Learn the Basics of DashBoard 8.6:

A CustomPanel Workbook

(Estimated time: 4 Hours)

A compilation of training exercises, examples, and useful tips to get the most out of your CustomPanels.

Contents

Introduction to CustomPanels

Custom Panel Files are XML files that describe the layout and scripts within your custom panel.

They can be modified in a text editor if so desired, DashBoard has built in tools to easily modify them and editing by hand should only be done by those who understand the format of the files.

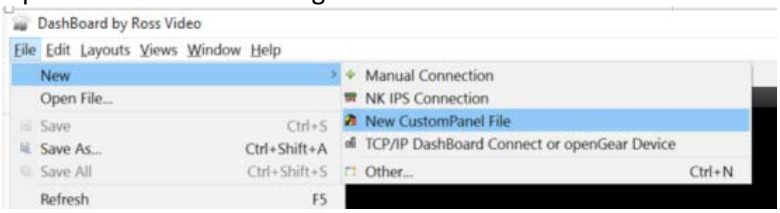
The Folder\Directory that you place the file in should be added to the **File Navigator** within DashBoard as shown in the next section.

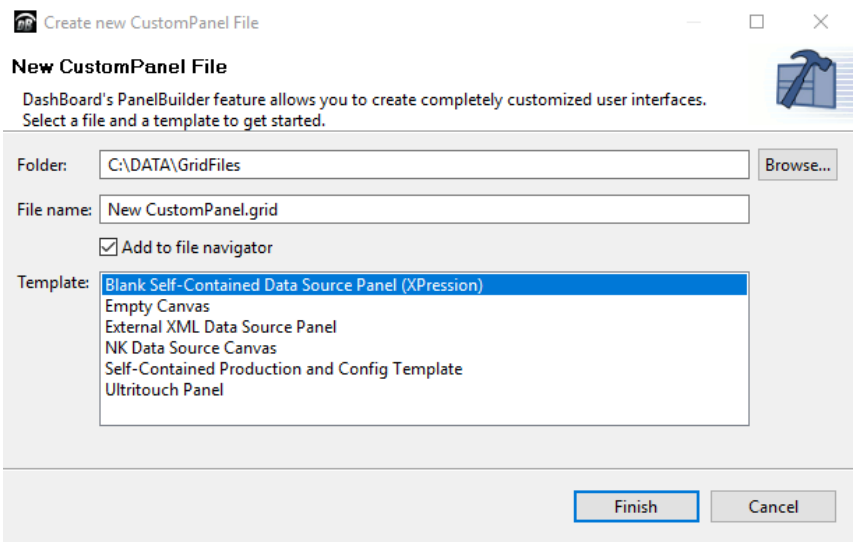
Creating a CustomPanel

You can learn how to create a CustomPanel in DashBoard.

To create a CustomPanel

1. Open DashBoard and navigate to **File** > **New** > **New CustomPanel File**.





A popup appears: (your directories will likely be different).

2. For the **Template**, leave the default selection, or choose the appropriate template.

3. Click **Finish**.

This will create a custom panel file on your computer with the extension .grid.

Example CustomPanels

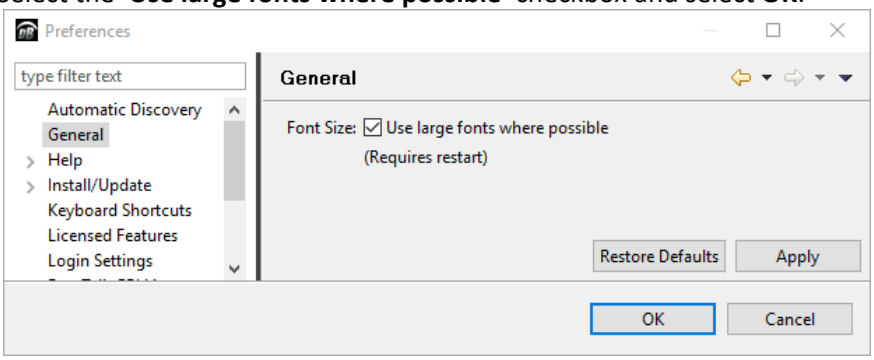
Check out some example panels below:

Using the DashBoard Basic Workspace

This section provides an introduction to working with DashBoard's workspace, including: workspace preferences, and how to use the File Navigator, PanelBuilder, and the Basic Tree View.

To Change the Font Size

1. On the DashBoard menu, navigate to **Window > Preferences > General**.
2. Select the '**Use large fonts where possible**' checkbox and select **OK**.



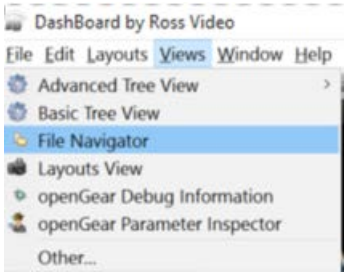
To Toggle Full Screen

Custom Panels can be made to go full screen by using <**Shift-F11**>. This keyboard shortcut can be changed in the **Preferences** menu.

To Open the DashBoard File Navigator

The File Navigator view shows a subset of the file system on your computer.

1. From the DashBoard menu, go to **Views > File Navigator**.

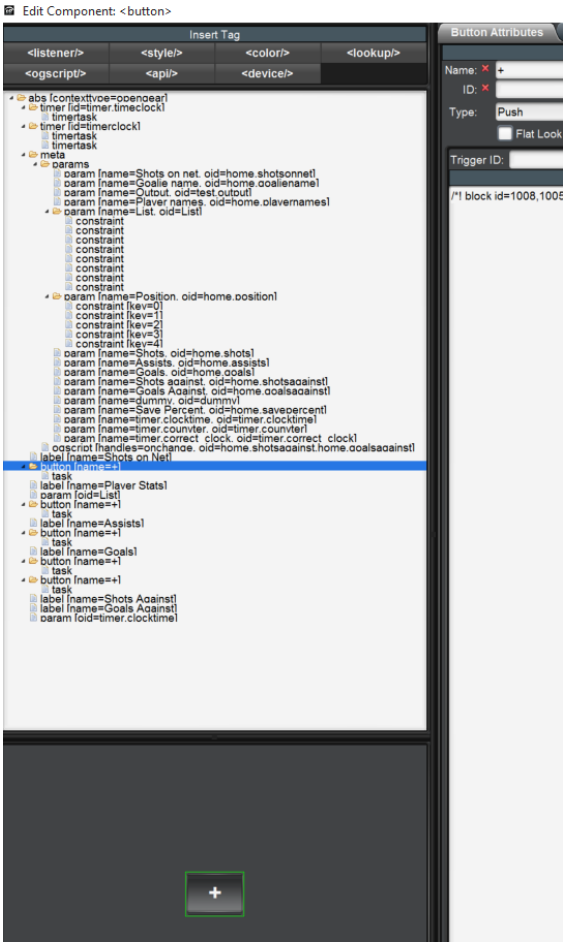


2. To have a directory appear in the **File Navigator** use the small green '+' button below the File Navigator tab:

Any folder you choose along with its subfolders will now appear in the file navigator.

The File navigator is often the primes method used to open CustomPanel files. It displays both **.grid** files and **.ogd** files (open gear devices) that can be opened from the navigator.

Navigating the Custom Panel Tree



Your Custom panel is an XML file.
If you double click on any element of your panel while in edit mode you are taken to an editor with the element you selected highlighted in the tree on the left and the editor for that element on the right.

In the example to the left, the Button was double-clicked and you can see the editor has that line highlighted in blue.

To view the attributes for other items you do not need to close this window and double click on a new item. New items to edit/view can be selected by clicking on elements within the tree.

