3c.dom.Document</td>
<td>Parse and return an XML document using the org.w3c.dom.Document API.</td>
</tr>
<tr>
<td>putObject</td>
<td>String [Key] String [Value]</td>
<td>N/A</td>
<td>Defines a stored object.</td>
</tr>
<tr>
<td>putPrivateString</td>
<td>String [LookupID] String [Key] String [Value]</td>
<td>N/A</td>
<td>Add or replace a string in a private lookup table.</td>
</tr>
<tr>
<td>putString</td>
<td>String [key] String [value]</td>
<td>N/A</td>
<td>Add or replace a string in the global lookup table.</td>
</tr>
<tr>
<td>reload</td>
<td>String [ID]</td>
<td>Null, if null is provided as the ID.</td>
<td>Rebuild the UI element with the given ID. If no ID is provided, rebuilds entire document.</td>
</tr>
<tr>
<td>rename</td>
<td>String [ID] String [Name]</td>
<td>N/A</td>
<td>Modify the text for a tab name, button, or label with the specified ID.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>reposition</td>
<td>String [ID]</td>
<td>N/A</td>
<td>Moves object to specified XY pixel location</td>
</tr>
</tbody>
</table>
| 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Function reference</td>
<td>Yes</td>
<td>Function to be added on close.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>String</td>
<td>Yes</td>
<td>String that triggers the specified function to execute.</td>
</tr>
<tr>
<td>Function</td>
<td>String</td>
<td>Yes</td>
<td>The function to execute, including its parameters (if any).</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>container ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the container to append to. Valid only in &lt;abs/&gt; containers.</td>
</tr>
<tr>
<td>XML snippet</td>
<td>String (XML object)</td>
<td>Yes</td>
<td>XML code to append</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>function</td>
<td>Function</td>
<td>Yes</td>
<td>Reference to the function to be executed. Can also be an anonymous function.</td>
</tr>
</tbody>
</table>

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:

```html
<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:

```html
<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.
      }
    }
  </meta>
</api>
<params>
```
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 ogscript handler to monitor changes to the attribute to show progress.

Syntax

ogscript.asyncFTP (host, port, username, password, destPath, destName, binary, sourceFilePath, callback);

Parameters
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>String</td>
<td>Yes</td>
<td>The host name of the destination computer.</td>
</tr>
<tr>
<td>port</td>
<td>Integer</td>
<td>Yes</td>
<td>The port number to which the data is to be sent.</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>Yes</td>
<td>The username required to log onto the destination computer.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Yes</td>
<td>The password required to log onto the destination computer.</td>
</tr>
<tr>
<td>destPath</td>
<td>String</td>
<td>No</td>
<td>The directory path where the data is to be saved on the destination computer.</td>
</tr>
<tr>
<td>destName</td>
<td>String</td>
<td>No</td>
<td>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.</td>
</tr>
<tr>
<td>binary</td>
<td>Boolean</td>
<td>Yes</td>
<td>Specifies the transfer mode. When true, binary transfer is used. When false, ASCII transfer is used.</td>
</tr>
<tr>
<td>sourceFilePath</td>
<td>String</td>
<td>Yes</td>
<td>The directory path to the source file. The path can be absolute or relative.</td>
</tr>
<tr>
<td>callback</td>
<td>function reference</td>
<td>No</td>
<td>The callback is called when the operation is complete, whether or not the operation is successful.</td>
</tr>
</tbody>
</table>

**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.

```plaintext
<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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>host</td>
<td>String</td>
<td>Yes</td>
<td>The host name of the source computer, from which the file is to be retrieved</td>
</tr>
<tr>
<td>port</td>
<td>Integer</td>
<td>Yes</td>
<td>The port number required to access the source computer.</td>
</tr>
<tr>
<td>username</td>
<td>String</td>
<td>Yes</td>
<td>The username required to log onto the source computer.</td>
</tr>
<tr>
<td>password</td>
<td>String</td>
<td>Yes</td>
<td>The password required to log onto the source computer.</td>
</tr>
<tr>
<td>srcPath</td>
<td>String</td>
<td>No</td>
<td>The directory path where the source file is located.</td>
</tr>
<tr>
<td>srcName</td>
<td>String</td>
<td>Yes</td>
<td>The name of the file to be retrieved.</td>
</tr>
<tr>
<td>binary</td>
<td>Boolean</td>
<td>Yes</td>
<td>Specifies the transfer mode. When true, binary transfer is used. When false, ASCII transfer is used.</td>
</tr>
<tr>
<td>destFilePath or null</td>
<td>String</td>
<td>No</td>
<td>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.</td>
</tr>
<tr>
<td>callback</td>
<td>function reference</td>
<td>No</td>
<td>The callback is called when the operation is complete, whether or not the operation is successful.</td>
</tr>
</tbody>
</table>

Returns
N/A

Example
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

```javascript
ogscript.asyncPost (URL, HTTP Post Data, Callback Function, Include Response);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>URL</td>
<td>String</td>
<td>Yes</td>
<td>URL to send a post.</td>
</tr>
<tr>
<td>HTTP Post Data</td>
<td>String</td>
<td>Yes</td>
<td>Post to send to the specified URL.</td>
</tr>
<tr>
<td>Callback Function</td>
<td>Function</td>
<td>Yes</td>
<td>Function to call after the post completes.</td>
</tr>
<tr>
<td>Include Response</td>
<td>Boolean</td>
<td>No</td>
<td>If true, result is a JSON Object {</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>responseCode = HTTP RESPONSE CODE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>contentType = HTTP MIME TYPE,</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>url = URL Requested</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>bytes= BYTES RECEIVED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>} If true, result is a JSON Object</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Otherwise, it is content fetched over HTTP parsed as though it's a string (as before).</td>
</tr>
</tbody>
</table>

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

```javascript
ogscript.cancelTimer (Timer ID);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the timer to stop and clean up.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>src</td>
<td>byte array</td>
<td>Yes</td>
<td>The byte array to be copied.</td>
</tr>
<tr>
<td>offset</td>
<td>Integer</td>
<td>Yes</td>
<td>Index of the first byte to be copied. Use 0 for the start of the array.</td>
</tr>
<tr>
<td>length</td>
<td>Integer</td>
<td>Yes</td>
<td>The number of bytes to copy. Tip: To copy the entire array, use src.length.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>Integer</td>
<td>Yes</td>
<td>The size of the new array, in bytes.</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File path</td>
<td>String</td>
<td>Yes</td>
<td>Path of the file to open (can be relative to the panel)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File path</td>
<td>String</td>
<td>Yes</td>
<td>File path of the file to be created or appended.</td>
</tr>
<tr>
<td>appendToExistingfile</td>
<td>Boolean</td>
<td>Yes</td>
<td>When true, data is appended to existing file. When false, a new file is created.</td>
</tr>
</tbody>
</table>

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.

```javascript
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageBytes</td>
<td>byte array</td>
<td>Yes</td>
<td>The source byte array.</td>
</tr>
</tbody>
</table>

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.

```javascript
var parsedMessage = ogscript.createMessageParser(messageArray);
ogscript.debug(parsedMessage.readBoolean());
ogscript.debug(parsedMessage.readByte());
ogscript.debug(parsedMessage.readUnsignedByte());
```
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>String</td>
<td>Yes</td>
<td>Message to display in the openGear Debug Information View.</td>
</tr>
</tbody>
</table>

Returns

N/A

Example 1

`ogscript.debug ('This is a message');`

Example 2

```javascript
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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger</td>
<td>String</td>
<td>Yes</td>
<td>GPI Trigger message.</td>
</tr>
<tr>
<td>State</td>
<td>String</td>
<td>No</td>
<td>Sends optional data string, which can be read by the script.</td>
</tr>
<tr>
<td>Parameter</td>
<td>Type</td>
<td>Required</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>--------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Global</td>
<td>Boolean</td>
<td>Yes</td>
<td>When true, applies to all open panels. When false, applies only the panel initiating the trigger.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the objects in the current context to get.</td>
</tr>
</tbody>
</table>

**Returns**

Object []

**Example**

Coming soon.

### getAttribute

Get an attribute registered in the context with the given ID.

**Syntax**

`ogscript.getAttribute (Attribute ID);`

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute ID</td>
<td>String</td>
<td>Yes</td>
<td>ID from which to get a registered in context attribute.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object ID</td>
<td>String</td>
<td>Yes</td>
<td>ID from which to get all Java Swing components accessible in the current context.</td>
</tr>
</tbody>
</table>

**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:

```html
<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:

```javascript
//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

```javascript
ogscript.getIncludeById (Include ID);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Include ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the include to find.</td>
</tr>
</tbody>
</table>

Returns

IncludeReloadableContainer

Example

Coming soon.

getListenerById

Starts or stops a listener. Can also check whether a listener is started.

Syntax

```javascript
ogscript.getListenerById (ID);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the listener.</td>
</tr>
</tbody>
</table>

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

```javascript
var myListener = ogscript.getListenerById (myId); myListener.start ();
myListener.stop ();
myListener.isStarted ();
```

**getModificationDate**

Retrieves the time the specified file was last modified.

**Syntax**

```javascript
ogscript.getModificationDate (file path);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File path</td>
<td>String</td>
<td>Yes</td>
<td>Path to the file.</td>
</tr>
</tbody>
</table>

**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 can 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**

```javascript
ogscript.getObject(Key);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>String</td>
<td>Yes</td>
<td>The name used to reference what is being stored.</td>
</tr>
</tbody>
</table>

**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`.

```javascript
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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>The ID of the panel object.</td>
</tr>
</tbody>
</table>

**Returns**

JAVA point object containing public variables `x` and `y`, populated with values for the horizontal (x) 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.

```xml
<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() < size.width - 10 && event.y < 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>
</meta>
</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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the string to find in the specified lookup table.</td>
</tr>
<tr>
<td>Key</td>
<td>String</td>
<td>Yes</td>
<td>Private lookup table in which to find the specified string.</td>
</tr>
</tbody>
</table>

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:

```xml
<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:

```javascript
//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**

```javascript
ogscript.getScopedAttribute (Scope Name, Attribute ID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope Name</td>
<td>String</td>
<td>Yes</td>
<td>Name of the scope in which to get and attribute.</td>
</tr>
<tr>
<td>Attribute ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the attribute to get in the named scope.</td>
</tr>
</tbody>
</table>

**Returns**

Object

**Example**

Coming soon.

**getSize**

Retrieves the width and height of the specified panel object, in pixels.

**Syntax**

```javascript
ogscript.getSize (ID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the panel object.</td>
</tr>
</tbody>
</table>

**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.

```html
<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() < size.width - 10 && event.y < 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**

```javascript
ogscript.getString (Key);
```
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>String</td>
<td>Yes</td>
<td>Private lookup table from which to get string.</td>
</tr>
</tbody>
</table>

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:

```xml
<lookup>
  <entry key="Tom">television</entry>
</lookup>
```

The script to read an entry from the lookup table is as follows:

```javascript
//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.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameter Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>isRunning()</td>
<td>N/A</td>
<td>Checks whether the time is running.</td>
</tr>
<tr>
<td>startTimer(Boolean reset)</td>
<td>Yes true or false</td>
<td>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.</td>
</tr>
<tr>
<td>stopTimer(Boolean reset)</td>
<td>Yes true or false</td>
<td>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.</td>
</tr>
<tr>
<td>resetTimer()</td>
<td>N/A</td>
<td>Resets the timer to the start time.</td>
</tr>
<tr>
<td>setStart(Long valueInMilliseconds)</td>
<td>Yes Milliseconds (Long)</td>
<td>Sets the start time of the timer.</td>
</tr>
<tr>
<td>Method</td>
<td>Parameter Required</td>
<td>Required</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------</td>
<td>----------</td>
</tr>
<tr>
<td>setStop(Long valueInMilliseconds)</td>
<td>Yes</td>
<td>Milliseconds (Long)</td>
</tr>
<tr>
<td>setTime(Long valueInMilliseconds)</td>
<td>Yes</td>
<td>Milliseconds (Long)</td>
</tr>
<tr>
<td>getStart()</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>getStop()</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>getCurrent()</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>incrementTime(Long difference)</td>
<td>Yes</td>
<td>Milliseconds (Long)</td>
</tr>
<tr>
<td>setPattern(String dateTimePattern)</td>
<td>Yes</td>
<td>Time format definition</td>
</tr>
</tbody>
</table>

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

```javascript
// 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

```javascript
// Get the start time of a timer named 'selftimer'
var startTime =
ogscript.getTimerManager().getTimer('selftimer').getStart();
```

Example 9 — getTimerManager function using getStop method

```javascript
// Get the stop time of a timer named 'selftimer'
var stopTime =
ogscript.getTimerManager().getTimer('selftimer').getStop();
```

Example 10 — getTimerManager function using getCurrent method

```javascript
// Get the current time of a timer named 'selftimer'
var currentTime =
ogscript.getTimerManager().getTimer('selftimer').getCurrent();
```

Example 11 — getTimerManager function using incrementTime method

```javascript
// Increase the current time of a timer named 'selftimer' by 30 seconds
ogscript.getTimerManager().getTimer('selftimer').incrementTime(30000);
```

```javascript
// 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).

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date or Time Component</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>Day</td>
<td>Number</td>
<td>189</td>
</tr>
<tr>
<td>H</td>
<td>Hour of the day (0-23)</td>
<td>Number</td>
<td>8</td>
</tr>
<tr>
<td>m</td>
<td>Minute of the hour</td>
<td>Number</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>Second of the minute</td>
<td>Number</td>
<td>55</td>
</tr>
<tr>
<td>S</td>
<td>Millisecond</td>
<td>Number</td>
<td>768</td>
</tr>
<tr>
<td>G</td>
<td>Era designator</td>
<td>Text</td>
<td>AD</td>
</tr>
<tr>
<td>Y</td>
<td>Year</td>
<td>Number</td>
<td>1969; 69</td>
</tr>
<tr>
<td>M</td>
<td>Month of the year</td>
<td>Text or number</td>
<td>September; Sep; 09</td>
</tr>
<tr>
<td>w</td>
<td>Week of the year</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>W</td>
<td>Week of the month</td>
<td>Number</td>
<td>3</td>
</tr>
<tr>
<td>d</td>
<td>Day of the month</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>F</td>
<td>Day of the week in the month</td>
<td>Number</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>Day of the week</td>
<td>Text</td>
<td>Friday; Fri</td>
</tr>
<tr>
<td>Letter</td>
<td>Date or Time Component</td>
<td>Presentation</td>
<td>Examples</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------</td>
<td>--------------</td>
<td>----------</td>
</tr>
<tr>
<td>k</td>
<td>Hour of the day (1-24)</td>
<td>Number</td>
<td>22</td>
</tr>
<tr>
<td>K</td>
<td>Hour in AM/PM (0-11)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>Hour in AM/PM (1-12)</td>
<td>Number</td>
<td>10</td>
</tr>
<tr>
<td>a</td>
<td>AM/PM marker</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>z</td>
<td>Time zone</td>
<td>General Time Zone</td>
<td>Pacific Standard Time, PST,</td>
</tr>
<tr>
<td>Z</td>
<td>Time zone</td>
<td>RFC 822 time zone</td>
<td>-0800</td>
</tr>
</tbody>
</table>

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.

```javascript
//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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Popup ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the popup to hide.</td>
</tr>
</tbody>
</table>

**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="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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the timer to create and register in the ContextTimerManager.</td>
</tr>
<tr>
<td>Repeat</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — repeat the timer using the specified Delay and Repeat Delay. false — only run the timer once, do not repeat the timer.</td>
</tr>
<tr>
<td>Delay</td>
<td>Long</td>
<td>Yes</td>
<td>Number of milliseconds to wait before starting the timer.</td>
</tr>
<tr>
<td>Repeat Delay</td>
<td>Long</td>
<td>Yes</td>
<td>How frequently the associated function runs, in milliseconds.</td>
</tr>
<tr>
<td>Task</td>
<td>Function</td>
<td>Yes</td>
<td>ogScript function to run when the timer fires.</td>
</tr>
</tbody>
</table>

**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"/>
<br><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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer ID</td>
<td>String</td>
<td>Yes</td>
<td>true — a timer with the given ID exits and is in the “running” state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>false — a timer with the given ID does not exist or is not in the “running” state.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Document</td>
<td>String</td>
<td>Yes</td>
<td>XML document to parse.</td>
</tr>
</tbody>
</table>

**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](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.