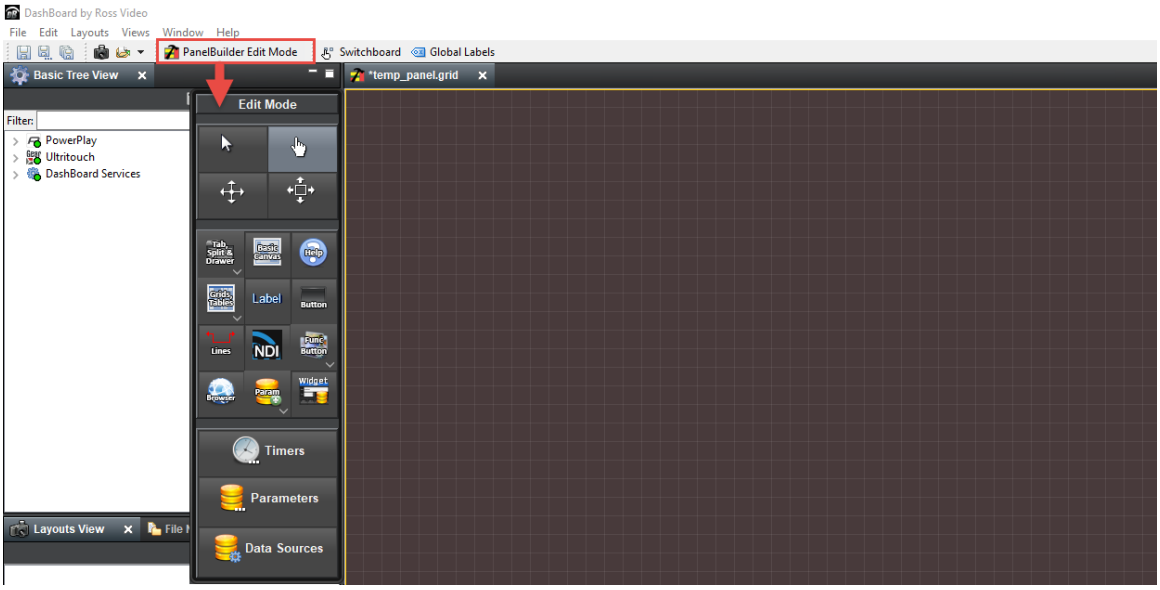
An Introduction to PanelBuilder

This section describes how to use PanelBuilder to customize your custom panel. This includes, creating menu tabs or split panes, adding images, creating links, and drawing lines.

To Use PanelBuilder Edit Mode

To edit or modify CustomPanels the CustomPanel must be in PanelBuilder edit mode. You can use the **PanelBuilder Edit Mode** button beneath the DashBoard upper menu to change in and out of the editor mode.

1. Click **PanelBuilder Edit Mode** or the **Ctrl + G** keyboard shortcut to begin editing the CustomPanel.



When you are in PanelBuilder Edit Mode, the left toolbar is displayed on screen.

2. Use the **Edit Mode** toolbar to build the CustomPanel elements.
3. When you wish to view the result, you can exit the editor by clicking the **PanelBuilder Edit Mode** button or pressing the **Ctrl + G** keyboard shortcut.

Tip: The shortkey is named **Ctrl + G** because it is short for “grid” mode, and the Custom panels have a .grid file extension. You can remap keyboard shortcuts in the **Window** menu, under **Preferences > Keyboard Shortcuts**.

To Create Menus Tabs or Split Panes

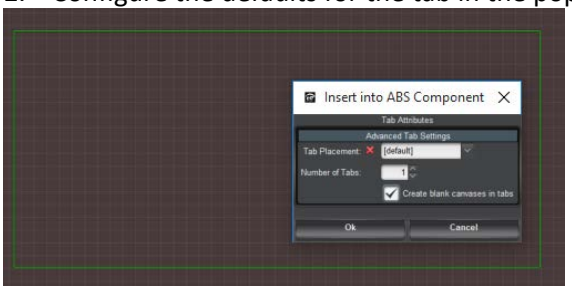
The **Edit Mode** toolbar allows you to add **Tabs** and **Split panes** to your panel. To access it choose the **Tab, Split & Drawer** item from the **Edit mode** toolbar, this will change the bottom of the toolbar to show the expanded choices available.

1. Add the **Tab** menu to your custom panel

To add a Tab select the Tab icon. Now drag and area in your custom panel where you want the tab to appear.

Note: Don't drag the icon *from* the Edit mode toolbar, drag a region on the CustomPanel that is the size that you want the Tab menu to be.

2. Configure the defaults for the tab in the popup.



The 'Tab' placement question allows you to choose where the tabs appear, the default is at the top.

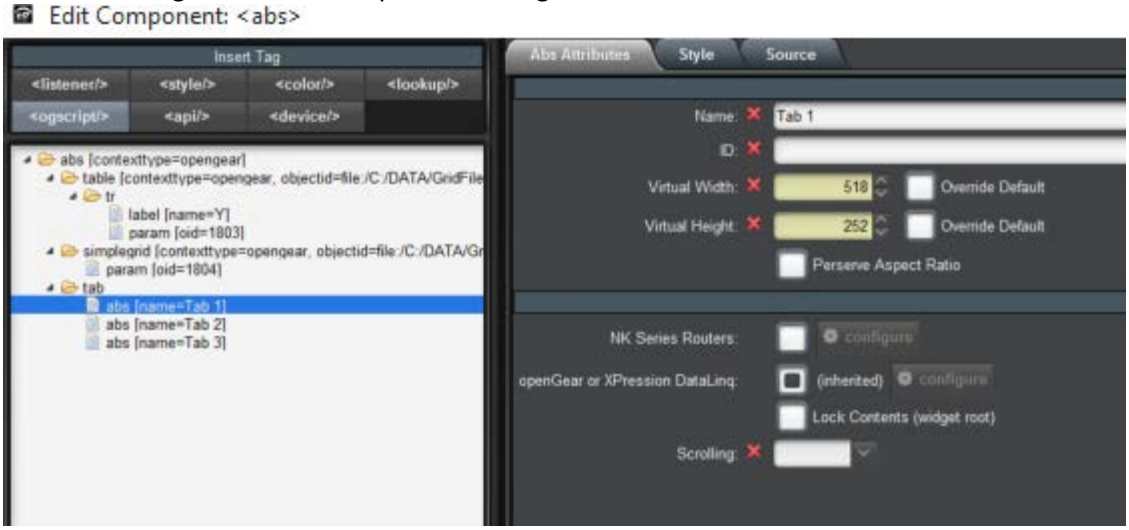
Note: The 'middle' choice for tab placement is special. This will create a set of tabs where the tab selector is hidden. This is very useful when you want to have several UI menus in the same location on screen. Changing tabs will be done programmatically using the scripting language

instead of by the user selecting a tab.

3. Editing and configuring the tabs

Example showing 3 tabs selected at the bottom:

Double-clicking on the menus open the configuration UI.



Important Note:
You can change which tab is selected by clicking in the Custom Panel tree on the left.
For tabs there is one node per Tab. Make sure you have the Tab you want to modify selected (as shown in the image above).
If you would like to modify the overall menu then select the higher level node marked ‘tab’ in the image above.

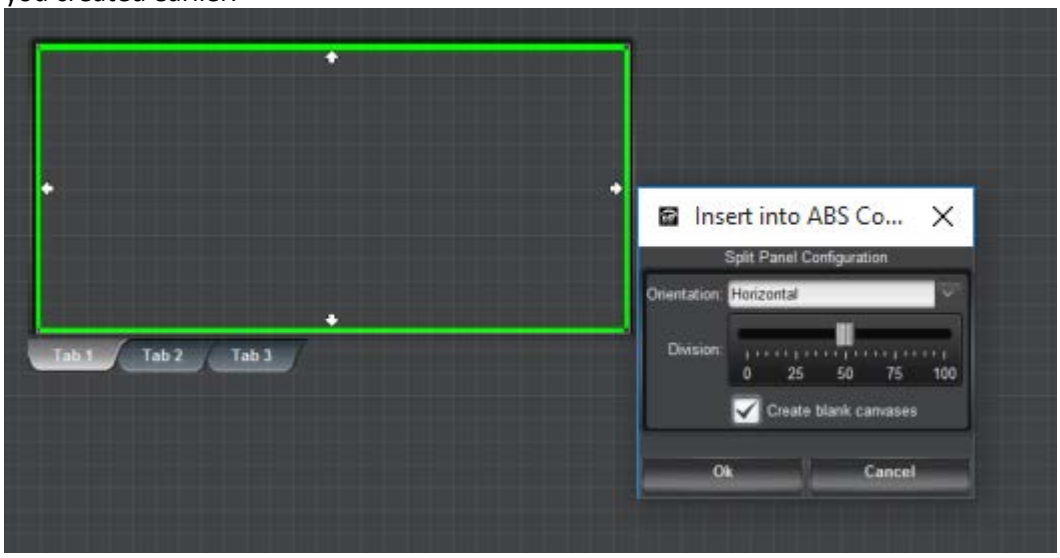
The ‘Name’ Field will be the text shown in the UI on the tab.
The ‘ID’ Filed is how you will reference this tab through scripts in your panel.

Important Note: This use of the ID for script access is throughout all features in Custom Panels, get used to it!

4. Nesting of UI objects and experimenting with Split panes. UI elements can be put inside of

each other.

To see an example of this select the Split pane tool and drag a region inside one of the tabs you created earlier.



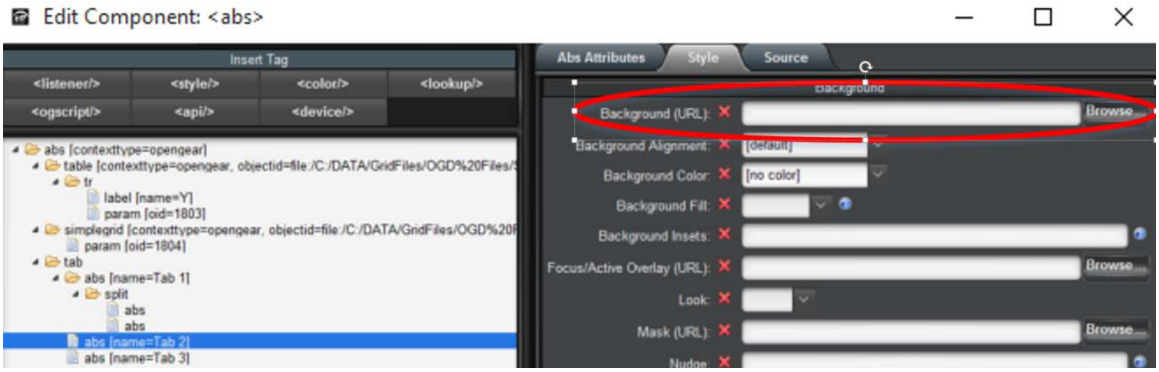
Note: The arrows in the image above indicate that the new region will go to the edges of the region it has been placed within.

Change the tabs in operators mode.
You will notice that one of your tabs has the Split pane but the others do not.

To Insert Images (with or without alpha masks) and Backgrounds

Images are very important in CustomPanels. They can make your CustomPanels more attractive and can also help operators understand the functions of UI elements better.

1. To add an image, in **Edit Mode**, double click on the element you want (or navigate the Custom Panel edit mode tree to that item).
2. In the **Style tab** you modify the **Background (URL)** item:



3. Click the **Browse** button to select an image that's on your system.

Important Note:
If you select an image in the same directory or a sub-directory of your .grid file then it will be 'relative' this means you can move the .grid file and its sub-directories/files to other computers and the image will still be found.
If you select an image that is NOT then it will be absolute, meaning if you move this .grid file to another computer it will be looking for that image in the exact directory you pointed it to initially. It is generally recommended that you use relative images.

Example of putting an image as the background within a tab:

There are many other locations images can go:
-Background for the entire panel (select the **Style** tab for the top level node in the CustomPanel tree)

-Buttons

-Text labels, the following is showing a text label with the words 'Text Label' using an image behind it:

-Masks: an image file may have a mask (sometimes called an alpha) signal. This allows the image to have a shape so that it is not simply a rectangle, so that when you place an image over top of a background it merges correctly.

Examples:
Without proper Mask/Alpha:

With proper mask/Alpha:

Advanced tip: It is possible to modify any style element using script, this includes changing what images are used on a UI element.

To Create Links Between CustomPanels

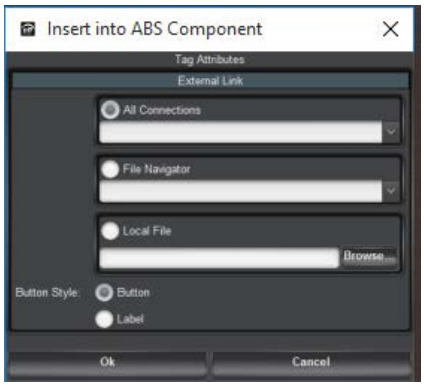
Custom panels can open other Custom panels.
This can be very useful for making a set of Custom panels which point to each other and can therefore be part of a large menu system.

Links can be put on buttons or on labels.

1. To add a link, select the **Func Button > Panel Link** item on the **Edit Mode** toolbar.

2. Now drag the location on your custom panel you would like the link.

There are several types of links that you can add:



The **All Connections** link allows you to open up the editor for a specific product from the tree view.

The **File Navigator** allows you to open up an item from your File Navigator (typically another Custom Panel).

The **Local File** allows you to open up a file from anywhere on your computer (typically another Custom Panel from a directory not in your File navigator path).

Note:
You can run DashBoard so that the editors and CustomPanels are in Full Screen mode. Use the **Shift + F11** shortkey on your keyboard. In full screen, the user will not have access to the Basic Tree Views, File Navigator, and etc…
This Panel Link feature allows you to give the user access to a set of hard coded Editors and Products without having to give them access to all of DashBoard.

Tip: Try it out by making two CustomPanels with a link to each other. Also, try running it in **Full Screen** mode.

Note:
This is where you may want to use the Embedded Editor Feature from earlier, you can place a products entire UI in a section of the CustomPanel, leaving some space for a set of buttons which open up other CustomPanels and editors, all while staying in **Full Screen Mode**.

To Create CustomPanel Links to Embedded Browsers

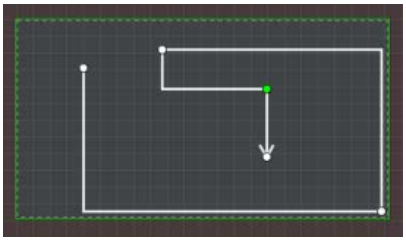
DashBoard allows for the placing of a web browser, pointing to a specific URL inside of a Custom Panel.

The web browser used is the default web browser on the computer.

This can be useful for getting access to product interfaces that are only available through a web

page.

To Draw Lines in a CustomPanel



To show the flow between elements you may want to draw lines between them.

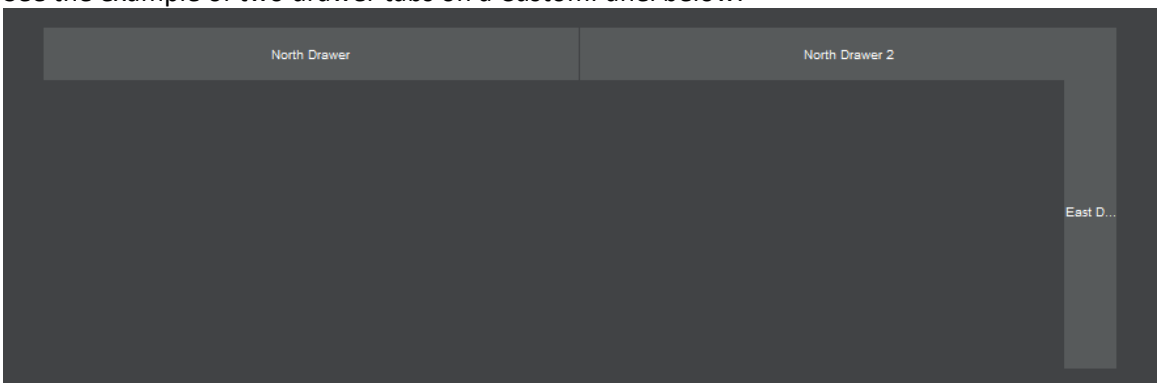
1. Select the line tool in the edit palette.
2. Draw the region in which you want the line to appear.
   You will be given a two-point line with an arrow at one end.
3. To move the line ends around hold down the **Ctrl** key and select the point you want to move, you can now drag that point around. (The selected point you are modifying will be highlighted).
4. To add line segments, holding the **Ctrl** key down double-click in the line region.
   The line itself can have its attributes adjusted, double click in the line region to get the line attributes UI. (but, *not* while holding the **Ctrl** key down. ☺)

To Create Drawers for CustomPanels with Limited Space

In **PanelBuilder Edit Mode**, you can use the Drawer button 

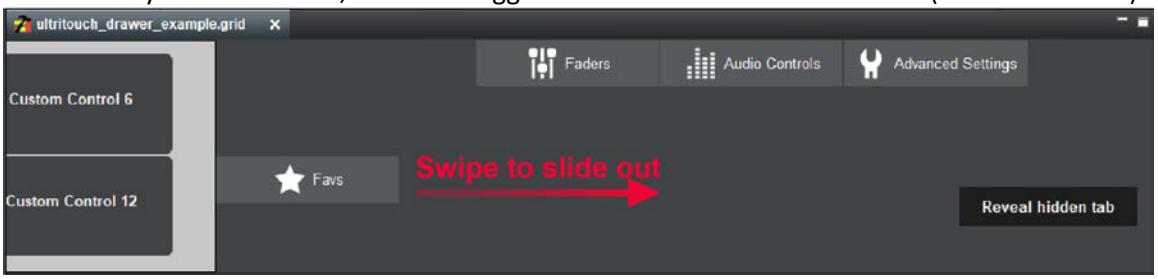
to create drawer tabs for CustomPanels with limited space.