```javascript
function myFunc(pageContent)
{
  var xmlPageContent = '<?xml version="1.0" encoding="UTF-8"?>
' + pageContent;
  var document = ogscript.parseXML(xmlPageContent); var nodeList = ogscript.runXPath('/response/sports/sportsItem/leagues/leaguesItem/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=ksjdur7eujru47fkbos85kg', 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 can reference it whenever 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

```javascript
ogscript.putObject(Key, Value);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>String</td>
<td>Yes</td>
<td>The name of the object in which the data is being stored.</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Yes</td>
<td>The value to be stored.</td>
</tr>
</tbody>
</table>

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`.

```javascript
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**

```javascript
ogscript.putPrivateString (Lookup ID, Key, Value);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the string to create or replace in the specified lookup table.</td>
</tr>
<tr>
<td>Key</td>
<td>String</td>
<td>Yes</td>
<td>Private lookup table in which to create or replace the specified string.</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Yes</td>
<td>New value for the specified string.</td>
</tr>
</tbody>
</table>

**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:

```xml
<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:

```javascript
//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**

```javascript
ogscript.putString (Lookup ID, Value);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the string to create or replace in the global lookup table.</td>
</tr>
<tr>
<td>Value</td>
<td>String</td>
<td>Yes</td>
<td>New value for the specified string.</td>
</tr>
</tbody>
</table>

**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:

```xml
<lookup>
  <entry key="Tom">television</entry>
</lookup>
```

The script to replace an entry in the lookup table is as follows:

```javascript
//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**

```javascript
ogscript.reload (User Interface ID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the user interface element to rebuild.</td>
</tr>
</tbody>
</table>

**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.

```javascript
// 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.

![Component ID in the Edit Component dialog box](image)

Figure 2.1 Component ID in the Edit Component dialog box

Syntax

`ogscript.rename (Component ID, Name);`

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the user interface component to rename.</td>
</tr>
<tr>
<td>Name</td>
<td>String</td>
<td>Yes</td>
<td>New text to display on the screen for the specified user interface component.</td>
</tr>
</tbody>
</table>

Returns

N/A

Example 1

```javascript
// Set the item with ID='Seat 5' to have the text 'Mika Andersen'
ogscript.rename ('Seat 5','Mika Andersen');
```

Example 2

```javascript
// 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the component you want to reposition</td>
</tr>
<tr>
<td>x position</td>
<td>Integer</td>
<td>Yes</td>
<td>Number of pixels from the left</td>
</tr>
<tr>
<td>y position</td>
<td>Integer</td>
<td>Yes</td>
<td>Number of pixels from the right</td>
</tr>
</tbody>
</table>

Returns

N/A

Example

In this example, the task associated with the “Top Left” button uses the ogscript.reposition function to reposition a label.

```html
<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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of the component you want to reposition</td>
</tr>
<tr>
<td>x percent</td>
<td>Integer</td>
<td>Yes</td>
<td>Distance from the left, as a percentage of container width</td>
</tr>
<tr>
<td>y percent</td>
<td>Integer</td>
<td>Yes</td>
<td>Distance from the top, as a percentage of container height</td>
</tr>
<tr>
<td>center x</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — Shows the full width of the object. false — Crops the object if it extends beyond the horizontal boundaries of the container.</td>
</tr>
<tr>
<td>center y</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — Shows the full height of the object. false — Crops the object if it extends</td>
</tr>
</tbody>
</table>

Returns

N/A

Example

In this example, the task associated with the “Top Left” button uses the ogscript.repositionByPercent function to reposition a label.

```html
<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.repositionByPercent('myLabel', 0.5, 0.5, true, true);
  </task>
</button>
```
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>beyond the vertical boundaries of the container.</td>
</tr>
</tbody>
</table>

**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.

```xml
<label height="41" id="myLabel" left="160" name="myLabel" style="text-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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Interface ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the popup to open or the tab to bring to the foreground.</td>
</tr>
</tbody>
</table>

**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.

```xml
<tab height="91" left="580" tabposition="none" top="373" width="221">
  <abs id="pagel" 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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPath</td>
<td>String</td>
<td>Yes</td>
<td>The XPath command to execute on the given XML Document or XML Element</td>
</tr>
<tr>
<td>Document</td>
<td>String</td>
<td>Yes</td>
<td>XML Document on which to execute the given XPath command.</td>
</tr>
<tr>
<td>Element</td>
<td></td>
<td>Yes</td>
<td>XML Element on which to execute the given XPath command.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>path</td>
<td>String</td>
<td>Yes</td>
<td>The directory path to the destination file.</td>
</tr>
<tr>
<td>data</td>
<td>String, byte[], or XML</td>
<td>Yes</td>
<td>The data to be saved to file.</td>
</tr>
<tr>
<td>overwrite</td>
<td>Boolean</td>
<td>Yes</td>
<td>When true, existing file of the same name is overwritten. When false, existing file of the same name is not overwritten.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host name to send the given data through UDP.</td>
</tr>
<tr>
<td>Port</td>
<td>Integer</td>
<td>Yes</td>
<td>Port number on the given host to be sent given data through UDP.</td>
</tr>
<tr>
<td>Data</td>
<td>ASCII string</td>
<td>Yes</td>
<td>Data to be converted to bytes and sent through UDP to the specified host/port.</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host name to send the given Data bytes through UDP.</td>
</tr>
<tr>
<td>Port</td>
<td>Integer</td>
<td>Yes</td>
<td>Port number on the given Host to send given Data byte through UDP.</td>
</tr>
<tr>
<td>Data</td>
<td>Byte</td>
<td>Yes</td>
<td>Data bytes to send through UDP to the given Host and Port.</td>
</tr>
</tbody>
</table>

**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**

```javascript
ogscript.sendUDPString (Host, Port, Data);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host</td>
<td>String</td>
<td>Yes</td>
<td>Host name to send the given Data string through UDP.</td>
</tr>
<tr>
<td>Port</td>
<td>Integer</td>
<td>Yes</td>
<td>Port number on the given Host to send given Data string through UDP.</td>
</tr>
<tr>
<td>Data</td>
<td>String</td>
<td>Yes</td>
<td>Data string to convert to bytes and send through UDP to the given Host and Port.</td>
</tr>
</tbody>
</table>

**Returns**

N/A

**Example**

This example uses the sendUDPString function to send a message to a particular host/port.

```javascript
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**

```javascript
ogscript.setAnchorPoints (ID, top, left, bottom, right);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the object you want to anchor.</td>
</tr>
<tr>
<td>top</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — object is anchored to the top false — object is not anchored to the top</td>
</tr>
<tr>
<td>left</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — object is anchored to the left false — object is not anchored to the left</td>
</tr>
<tr>
<td>bottom</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — object is anchored to the bottom false — object is not anchored to the bottom</td>
</tr>
<tr>
<td>right</td>
<td>Boolean</td>
<td>Yes</td>
<td>true — object is anchored to the right false — object is not anchored to the right</td>
</tr>
</tbody>
</table>
Returns
N/A

Example
The button in this example has a task that anchors an object (with ID 'dialog') to the top left.

```xml
<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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the panel object to be resized. Valid only in &lt;abs/&gt; containers.</td>
</tr>
<tr>
<td>width</td>
<td>Integer</td>
<td>Yes</td>
<td>New width of the panel, in pixels.</td>
</tr>
<tr>
<td>height</td>
<td>Integer</td>
<td>Yes</td>
<td>New height of the panel, in pixels.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the Component to style with the given Style parameters.</td>
</tr>
<tr>
<td>Style</td>
<td>String</td>
<td>Yes</td>
<td>Style parameters with which to style the given Component.</td>
</tr>
</tbody>
</table>
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="labell" 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('labell','bg#FF0000');

//second change - set the text colour to black and text size to big
ogscript.setStyle('labell','fg#000000;size:big');

//third change - modify the text alignment from left to right
ogscript.setStyle('labell','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('labell','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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the component in which you want to replace XML</td>
</tr>
<tr>
<td>new XML content</td>
<td>String</td>
<td>Yes</td>
<td>The new XML content</td>
</tr>
</tbody>
</table>
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`.