See the example of two drawer tabs on a CustomPanel below:



1. In PanelBuilder Edit Mode, click **Drawer**, and click and drag the rectangle into the shape of your drawer canvas. Set the tab fill to **Both**, **None**, **Horizontal** or **Vertical**, depending on the type of tab you want. **Both** is most commonly used to cause tabs to automatically evenly adjust the tab size based on the number of tabs. None allows you to set the Tab **Width** and **Height** manually.
2. Now on the Edit Mode toolbar, select **Basic Canvas** to draw a rectangle in the space you would like the drawer to occupy. Typically this will start from the side you want to anchor it to, and then as far as you would like the drawer tab to pull out to. Once drawn, set the anchor to the side you want the tab to appear on (top, bottom, right or left).
   Tip: Do **NOT** use the **Drawer** button to add drawer tabs (always use the **Canvas** button).

3. Refresh with (Ctrl + F5) and then add more drawers if you like.
4. When you're done adding drawer tabs, you can add a basic canvas that is anchored to the center, and then you can build your main CustomPanel area without having tabs interfere.
5. To add content to the drawer areas, toggle out of PanelBuilder Edit Mode and pull out the drawer you want to edit, and then toggle back to PanelBuilder Edit Mode (as shown below).



Here's a more advanced example (with icons included and hidden tabs):

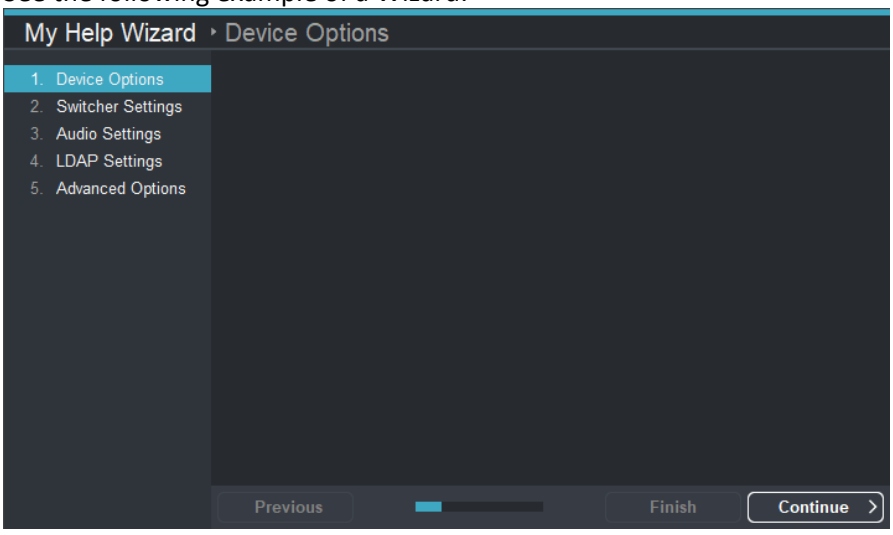
Advanced Tip: If you need to add more tabs later, it's easier to duplicate and modify existing tabs in the source code for the drawer.

To Create a Wizard

In **PanelBuilder Edit Mode**, you can use the Wizard 

button to create a step-by-step wizard. You can create wizards that contain a title, a page navigation pane, and a progress bar. You can choose which features you would like to be visible, and how many pages appear in the wizard. Feature options include the tabs, progress bar, and dialog width and height.

See the following example of a Wizard:



1. In the **PanelBuilder Edit Mode**, click **Tab, Split & Drawer** > **Wizard**. Then click to draw the shape of the wizard on the canvas.
2. Set the following fields: Name, ID, number of pages, and check off any features you would like visible. (It is recommended that you create the wizard NOT in Dialog mode, and later change it to Dialog if you want it to appear as a separate dialog.)
3. Click **Ok** and add your content.

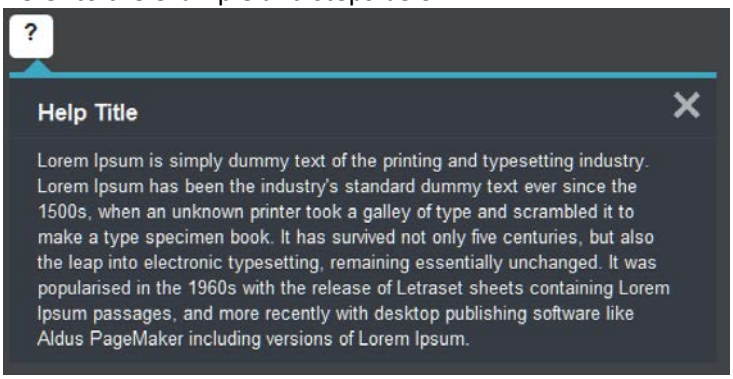
Advanced tip: If you would like to modify the basic wizard, you can customize it using script functions. (This is not covered here, but you can refer to the *DashBoard User Guide* and *DashBoard CustomPanel Guide* for more details).

To Create a Help Control Popup

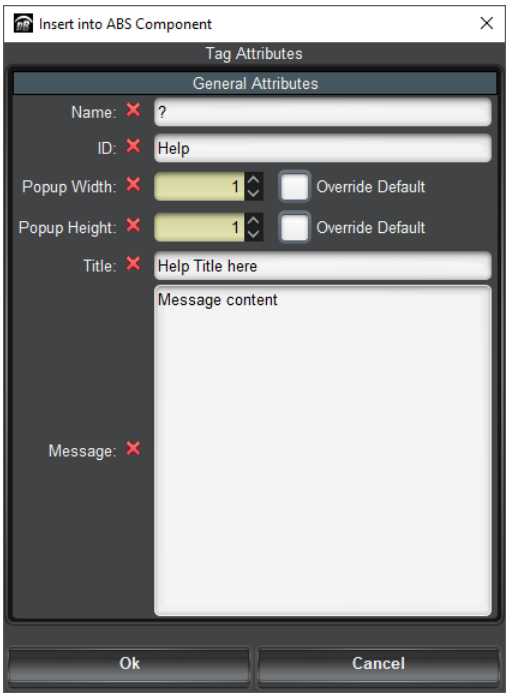
In **PanelBuilder Edit Mode**, you can use the 

Help button to create a help text pop-up. This includes a basic canvas container, and in the Tag Attributes you can set the width, height, title and message. Messages can be plain text or html with many common html tags supported including hyperlinks.

Refer to the example and steps below:



1. Click the **Help** button and draw the shape of the button on the canvas.
2. Set the following fields:


Insert into ABS Component
Tag Attributes
General Attributes
Name:
?
ID:
Help
Popup Width:
1
Override Default
Popup Height:
1
Override Default
Title:
Help Title here
Message content
Message:
Ok
Cancel

3. Click **Ok**.

An Introduction to Tables (Mostly simple grid)

A table allows you to define a grid of cells to put UI elements into.

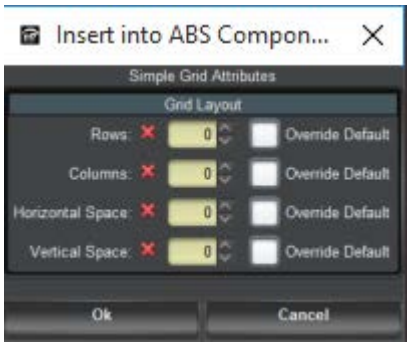
There are several types of tables that DashBoard allows:

- **Tables** (fairly complex, but required for some uses)
- **Simple Grids** (the one most frequently used)
- **Wrap Content** (Does not keep a static set of cells or sizes, but changes based on the content)
- **Scripted tables** (the most flexible, but since there is no UI is available to create these it's an advanced topic)

To insert a table select the **Grids,Tables** button on the **Edit Mode** toolbar. The lower portion of the Edit Mode toolbar allows you to select the table type. (Note: there is no UI to create Scripted tables at this point)

To Create a Simple Grid

1. *Select the Simple grid and drag the region in the Custom Panel you want it to appear on.*
   You will get the following popup:



The Rows and Columns items define the number of Cells that appear in your table.
The number of cells will be fixed as your resize the table. The size of the cells will stretch as you resize the table. (Don't forget to click the 'Override Default' item if yo uwant to set the values).