```xml
<br>
   <abs id="my-abs" name="Destination" />
   var oglml '&lt;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.

```
<abs contexttype="opengear">
   <meta>
      <params>
         <param access="1" maxlength="0" name="Text for Label" oid="txt" type="STRING" value="Type Here!" widget="text"/>
      </params>
      <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">&lt;task tasktype="ogscript">var oglml = '&lt;label name="', + params.getValue('txt',0) + '" style="txt-align:center;" anchor="center"</task>&lt;/label'></button>
         </tr>
      </table>
   </meta>
</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:

![Before](image1.png)

After the button is clicked:

![After](image2.png)
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

\[ \text{ogscript.toBottom} \left( \text{ID} \right); \]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID object to be sent to the bottom</td>
</tr>
</tbody>
</table>

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.

```html
<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

\[ \text{ogscript.toTop} \left( \text{ID} \right); \]
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID object to be sent to the top</td>
</tr>
</tbody>
</table>

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.

```xml
<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

```xml
ogscript.upload (Filename);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>String</td>
<td>Yes</td>
<td>Name of the file with which to open the File Upload dialog box.</td>
</tr>
</tbody>
</table>

Returns

N/A

Example

Coming soon.
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:

```javascript
params.function name (parameters);
```

For example:

```javascript
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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createCopy</td>
<td>Source OID Destination OID</td>
<td>ParamScriptable</td>
<td>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.</td>
</tr>
<tr>
<td>createIntChoiceConstraint</td>
<td>[choices]</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>createLinkedCopy</td>
<td>Source OID Destination OID</td>
<td>ParamScriptable</td>
<td>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.</td>
</tr>
<tr>
<td>createMultiSet</td>
<td>String [OID] Integer [Index] Object [Value]</td>
<td>multiset object</td>
<td>Replaces multiple parameter values all at once.</td>
</tr>
<tr>
<td>createParam</td>
<td>JSON parameter definition</td>
<td>N/A</td>
<td>Creates a parameter based on a JSON parameter definition.</td>
</tr>
<tr>
<td>deleteParam</td>
<td>OID of parameter to delete</td>
<td>N/A</td>
<td>Deletes the specified parameter.</td>
</tr>
<tr>
<td>getAllValues</td>
<td>String [OID]</td>
<td>The entire array of values within the parameter.</td>
<td>Retrieves the entire array of values within the parameter.</td>
</tr>
<tr>
<td>getConstraint</td>
<td>String [OID]</td>
<td>Constraint</td>
<td>Get the constraint from the parameter with the specified OID.</td>
</tr>
<tr>
<td>getDeviceStatus</td>
<td>String [OID]</td>
<td>Device status information</td>
<td>Checks the status of a device and returns an Integer value indicating that status:</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>getElementCount</td>
<td>Integer [Context ID] String [OID] Integer [Index]</td>
<td>ParamScriptable</td>
<td>Get the information about an element in a parameter with the specified OID.</td>
</tr>
<tr>
<td>getIdentifiedConstraint</td>
<td>String [ID]</td>
<td>String</td>
<td>Get the constraint with the specified ID. If the ID is an external object URL, get the constraint defined in the specified external object.</td>
</tr>
<tr>
<td>getParam</td>
<td>String [Context ID] String [OID] Integer [Index]</td>
<td>ParamScriptable</td>
<td>Gets information about an element in the parameter with the specified Object ID.</td>
</tr>
<tr>
<td>getParam (OID, Index).remove</td>
<td>String or Integer [OID] Integer [Index]</td>
<td>N/A</td>
<td>Removes a parameter element. If the parameter is an array with more than one element, the element at the index location is removed.</td>
</tr>
<tr>
<td>getStream</td>
<td>String [OID]</td>
<td>Boolean</td>
<td>Check whether streaming of parameter values to XPression is enabled.</td>
</tr>
<tr>
<td>getValue</td>
<td>String [OID] Integer [Index]</td>
<td>String</td>
<td>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.</td>
</tr>
<tr>
<td>getValueAsString</td>
<td>String [OID] Integer [Index]</td>
<td>String</td>
<td>Get a string representation of an element in a parameter with the specified OID.</td>
</tr>
<tr>
<td>isDeviceOnline</td>
<td>String [OID]</td>
<td>Online status of the device as Boolean</td>
<td>Queries a device to determine whether it is online.</td>
</tr>
<tr>
<td>isPrivateParamContext</td>
<td>N/A</td>
<td>Boolean</td>
<td>Returns true if local OGLML-based parameters are operating disconnected from the real device.</td>
</tr>
<tr>
<td>replaceConstraint</td>
<td>String [OID] String [Constraint ID]</td>
<td>N/A</td>
<td>Replace the constraint for the parameter with the specified OID with the constraint with the specified constraint ID.</td>
</tr>
<tr>
<td>replaceViewConstraint</td>
<td>String [view OID] String [constraint object]</td>
<td>N/A</td>
<td>Replaces the constraint object of a parameter view.</td>
</tr>
<tr>
<td>resetAllValues</td>
<td>String [parameter OID]</td>
<td>N/A</td>
<td>If the specified parameter is a copy of a base parameter, this function resets the parameter’s values to those of the base parameter.</td>
</tr>
<tr>
<td>setAccess</td>
<td>String [OID], Integer [Access]</td>
<td>N/A</td>
<td>Set the access level of the parameter with the provided OID.</td>
</tr>
<tr>
<td>setAllValues</td>
<td>OID Object[] Values</td>
<td>N/A</td>
<td>For an array parameter, replaces the current array with the new array.</td>
</tr>
<tr>
<td>setMenuState</td>
<td>Integer [Static Menu ID], Integer [Menu State]</td>
<td>N/A</td>
<td>Set the menu state of the menu with the specified static menu ID.</td>
</tr>
<tr>
<td>setPrivateParamContext</td>
<td>Boolean [Value]</td>
<td>N/A</td>
<td>true — disconnect parameters defined in the OGLML document from the device.</td>
</tr>
</tbody>
</table>
### Function Parameters

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setStream</td>
<td>String [OID], Boolean [true/false]</td>
<td>N/A</td>
<td>Controls whether the parameter streams its values to XPression when XPression streaming is on. When true, streaming is ON. When false, streaming is OFF.</td>
</tr>
<tr>
<td>setValue</td>
<td>String [OID], Integer [Index], Object [Value]</td>
<td>N/A</td>
<td>Set the value of an element in a parameter with the provided OID to the provided value.</td>
</tr>
<tr>
<td>setValueRelative</td>
<td>String [OID], Integer [Index], Integer [change in value]</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>toOid</td>
<td>String (OID)</td>
<td>N/A</td>
<td>Creates an OID object.</td>
</tr>
</tbody>
</table>

#### 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**

```java
params.createCopy (Source OID, Destination OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter to copy</td>
</tr>
<tr>
<td>Destination OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the new parameter.</td>
</tr>
</tbody>
</table>

**Returns**

Returns ParamScriptable. For more information, see [ParamScriptable Object](#) on page 277.

**Example**

Coming soon.

#### createIntChoiceConstraint

**Syntax**

```java
params.createIntChoiceConstraint (Choices);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Choices</td>
<td>String</td>
<td>Yes</td>
<td>Name of the array variable that contains the choices.</td>
</tr>
</tbody>
</table>

**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

```javascript
params.createLinkedCopy (Source OID, Destination OID);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter to copy</td>
</tr>
<tr>
<td>Destination</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the new parameter.</td>
</tr>
</tbody>
</table>

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

```javascript
params.createMultiSet (OID, Index, Value);
```

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of object of interest.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.</td>
</tr>
<tr>
<td>Value</td>
<td>Object</td>
<td>Yes</td>
<td>New value for the OID.</td>
</tr>
</tbody>
</table>

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.

```html
<abs contexttype="opengear" gridsize="20" style="">
<meta>
<params>
```
<param access="1" maxlen="0" name="Value 1" oid="Value_1" stateless="true" type="STRING" value="Original Value 1" widget="100"/>
<param access="1" maxlen="0" name="Value 2" oid="Value_2" stateless="true" type="STRING" value="Original Value 2" widget="100"/>
<param access="1" maxlen="0" name="Value 3" oid="Value_3" stateless="true" type="STRING" value="Original Value 3" widget="100"/>
<param access="1" maxlen="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>

createParam

Creates a parameter based on a JSON parameter definition.