The Horizontal space and Vertical space items allow you to put space between the items in your cells so that they don't have every UI control beside each other to the pixel.

Example: Configuration using some vertical sliders:

Note:
There are some interesting Style elements going on above.
The table itself has been set to have a Thick red border and a background image.
The individual sliders have a background colour set.

Colors also have a transparency portion, which is why you can see the tables background image through them.

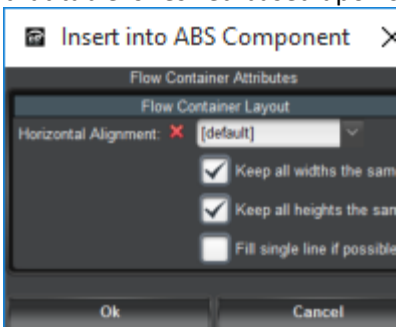
Note: *Parameter Constraints and Simple Grids*
We haven't talked about Parameters yet, so, remember this for later!
It is possible to create a parameter that presents the user with a choice.
Example: vid_format: Choices are (1080i59.94, 1080i50, 720p59.94,720p50, 480i59.94, 576i50)
If we want to present this to the user in a set of buttons / radio buttons we would want them to nicely fill a region.

To do this, create a ***Simple Grid****, override the number of rows or columns or both you want. Place that one parameter in the Simple Grid. DashBoard will automagically extract the choices from that parameter and give a nice UI for it.*

Example of this:

To Create a Wrap Content Table

The **Wrap Content** table allows you to have elements in your table change where they appear as that table is resized based upon some rules you build into the table.



When you drag an element onto that table DashBoard records the size of the item.

If you have 'Keep all widths the same' then DashBoard finds the largest width and makes all items in the table have that width. (i.e. it makes smaller items larger to match the largest item).

The same applies for 'Keep all heights the same'.

The following table has 4 elements on it, when they were placed on the table they had different sizes but are drawn now as follows:

If we turn off those checkboxes we now get:

DashBoard starts drawing them using the size they were created at and simply starts a new row when it runs out of space on the current one.

If we make this table smaller now we now get:

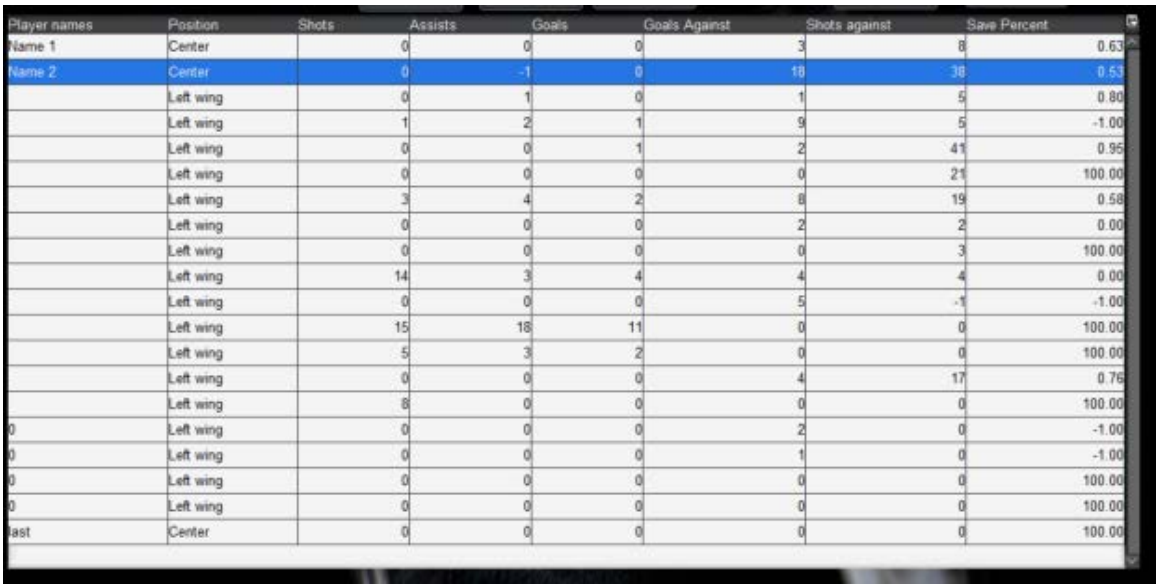
To Create a Parameter Based Table

Tables are a special UI tool that creates a grid of cells where:

- Each column represents a particular array
- Each row represents an index in that array.
- One of the rows is selected by the user.

Typically you would have all of your arrays be of the same size.
Columns do not have to be the same data type.

Example:



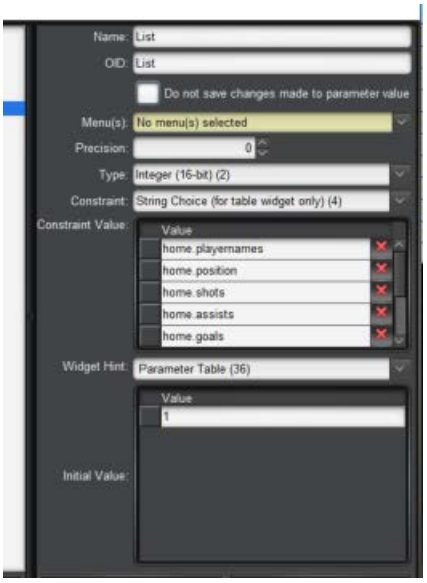
Player names	Position	Shots	Assists	Goals	Goals Against	Shots against	Save Percent
Name 1	Center	0	0	0	3	8	0.63
Name 2	Center	0	-1	0	18	38	0.53
	Left wing	0	1	0	1	5	0.80
	Left wing	1	2	1	9	5	-1.00
	Left wing	0	0	1	2	41	0.95
	Left wing	0	0	0	0	21	100.00
	Left wing	3	4	2	8	19	0.58
	Left wing	0	0	0	2	2	0.00
	Left wing	0	0	0	0	3	100.00
	Left wing	14	3	4	4	4	0.00
	Left wing	0	0	0	5	-1	-1.00
	Left wing	15	18	11	0	0	100.00
	Left wing	5	3	2	0	0	100.00
	Left wing	0	0	0	4	17	0.76
	Left wing	8	0	0	0	0	100.00
0	Left wing	0	0	0	2	0	-1.00
0	Left wing	0	0	0	1	0	-1.00
0	Left wing	0	0	0	0	0	100.00
0	Left wing	0	0	0	0	0	100.00
last	Center	0	0	0	0	0	100.00

The data above shows the basic stats for a hockey team.
Each column is an array.

In the above example there is a parameter called ‘Player Names’ which is an array (of 20) of strings in the first column.

The second row has been selected by the user and is highlighted.

Creation of this table is done through the parameters creation screen which we are looking at later. For the example above it looks as follows:



Note: The columns in the table are created by putting the OID name in the Constraint Value table (in the above picture it has home.playernames, home.position etc.. in it).

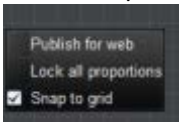
Note: the table OID is declared as an integer. The value of that table integer when the panel is operational will be the currently selected row number by the operator (starting from zero!). In the picture showing the UI the value of the parameter named 'List' will be 1. (the second row is selected, but, you count from zero).
This value will often be used in scripts as the array index into the parameters used in the columns.

Tips and Best Practices

Check out the following tips and best practices.

Snap to grid

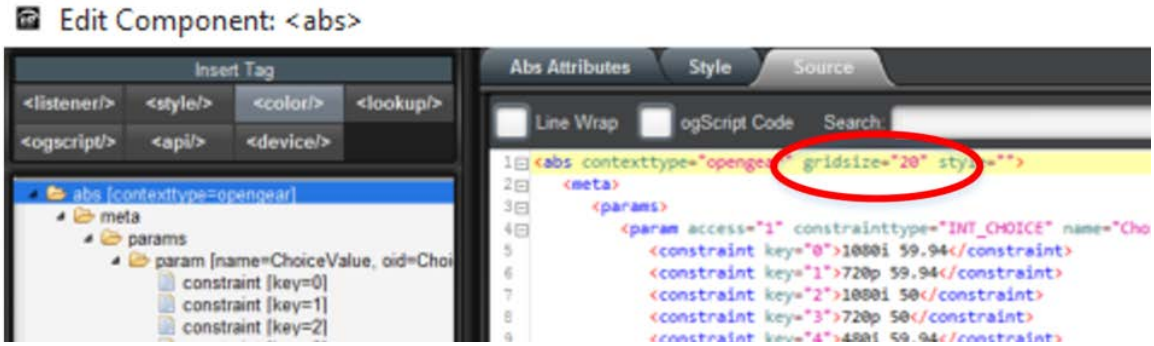
DashBoard can keep your elements lined up by aligning all UI elements to a grid as you move them around.
To enable / disable this right click on the background of your panel



The **gridsize** *is* editable, but, it has to be edited by hand in the source for the background of the panel.

To do so, make sure you have the top level node in the tree view of the custom panel and modify the item marked **gridfile** in the source tab as shown here: (which has the gridsize set to

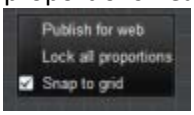
20 pixels)



Locking All Proportions

One of the big items to bear in mind when creating your custom panel is if it will be run on screens of different sizes, run windowed (i.e. not always full screen) etc..
When you resize your application you will notice that various items change size, move etc.. depending upon your anchor points (discussed later).

A simple method to have your panel resize usually the way you want is to use the 'Lock all proportions' feature.



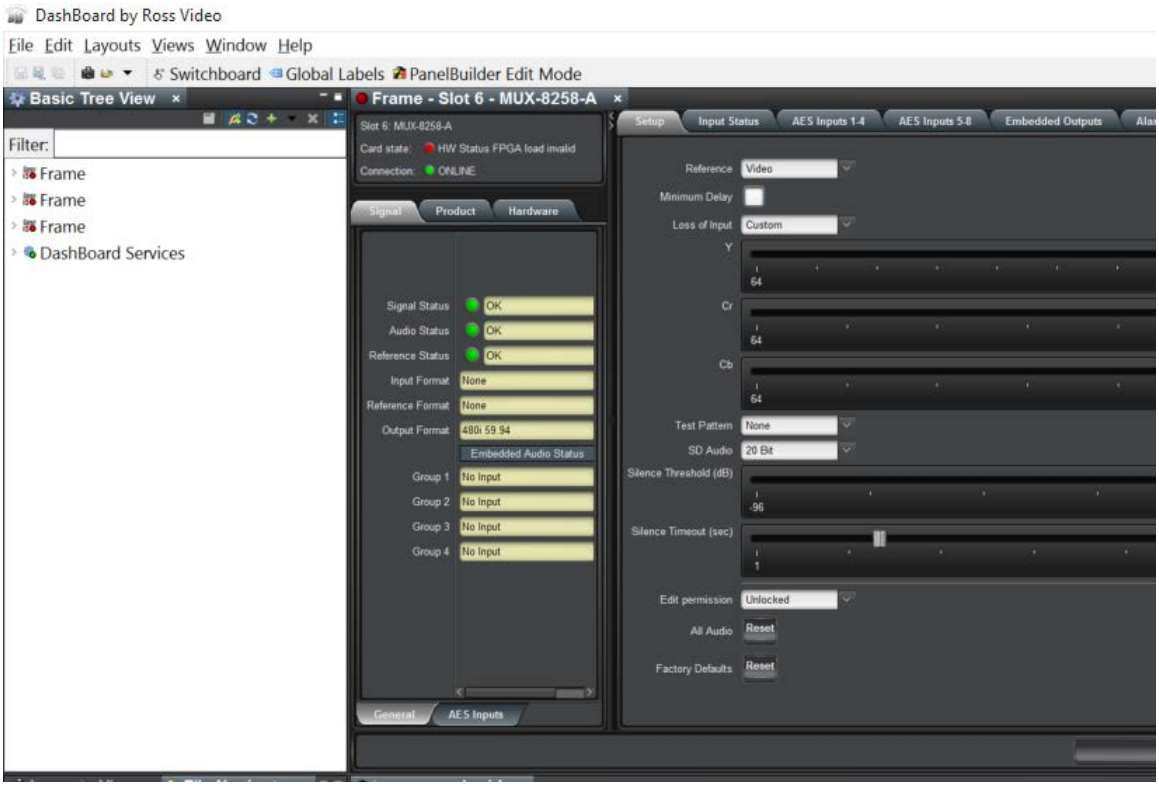
To do so, once you have your panel looking the way you like it right click on the background (while in edit mode) and select the 'Lock all proportions' item.

Now when your UI is on a different sized screen or windowed UI elements will resize to be in the same proportion as you currently have them setup.

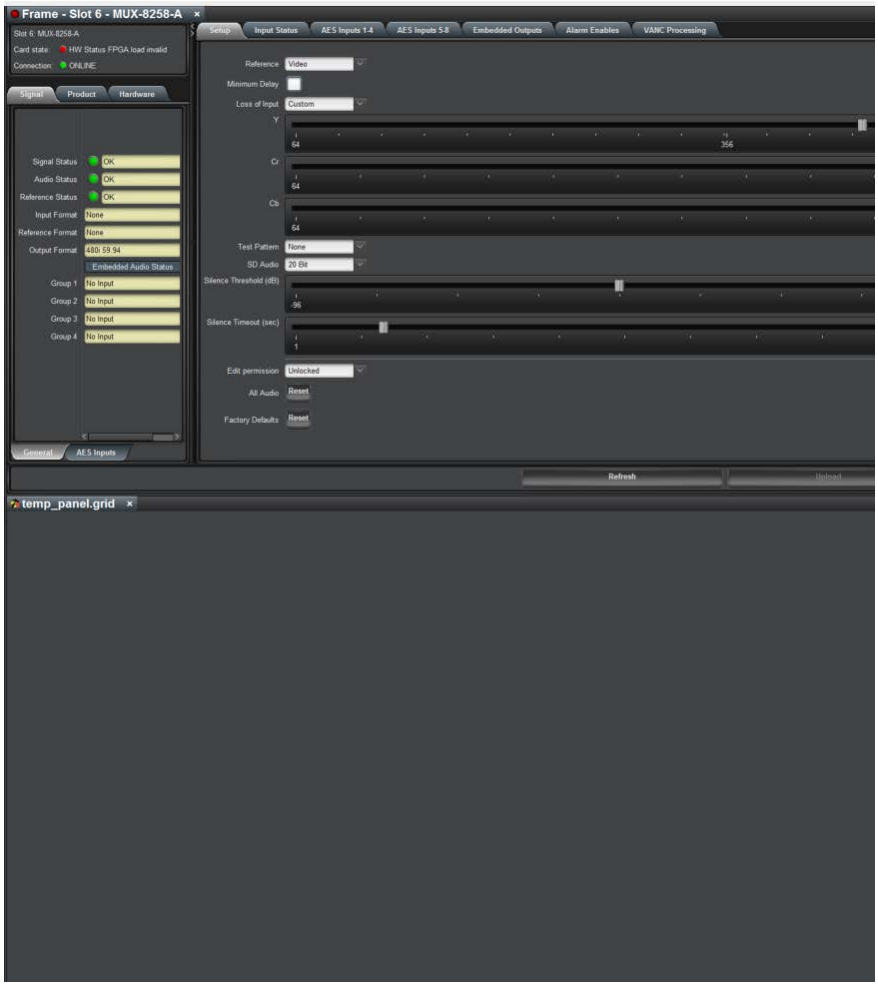
To Drag an Element from an Existing View

You can select and drag elements from an existing openGear device onto a CustomPanel. For example, you can drag a slider onto the CustomPanel, as shown in the expanded example below.