Syntax

params.createParam (JSON parameter definition);

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JSON parameter</td>
<td>String</td>
<td>Yes</td>
<td>definition of the parameter.</td>
</tr>
<tr>
<td>definition</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Returns

N/A

Example

Coming soon.
**deleteParam**

Deletes the specified parameter.

**Syntax**

```javascript
params.deleteParam (OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of parameter to delete</td>
</tr>
</tbody>
</table>

**Returns**

N/A

**Example**

Coming soon.

**getAllValues**

Retrieves the entire array of values from a parameter.

**Syntax**

```javascript
params.getAllValues (OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter</td>
</tr>
</tbody>
</table>

**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**

```javascript
params.getConstraint (OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of the object of interest.</td>
</tr>
</tbody>
</table>

**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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of the device to be queried.</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter.</td>
</tr>
</tbody>
</table>

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
<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the constraint of interest.</td>
</tr>
</tbody>
</table>
**Returns**
String

**Example**
Coming soon.

### `getParam`

Gets information about an element in the parameter with the specified Object ID.

**Syntax**

```javascript
params.getParam (Context ID, OID, Index);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Context ID</td>
<td>String</td>
<td>No</td>
<td>The context ID of the component that contains the parameter of interest.</td>
</tr>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of the object of interest.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.</td>
</tr>
</tbody>
</table>

**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**

```javascript
params.getParam ([oid], [index]).remove();
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String or Integer</td>
<td>Yes</td>
<td>OID can be a string or an integer, depending on how the parameter is defined.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not an array parameter, use an Index of 0.</td>
</tr>
</tbody>
</table>

**Returns**
N/A

**Example**
Coming soon.
**getStream**

Checks whether streaming of parameter values to XPression is enabled for the parameter.

**Syntax**

```javascript
params.getStream (OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of the parameter</td>
</tr>
</tbody>
</table>

**Returns**

Boolean, to indicate whether streaming is enabled.

**Example**

Coming soon.

---

**getValue**

Gets the value of a parameter with the specified Object ID.

**Syntax**

```javascript
params.getValue (OID, Index);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of object of interest.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.</td>
</tr>
</tbody>
</table>

**Returns**

String

**Example**

```javascript
var data = params.getValue (0x12,0);
```

---

**getValueAsString**

Gets a string representation of an element in a parameter with the specified Object ID.

**Syntax**

```javascript
params.getValueAsString (OID, Index);
```

**Example**

```javascript
var data = params.getValueAsString (0x12,0);
```
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of the object of interest.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>an array parameter, use an Index of 0. In most</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>cases, enter 0 as the Index.</td>
</tr>
</tbody>
</table>

### Returns

ParamScriptable

### Example

Coming soon.

### isDeviceOnline

Queries a device to determine whether it is online.

#### Syntax

```
params.isDeviceOnline (OID);
```

#### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of device to query.</td>
</tr>
</tbody>
</table>

### 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

\[
\text{params.replaceConstraint (OID, Constraint ID)};
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of object of interest.</td>
</tr>
<tr>
<td>Constraint ID</td>
<td>String</td>
<td>Yes</td>
<td>ID of the constraint with which to replace the constraint for the parameter with the specified Object ID.</td>
</tr>
</tbody>
</table>

Returns

N/A

Example

Coming soon.

replaceViewConstraint

Replaces the constraint object of a parameter view.

Syntax

\[
\text{params.replaceViewConstraint (view OID, constraint object)};
\]

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>view OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of the view.</td>
</tr>
<tr>
<td>constraint object</td>
<td>String</td>
<td>Yes</td>
<td>constraint object to use.</td>
</tr>
</tbody>
</table>

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

\[
\text{params.resetAllValues (parameter OID)};
\]
Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>parameter OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of object of interest.</td>
</tr>
<tr>
<td>Access</td>
<td>Integer</td>
<td>Yes</td>
<td>Access level to set for the specified OID. The available access levels are as follows: 0 — Read Only 1 — Read and Write</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>The OID of the parameter.</td>
</tr>
<tr>
<td>Array</td>
<td>String</td>
<td>Yes</td>
<td>The new array.</td>
</tr>
</tbody>
</table>

Returns

N/A

Example

Coming soon.
**setMenuState**

Set the menu state of the menu with the provided static menu ID.

**Syntax**

```javascript
params.setMenuState (Static Menu ID, Menu State);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static Menu ID</td>
<td>Integer</td>
<td>Yes</td>
<td>ID of the menu of interest.</td>
</tr>
<tr>
<td>Menu State</td>
<td>Integer</td>
<td>Yes</td>
<td>Menu state to set for the specified Static Menu ID. The available menu states are as follows: 0 — Hidden 1 — Disabled 2 — Normal</td>
</tr>
</tbody>
</table>

**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**

```javascript
params.setPrivateParamContext (Value);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>Boolean</td>
<td>Yes</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Returns**

N/A

**Example**

Coming soon.
**setStream**

Controls whether a parameter streams its values to XPression when XPression streaming is on.

**Syntax**

```javascript
params.getStream (OID, true/false);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>OID of the parameter</td>
</tr>
<tr>
<td>true/false</td>
<td>Boolean</td>
<td>Yes</td>
<td>When true, streaming is ON. When false, streaming is OFF.</td>
</tr>
</tbody>
</table>

**Returns**

N/A

**Example**

Coming soon.

**setValue**

Set the value of a parameter for the provided Object ID.

**Syntax**

```javascript
params.setValue (OID, Index, Value);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>Object ID of object of interest.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Array parameter index. If the parameter is not an array parameter, use an Index of 0. In most cases, enter 0 as the Index.</td>
</tr>
<tr>
<td>Value</td>
<td>Object</td>
<td>Yes</td>
<td>New value for the OID.</td>
</tr>
</tbody>
</table>

**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**

```javascript
params.setValueRelative (OID, Index, Change in value);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String or Integer</td>
<td>Yes</td>
<td>The OID of the object of which you want to change the value.</td>
</tr>
<tr>
<td>Index</td>
<td>Integer</td>
<td>Yes</td>
<td>Position of data in the parameter.</td>
</tr>
<tr>
<td>Change in Value</td>
<td>Integer</td>
<td>Yes</td>
<td>Amount by which the value is incremented. To decrement the value, use a negative integer.</td>
</tr>
</tbody>
</table>

**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**

```javascript
var subList = new Array();
subList.push("oid1");
subList.push("oid2");
var subscriptionOwnerObject = params.subscribe(subList, callback);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriptionOwnerObject</td>
<td><code>params.subscribe</code></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
subList | [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.

<table>
<thead>
<tr>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>subscriptionOwnerObject for later use to unsubscribe.</td>
</tr>
</tbody>
</table>

"params.subscribe" Example

```ogscript
<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);

<table>
<thead>
<tr>
<th>Function</th>
<th>Type</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>unsubscribe</td>
<td>[subscriptionOwnerObject]</td>
<td>N/A</td>
<td>Unsubscribes from the OIDs provided by the subscriptionOwnerObject.</td>
</tr>
</tbody>
</table>

"params.subscribe" Example

```ogscript
<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**

```javascript
params.toOid (OID);
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Type</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID</td>
<td>String</td>
<td>Yes</td>
<td>The value of the new OID object.</td>
</tr>
</tbody>
</table>

**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.

```javascript
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:

```ogScript
this.methodname(parameters);
```

For example:

```ogScript
this.replaceConstraint("0.0;100.0;0.0;100.0;1");
```

The following table lists the methods of the ParamScriptable object.

<table>
<thead>
<tr>
<th>Method</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deleteParam</td>
<td>N/A</td>
<td>N/A</td>
<td>Deletes the parameter</td>
</tr>
<tr>
<td>getConstraint</td>
<td>N/A</td>
<td>Returns the parameter constraint</td>
<td>Gets the parameter constraint</td>
</tr>
<tr>
<td>getAllValues</td>
<td>N/A</td>
<td>The entire array of values within the parameter.</td>
<td>Retrieves the entire array of values within the parameter.</td>
</tr>
<tr>
<td>getElementCount</td>
<td>N/A</td>
<td>The number of elements in the parameter array, as an Integer.</td>
<td>Gets the number of elements in the parameter array.</td>
</tr>
<tr>
<td>getIndex</td>
<td>N/A</td>
<td>Returns the array index of the current element</td>
<td>Gets the array index of the current element</td>
</tr>
<tr>
<td>getOid</td>
<td>N/A</td>
<td>Returns the OID of the changed parameter</td>
<td>Gets the OID of the changed parameter</td>
</tr>
<tr>
<td>getValue</td>
<td>N/A</td>
<td>Returns the value of the changed element</td>
<td>Gets the value of the changed element</td>
</tr>
<tr>
<td>getValueAsString</td>
<td>N/A</td>
<td>Returns a string representation of the changed value</td>
<td>Gets a string representation of the changed value</td>
</tr>
<tr>
<td>getValueAt</td>
<td>Integer [index]</td>
<td>Returns a string representation of the value at the provided index</td>
<td>Gets a string representation of the value at the provided index</td>
</tr>
<tr>
<td>getValueAtAsString</td>
<td>Integer [index]</td>
<td>Returns a string representation of the value at the provided index</td>
<td>Gets a string representation of the value at the provided index</td>
</tr>
<tr>
<td>setValue</td>
<td>String [value]</td>
<td>N/A</td>
<td>Sets the value of the changed element to the provided value.</td>
</tr>
<tr>
<td>getName</td>
<td>N/A</td>
<td>Returns the parameter name</td>
<td>Gets the parameter name</td>
</tr>
<tr>
<td>replaceConstraint</td>
<td>String [Constraint]</td>
<td>N/A</td>
<td>Replaces the parameter's constraint to the provided value.</td>
</tr>
<tr>
<td>remove</td>
<td>N/A</td>
<td>N/A</td>
<td>Removes the current array element</td>
</tr>
<tr>
<td>isArrayParameter</td>
<td>N/A</td>
<td>Returns true if the parameter is an array element</td>
<td>Checks whether the parameter is an array element</td>
</tr>
<tr>
<td>resetAllValues</td>
<td>N/A</td>
<td>N/A</td>
<td>If the parameter is a copy of a base parameter, this function resets the parameter's values to those of the base parameter.</td>
</tr>
<tr>
<td>setValueAt</td>
<td>Integer [index]</td>
<td>String [value]</td>
<td>Sets the value of element at the provided index to the provided value.</td>
</tr>
<tr>
<td>getElementCount</td>
<td>N/A</td>
<td>Returns the number of elements in the array</td>
<td>Gets the number of elements in the array</td>
</tr>
</tbody>
</table>
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:

```javascript
rosstalk.function name (parameters);
```

For example:

```javascript
rosstalk.setHost (Server01);
```

The following table lists the functions of the rosstalk object.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setHost</td>
<td>String [Host]</td>
<td>N/A</td>
<td>Set a default host to use for RossTalk commands where no host has been defined.</td>
</tr>
<tr>
<td>getHost</td>
<td>N/A</td>
<td>String</td>
<td>Get the default host previously defined.</td>
</tr>
<tr>
<td>setPort</td>
<td>Integer [Port]</td>
<td>N/A</td>
<td>Set a default port to use for RossTalk commands where no host has been defined.</td>
</tr>
<tr>
<td>getPort</td>
<td>N/A</td>
<td>Integer</td>
<td>Get the default port previously defined.</td>
</tr>
<tr>
<td>sendAsBytes</td>
<td>String [Host], Int [Port], String [Bytes as Hex String]</td>
<td>N/A</td>
<td>Equivalent of calling: sendAsBytes(host, port, bytes, null);</td>
</tr>
<tr>
<td>sendAsBytes</td>
<td>String [Host], Int [Port], String [Bytes as Hex String], Function [Callback]</td>
<td>N/A</td>
<td>Convert bytes from string (where string is formatted as ASCII representations of bytes e.g. &quot;FDDFEAAE12F9...&quot;) and send them to the provided host at the provided port. Invoke the callback function when done.</td>
</tr>
<tr>
<td>sendAsBytesWithResponse</td>
<td>String [Host], Int [Port], String [Bytes as Hex String], String [responseBytes], Function [Callback]</td>
<td>Response message provided by the recipient.</td>
<td>Convert bytes from string (where string is formatted as ASCII representations of bytes e.g. &quot;FDDFEAAE12F9...&quot;) 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.</td>
</tr>
<tr>
<td>sendBytes</td>
<td>String [Host], Int [Port], Byte[] [Data to Send], Function [Callback]</td>
<td>N/A</td>
<td>Send the provided bytes to the provided host at the provided port. Invoke the callback function when done.</td>
</tr>
<tr>
<td>sendBytesWithResponse</td>
<td>String [Host], Int [Port], Byte[] [Data to Send], Byte [responseTerminator] Function [Callback]</td>
<td>Response message provided by the recipient.</td>
<td>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 recipient, indicates the end of the response message.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td>---------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>sendMessage</td>
<td>String [RossTalk Command]</td>
<td>N/A</td>
<td>Equivalent of calling: sendMessage (getHost(), getPort(), RossTalk Command, null);</td>
</tr>
<tr>
<td>sendMessage</td>
<td>String [RossTalk Command], Function [Callback]</td>
<td>N/A</td>
<td>Equivalent of calling: sendMessage (getHost(), getPort(), RossTalk Command, Callback);</td>
</tr>
<tr>
<td>sendMessage</td>
<td>String [Host], Int [Port], String [RossTalk Command]</td>
<td>N/A</td>
<td>Equivalent of calling: sendMessage (Host, Port, RossTalk Command, null);</td>
</tr>
<tr>
<td>sendMessage</td>
<td>String [Host], Int [Port], String [RossTalk Command], Function [Callback]</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>sendMessageWithResponse</td>
<td>String [Host], Int [Port], String [RossTalk Command], String [responseTerminator], Function [Callback]</td>
<td>Response message provided by the recipient.</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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:

```javascript
rosstalkex.function name (parameters);
```

For example:

```javascript
rosstalkex.sendMessage("10.3.2.1", 8020, "DATALINQKEY 101:kl:v1", null);
```

The following table lists the functions of the rosstalkex object.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getConnection</td>
<td>String [Host], Integer [Port], Boolean [Creation Flag]</td>
<td>If the handshake is successful, the connection object is returned. If the handshake is not successful, a null value is returned.</td>
<td>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.</td>
</tr>
<tr>
<td>sendMessage</td>
<td>String [Host], Integer [Port], String [Message], Function [Callback]</td>
<td>If the authentication is not successful, no message is sent and an error message is thrown, otherwise, nothing is returned.</td>
<td>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.</td>
</tr>
</tbody>
</table>
**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:

```ogScript
robot.function name (parameters);
```

For example:

```ogScript
robot.setHost (Server01);
```

The following table lists the functions of the robot object.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setHost</td>
<td>String [Host]</td>
<td>N/A</td>
<td>Set a default host to use for CamBot commands where no host has been defined.</td>
</tr>
<tr>
<td>getHost</td>
<td>N/A</td>
<td>String</td>
<td>Get the default host previously defined.</td>
</tr>
<tr>
<td>setPort</td>
<td>Integer [Port]</td>
<td>N/A</td>
<td>Set a default port to use for CamBot commands where no host has been defined.</td>
</tr>
<tr>
<td>getPort</td>
<td>N/A</td>
<td>Integer</td>
<td>Get the default port previously defined.</td>
</tr>
<tr>
<td>sendCambot</td>
<td>String [CamBot Command]</td>
<td>N/A</td>
<td>Equivalent of calling: <code>sendCambot(getHost(), getPort(), command, null)</code></td>
</tr>
<tr>
<td>sendCambot</td>
<td>String [CamBot Command] Function</td>
<td>N/A</td>
<td>Equivalent of calling: <code>sendCambot(getHost(), getPort(), CamBot Command, Callback)</code></td>
</tr>
<tr>
<td>sendCambot</td>
<td>String [Host], Int [Port], String [CamBot Command] Function [Callback]</td>
<td>N/A</td>
<td>Equivalent of calling: <code>sendCambot(Host, Port, CamBot Command, null)</code></td>
</tr>
<tr>
<td>sendCambot</td>
<td>String [Host], Int [Port], String [CamBot Command] Function [Callback]</td>
<td>N/A</td>
<td>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)</td>
</tr>
</tbody>
</table>

**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:

```ogScript
vdcp.function name (parameters);
```

For example:

```ogScript
vdcp.setHost (Server01);
```

The following table lists the functions of the vdcp object.
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>setHost</td>
<td>String [Host]</td>
<td>N/A</td>
<td>Set a default host to use for VDCP commands where no host has been defined.</td>
</tr>
<tr>
<td>getHost</td>
<td>N/A</td>
<td>String</td>
<td>Get the default host previously defined.</td>
</tr>
<tr>
<td>setPort</td>
<td>Integer [Port]</td>
<td>N/A</td>
<td>Set a default port to use for VDCP commands where no host has been defined.</td>
</tr>
<tr>
<td>getPort</td>
<td>N/A</td>
<td>Integer</td>
<td>Get the default port previously defined.</td>
</tr>
<tr>
<td>activeClip</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td>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)</td>
</tr>
<tr>
<td>clipDuration</td>
<td>String [Host], Int [Port], Int [Channel], String [ClipID], Function [Callback]</td>
<td>N/A</td>
<td>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)</td>
</tr>
<tr>
<td>continuePlay</td>
<td>String [Host], Int [Port], Int [Channel]</td>
<td>N/A</td>
<td>Sends the vdcp continuePlay command.</td>
</tr>
<tr>
<td>cueClip</td>
<td>String [Host], Int [Port], Int [Channel]</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>fastForward</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>listClips</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>pause</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>play</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td>Sends the vdcp variPlay command.</td>
</tr>
<tr>
<td>rewind</td>
<td>String [Host], Int [Port], Int [Channel]</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>stop</td>
<td>String [Host], Int [Port], Int [Channel], Function [Callback]</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>convertCommaSeparatedLevelsToMask</td>
<td>String [Levels], Boolean [SearchTags]</td>
<td>Long Levelmask</td>
<td>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'.</td>
</tr>
<tr>
<td>doSwitch</td>
<td>N/A</td>
<td>Boolean</td>
<td>Equivalent of calling: doSwitch(getActiveDst(), getActiveSrc(), getLevelMask());</td>
</tr>
<tr>
<td>doSwitch</td>
<td>Int [Dst], Int [Src], Long [Levels]</td>
<td>Boolean</td>
<td>Do a switch on the active IPS to route the given destination to the given source on the given levels.</td>
</tr>
<tr>
<td>doSwitchWithLabels</td>
<td>String [Destination], String [Source], String [Levels]</td>
<td>Boolean</td>
<td>Allows you to switch between levels by name.</td>
</tr>
<tr>
<td>getActiveDst</td>
<td>N/A</td>
<td>Int</td>
<td>Get the active destination number (0-indexed). Returns -1 if there is no active destination.</td>
</tr>
<tr>
<td>getActiveDstName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active destination (from the switchboard configuration). Returns null if there is no active destination.</td>
</tr>
<tr>
<td>getActiveIPS</td>
<td>N/A</td>
<td>String</td>
<td>Get the serial number of the active IPS.</td>
</tr>
<tr>
<td>getActiveIPSName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active IPS.</td>
</tr>
<tr>
<td>getActiveSrc</td>
<td>N/A</td>
<td>Int</td>
<td>Get the active source number (0-indexed). Returns -1 if there is no active source.</td>
</tr>
<tr>
<td>getActiveSrcName</td>
<td>N/A</td>
<td>String</td>
<td>Get the name of the active src (from the switchboard configuration). Returns null if there is no active source.</td>
</tr>
<tr>
<td>getActiveSystem</td>
<td>N/A</td>
<td>NKSystem</td>
<td>Get the currently active NKSystem.</td>
</tr>
<tr>
<td>getDstName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the destination name of the given source.</td>
</tr>
</tbody>
</table>
| getLevelMask                    | N/A                                 | Long             | Get the current level mask (as a bit field) Level mask is a bit field where
<table>
<thead>
<tr>
<th>Function</th>
<th>Parameters</th>
<th>Returns</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getLevelName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the level name of the given source.</td>
</tr>
<tr>
<td>getProtectStatus</td>
<td>String [Destination], String [Levels]</td>
<td>Boolean</td>
<td>Get the protect status of the destination level.</td>
</tr>
<tr>
<td>getSrcName</td>
<td>String [Source]</td>
<td>String</td>
<td>Get the source name.</td>
</tr>
<tr>
<td>getStatu</td>
<td>String [Destination], String [Level]</td>
<td>Int</td>
<td>Get the status of the given destination level.</td>
</tr>
<tr>
<td>isLevelActive</td>
<td>Int [Level Num]</td>
<td>Boolean</td>
<td>Is the current level active.</td>
</tr>
<tr>
<td>isMCFlag</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is the Machine Control flag set.</td>
</tr>
<tr>
<td>isProtected</td>
<td>N/A</td>
<td>Boolean</td>
<td>Verifies whether the active destination is protected or not.</td>
</tr>
<tr>
<td>isProtected</td>
<td>Int [Destination], Long [Levels]</td>
<td>Boolean</td>
<td>Verifies whether the given destination is protected; or not.</td>
</tr>
<tr>
<td>isProtectedByMe</td>
<td>N/A</td>
<td>Boolean</td>
<td>Is the active destination protected by this virtual panel.</td>
</tr>
<tr>
<td>isSrcActive</td>
<td>Int [Src]</td>
<td>Boolean</td>
<td>Verifies whether the given source is active on the active destination of all levels.</td>
</tr>
<tr>
<td>isSrcActive</td>
<td>Int [Dst], Int [Src], Long [Levels]</td>
<td>Boolean</td>
<td>Verifies whether the provided source is active on the specified destination and level mask.</td>
</tr>
<tr>
<td>isVirtual</td>
<td>N/A</td>
<td>Boolean</td>
<td>Verifies whether virtual routing is in use (for switch commands and status requests).</td>
</tr>
<tr>
<td>setActiveDst</td>
<td>Int [Dst]</td>
<td>N/A</td>
<td>Set the active destination (0-indexed).</td>
</tr>
<tr>
<td>setActiveIPS</td>
<td>String [Serial]</td>
<td>Boolean</td>
<td>Set the IPS with the given serial number as the active IPS to receive commands and send status. Deactivate any currently active IPS.</td>
</tr>
<tr>
<td>setActiveSrc</td>
<td>Int [Src]</td>
<td>N/A</td>
<td>Set the active source (0-indexed).</td>
</tr>
<tr>
<td>setLevelActive</td>
<td>Int [Level Num], Boolean [Active]</td>
<td>Boolean</td>
<td>Set the given level as active.</td>
</tr>
<tr>
<td>setLevelMask</td>
<td>Long [Level Mask]</td>
<td>N/A</td>
<td>Set the complete level mask bitfield.</td>
</tr>
<tr>
<td>setMCFlag</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Set the Machine Control flag to true or false.</td>
</tr>
<tr>
<td>setProtected</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Request the router to protect the active destination.</td>
</tr>
<tr>
<td>setVirtual</td>
<td>Boolean</td>
<td>Boolean</td>
<td>Set virtual routing on/off for switch commands and status.</td>
</tr>
<tr>
<td>Function</td>
<td>Parameters</td>
<td>Returns</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>------------</td>
<td>---------</td>
<td>------------------------------------------</td>
</tr>
<tr>
<td>verifyConfiguration</td>
<td>N/A</td>
<td>Boolean</td>
<td>Re-activate the current iPS.</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>OID</th>
<th>Type</th>
<th>Constr int</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUPPLIER_NAME*</td>
<td>0x0102</td>
<td>String</td>
<td>N/A</td>
<td>Name of the card manufacturer or OEM supplier (i.e. who customer should call for support). Reported as a generic card parameter by SNMP.</td>
</tr>
<tr>
<td>PRODUCT_NAME**</td>
<td>0x0105</td>