In the card in the screenshot below, if I wanted to drag the 'Test Pattern' parameter from the MUX-8258 card to a CustomPanel I would follow these steps:

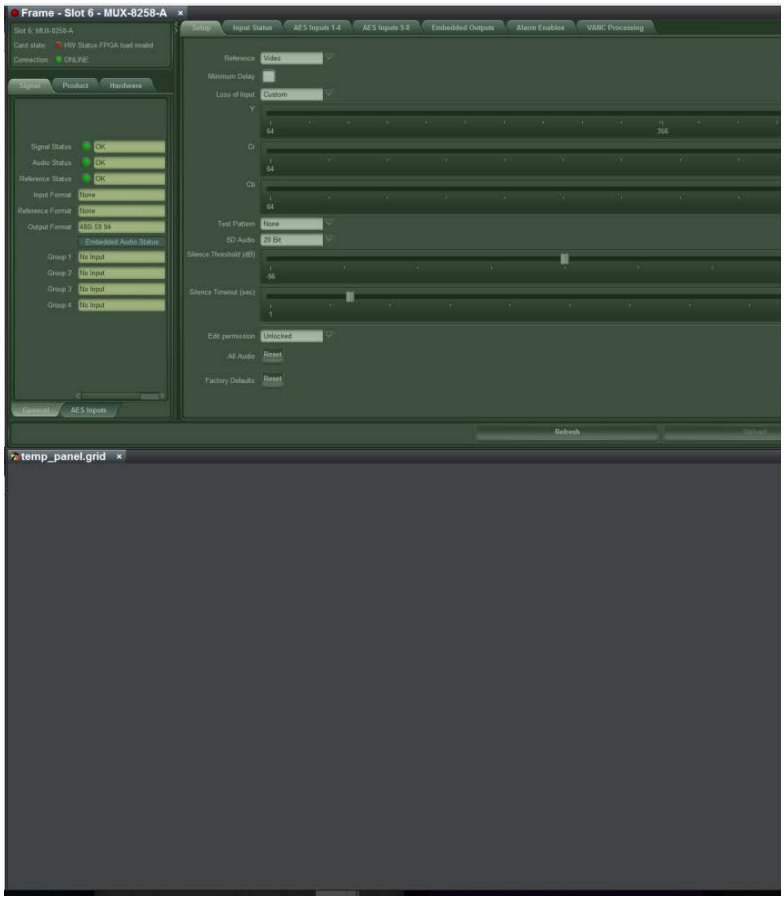


1. Arrange the editor to make sure both the CustomPanel and the UI element that you want to drag from visible on screen at the same time by. They are shown on the top and bottom here, but could easily be arranged side by side.

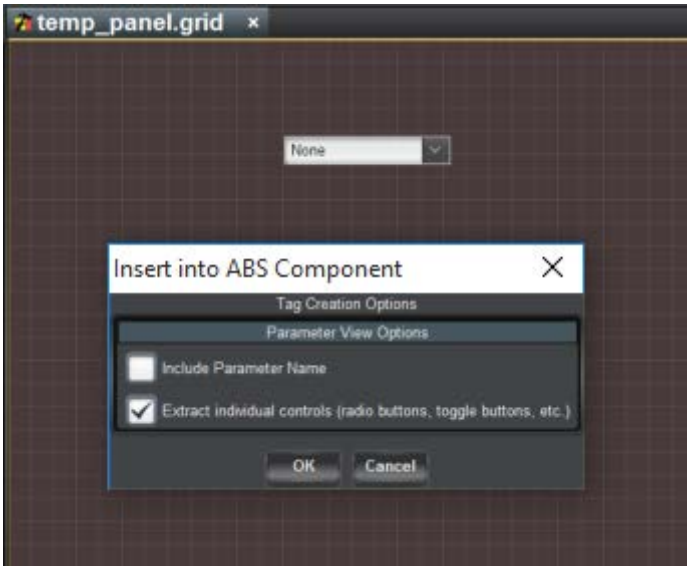


2. Take the editor you want to drag the control from into **Edit mode** (in this case, the MUX-8258 UI). Click anywhere in that editor and go into Edit mode. That editor will now be highlighted.

   Note: You will not see the CustomPanel edit toolbar because that toolbar only appears when you are editing a CustomPanel, not when you are in edit mode for a products' UI.



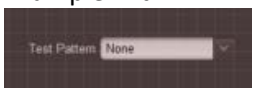
3. Select with the mouse the item you want and drag it onto the Custom Panel you would like.



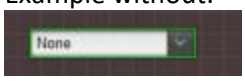
A pop-up appears.

4. Select 'Include Parameter Name' if you would like to have a label on your UI for this item. Refer to the examples below to see the options:

   Example with:



   Example without:



5. Select the **Extract Individual Controls** item.

Tip: You will 99.9% want to leave this checked, the only times this goes unchecked are for very unique UI layouts that we are not concerned about at this time.

6. Click **PanelBuilder Edit Mode** to exit Edit mode and enjoy your new parameter!

Note: The controls are now live in your custom panel, and any modifications you make to them will happen live on the actual device (so be careful!)

Tips:

- Parameters from as many products as you want can be dragged onto the same custom panel
- Entire menus can be dragged onto a custom panel, not just individual controls. Give it a try!

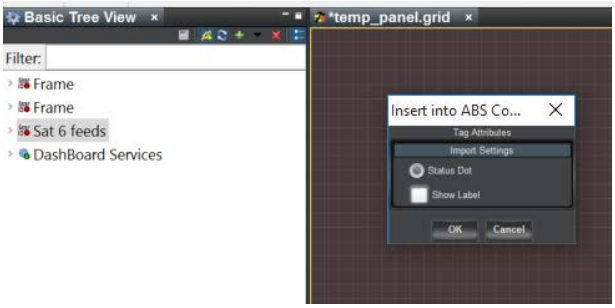
To Drag a Status Item from the Tree View

The basic and advanced tree views can have items dragged into custom panels from them.

1. Drag a Frame in.

Drag and drop a frame from the Basic Tree View on the custom panel.

Here, we have selected the frame named 'Sat 6 Feeds' and dropped it onto the Custom panel and we get:

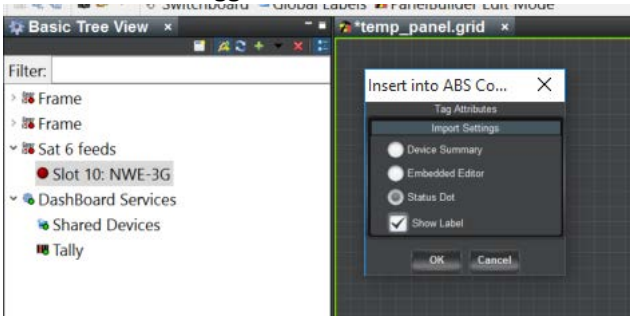


You will get an LED on your custom panel representing the current status of the frame, this LED will match the LED in the tree view.

Example:

Note: When your panel is operating the user can click on that led node to get a popup of all of the cards in the frame and their current status.
Selecting a card in this mode will popup the editor for that card.

2: Drag a Card in.
Here we have dragged slot 10 from the 'Sat 6 Feeds' frame in:



There are now some choices as to what appears.

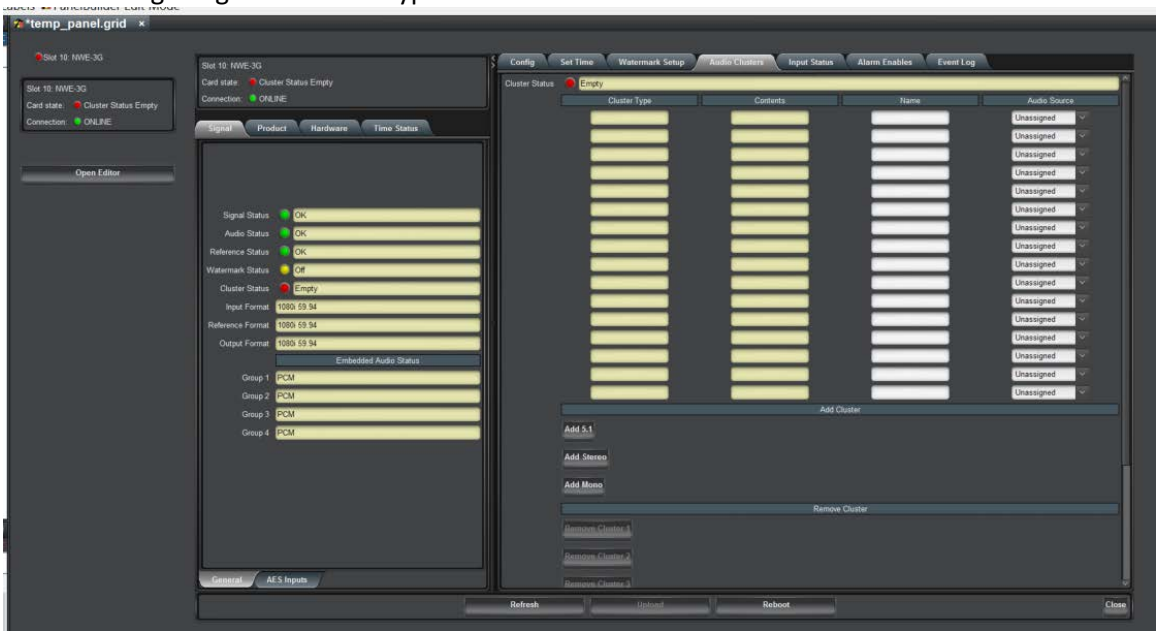
The Status dot item is like the one for the frame. An led showing the status of the card. Clicking

on this LED allows the operator to open the editor for that frame.