<td>String (32-bytes max)</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>SERIAL_NUMBER</td>
<td>0x0106</td>
<td>String</td>
<td>N/A</td>
<td>Unique serial number.</td>
</tr>
<tr>
<td>SOFTWARE_REV**</td>
<td>0x010B</td>
<td>String (20-bytes max)</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>FPGA_REV+</td>
<td>0x010C</td>
<td>String</td>
<td>N/A</td>
<td>This value is used by DashBoard compare software versions when uploading the Main Board FPGA Type (upload type 1).</td>
</tr>
<tr>
<td>OPTION_SOFTWARE_REV+</td>
<td>0x010D</td>
<td>String</td>
<td>N/A</td>
<td>This value is used by DashBoard compare software versions when uploading the Option Board Software Type (upload type 2).</td>
</tr>
<tr>
<td>OPTION_FPGA_REV+</td>
<td>0x010E</td>
<td>String</td>
<td>N/A</td>
<td>This value is used by DashBoard compare software versions when uploading the Option Board FPGA Type (upload type 3).</td>
</tr>
<tr>
<td>SMPTE_STATUS</td>
<td>0x0201</td>
<td>Int16</td>
<td>N/A</td>
<td>Card status to be reported via frame fault LEDs. Value of 0 indicates no error. Non-zero values indicate error state.</td>
</tr>
<tr>
<td>CURRENT_MILLIS</td>
<td>0x0205</td>
<td>Int16</td>
<td>N/A</td>
<td>Current consumption in milliamps at 12 V. This may be used by the fan controller to adjust fan speed for high-current cards.</td>
</tr>
<tr>
<td>EDIT_PERMISSION</td>
<td>0x0601</td>
<td>Int16</td>
<td>Choice</td>
<td>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.</td>
</tr>
<tr>
<td>FRAME_POWER_CAPABILITY</td>
<td>0xFE0F</td>
<td></td>
<td></td>
<td>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</td>
</tr>
</tbody>
</table>

---
<table>
<thead>
<tr>
<th>Name</th>
<th>OID</th>
<th>Type</th>
<th>Constraint</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name_OVERRIDE</td>
<td>0xFF01</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>CONNECT_VERIFY</td>
<td>0xFF03</td>
<td>Mixed</td>
<td>N/A</td>
<td>This parameter is used for communicating DashBoard's connection handshake and response.</td>
</tr>
<tr>
<td>UPLOAD_URL</td>
<td>0xFF02</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>FRAME_ID</td>
<td>0xFF04</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>BACKWARDS_COMPATIBLE</td>
<td>0xFF05</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>RESTORE_SET_DELAY</td>
<td>0xFF06</td>
<td>Int16</td>
<td>N/A</td>
<td>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</td>
</tr>
<tr>
<td>Name</td>
<td>OID</td>
<td>Type</td>
<td>Constraint</td>
<td>Function</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td>RESTORE_START</td>
<td>0xFF07</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>RESTORE_STOP</td>
<td>0xFF08</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>DATASAFE_NAME</td>
<td>0xFF09</td>
<td>String</td>
<td>N/A</td>
<td>Alternative card name for determining DataSafe compatibility.</td>
</tr>
<tr>
<td>UPLOAD_NAME</td>
<td>0xFF0A</td>
<td>Int16</td>
<td>Choice</td>
<td>Alternative card name for file upload purposes.</td>
</tr>
<tr>
<td>DISPLAY_OPTIONS</td>
<td>0xFF0B</td>
<td>Int16</td>
<td>ARRA Y</td>
<td>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.</td>
</tr>
<tr>
<td>DEVICE_ICON</td>
<td>0xFF0C</td>
<td>Int16</td>
<td>N/A</td>
<td>Contains an external object ID for an encapsulated icon.</td>
</tr>
<tr>
<td>DEVICE_INDEX_URL</td>
<td>0xFF0D</td>
<td>String</td>
<td>N/A</td>
<td>URL for a DashBoard Connect XML Definition.</td>
</tr>
<tr>
<td>OGLML_DESCRIPTOR</td>
<td>0xFF0E</td>
<td>String</td>
<td>N/A</td>
<td>Provides an OGLML URL that describes a</td>
</tr>
<tr>
<td>Name</td>
<td>OID</td>
<td>Type</td>
<td>Constraint</td>
<td>Function</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------</td>
<td>---------</td>
<td>------------</td>
<td>----------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>layout to use in place of the standard configuration screen in DashBoard.</td>
</tr>
<tr>
<td>DEDICATED_CONNECTION</td>
<td>0xFF0F</td>
<td>Binary</td>
<td>N/A</td>
<td>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</td>
</tr>
<tr>
<td>DEVICE_IP_ADDRESS</td>
<td>0xFF10</td>
<td>Int32</td>
<td>IP_ADDRESS</td>
<td>Cards that have their own Ethernet port should use this OID to report their current IPv4 address.</td>
</tr>
<tr>
<td>FAN_SPEED_REQUEST</td>
<td>0xFF11</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>OCCUPIED_SLOTS</td>
<td>0xFF12</td>
<td>Int16</td>
<td>N/A</td>
<td>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 &lt;&lt; 8)</td>
</tr>
<tr>
<td>UPLOAD_FILE_EXTENSIONS</td>
<td>0xFF13</td>
<td>String</td>
<td>Array</td>
<td>Extensions of file types allowed to be sent to the device. Array elements have the format: &quot;[Description]&lt;ext:[extension without dot]&gt;&quot;</td>
</tr>
<tr>
<td>RESERVED</td>
<td>0xFF14 to 0xFFFF</td>
<td>...</td>
<td>...</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>

** 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.

<table>
<thead>
<tr>
<th>Name</th>
<th>OID</th>
<th>Type</th>
<th>Constr int</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOOR_STATE</td>
<td>0x0709</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>SLOT_NAMES</td>
<td>0x803</td>
<td>Int16_Array</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>SLOT_DATA_SAFE</td>
<td>0x802</td>
<td>Int16_Array</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>URM_STATE</td>
<td>0xFE01</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>MASTER_PASSWORD</td>
<td>0xFE02</td>
<td>String (20-bytes max)</td>
<td>N/A</td>
<td>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”.</td>
</tr>
<tr>
<td>APPLY_BUTTON</td>
<td>0xFE03</td>
<td>Int16</td>
<td>Choice</td>
<td>The button DashBoard must press to apply changes to the master password or URM state parameters.</td>
</tr>
<tr>
<td>CANCEL_BUTTON</td>
<td>0xFE04</td>
<td>Int16</td>
<td>Choice</td>
<td>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.</td>
</tr>
<tr>
<td>DEVICE_CATEGORY</td>
<td>0xFE05</td>
<td>String</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>FRAME_ICON</td>
<td>0xFE06</td>
<td>Int16</td>
<td>N/A</td>
<td>Contains an external object ID for an encapsulated icon.</td>
</tr>
<tr>
<td>CONFIG_SLOT</td>
<td>0xFE07</td>
<td>Int16</td>
<td>N/A</td>
<td>Default: 0 The slot # for the device to open when the frame is ‘opened’ for configuration.</td>
</tr>
<tr>
<td>CONFIG_URL</td>
<td>0xFE08</td>
<td>String</td>
<td>N/A</td>
<td>Default: [none] If defined and non-empty, the URL of a web page to open when the frame is ‘opened’ for configuration.</td>
</tr>
<tr>
<td>INDEX_URL</td>
<td>0xFE09</td>
<td>String</td>
<td>N/A</td>
<td>URL for a DashBoard Connect XML Definition.</td>
</tr>
<tr>
<td>MASTER_PASSWORD_SAVE</td>
<td>0xFE0A</td>
<td>String</td>
<td>N/A</td>
<td>Same as 0xFE02 above, but used for internal storage on the MFC controller.</td>
</tr>
<tr>
<td>Name</td>
<td>OID</td>
<td>Type</td>
<td>Constraint</td>
<td>Function</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>--------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>FAN_DOOR_STATUS</td>
<td>0xFE0B</td>
<td>Int16</td>
<td>N/A</td>
<td>Broadcast by the MFC every 10 seconds to indicate door status. 1= closed 2= open This replaces legacy OID 0x0709.</td>
</tr>
<tr>
<td>FAN_AMBIENT_TEMP</td>
<td>0xFE0C</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>FAN_SPEED_REPORT</td>
<td>0xFE0D</td>
<td>Int16</td>
<td>N/A</td>
<td>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.</td>
</tr>
<tr>
<td>RESERVED</td>
<td>0xFE0E</td>
<td>...</td>
<td>...</td>
<td>Reserved for future use</td>
</tr>
</tbody>
</table>