The Device Summary view gives a slightly larger version that shows the card status as well as the text for the current status

The 'Embedded Editor' puts in a region of the Custom Panel the regular Entire UI for that card. This UI is sizeable and moveable.
This can be a very convenient feature when you want to show the regular UI for a card but you would like to put some additional controls and features around that UI.

The following image shows all 3 types on screen at the same time.

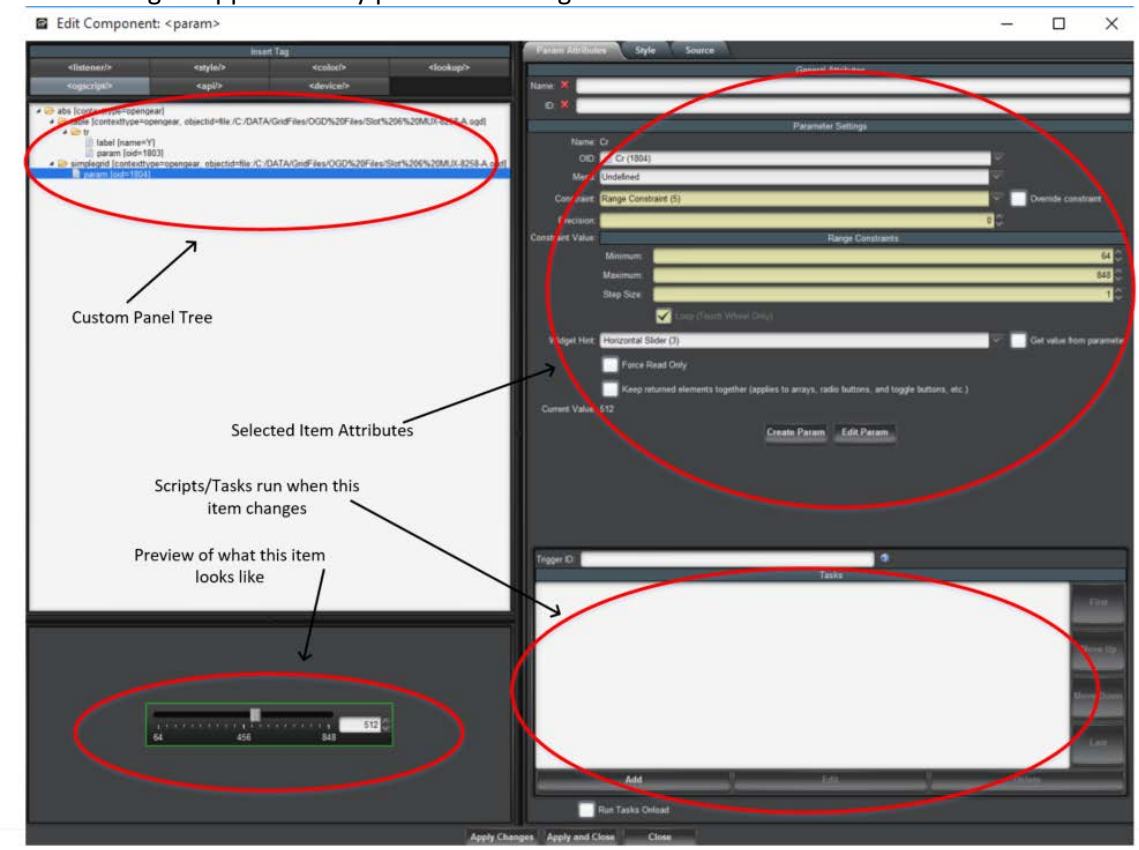


To Edit a Dragged Item from the Tree View

When a control from a product/card is dragged onto your custom panel it will look and operate just as it did in the original UI. Every element about this item can be modified in your custom panel.

1. To edit an item on your panel, go into **Panel Builder Edit Mode**. Double-click on the item you would like to modify.

The following UI appears in my panel for editing a slider control:



For this section we will be looking at the upper right portion the ‘Selected Item Attributes’. Notice that there are multiple tabs:

‘Param Attributes’ these are the global values for the selected parameter.
‘Style’ This changes how the parameter is drawn on the UI, colours, font size etc…
‘Source’ At the end of the day, everything in a Custom Panel gets written to the .grid file as XML, you can view and edit the XML source in this tab. Some features are only available through editing the source directly.

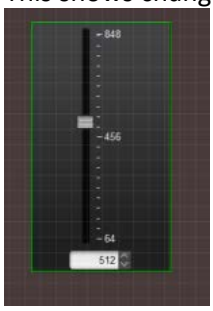
Param Attribute items to focus on:
Widget Hint:

This tells the Custom Panel what widget to use to draw the value on screen.
Try changing this for your item.

Note: after changing the widget you may very well have to change the screen region your

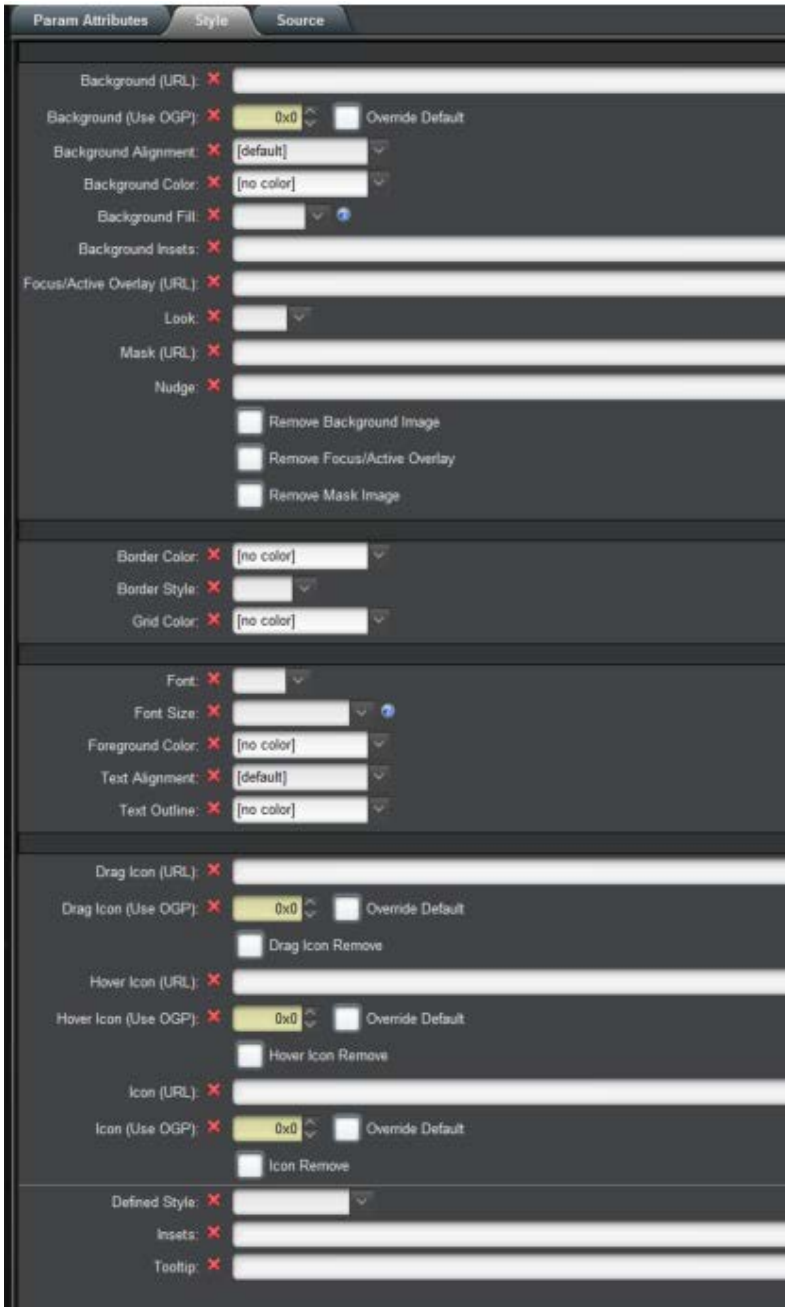
parameter is using to draw nicely.

This shows changing that slider to being vertical instead of horizontal:



Force Read only:
Allows you to set a parameter as read only instead of editable in a panel.

Style Tab:

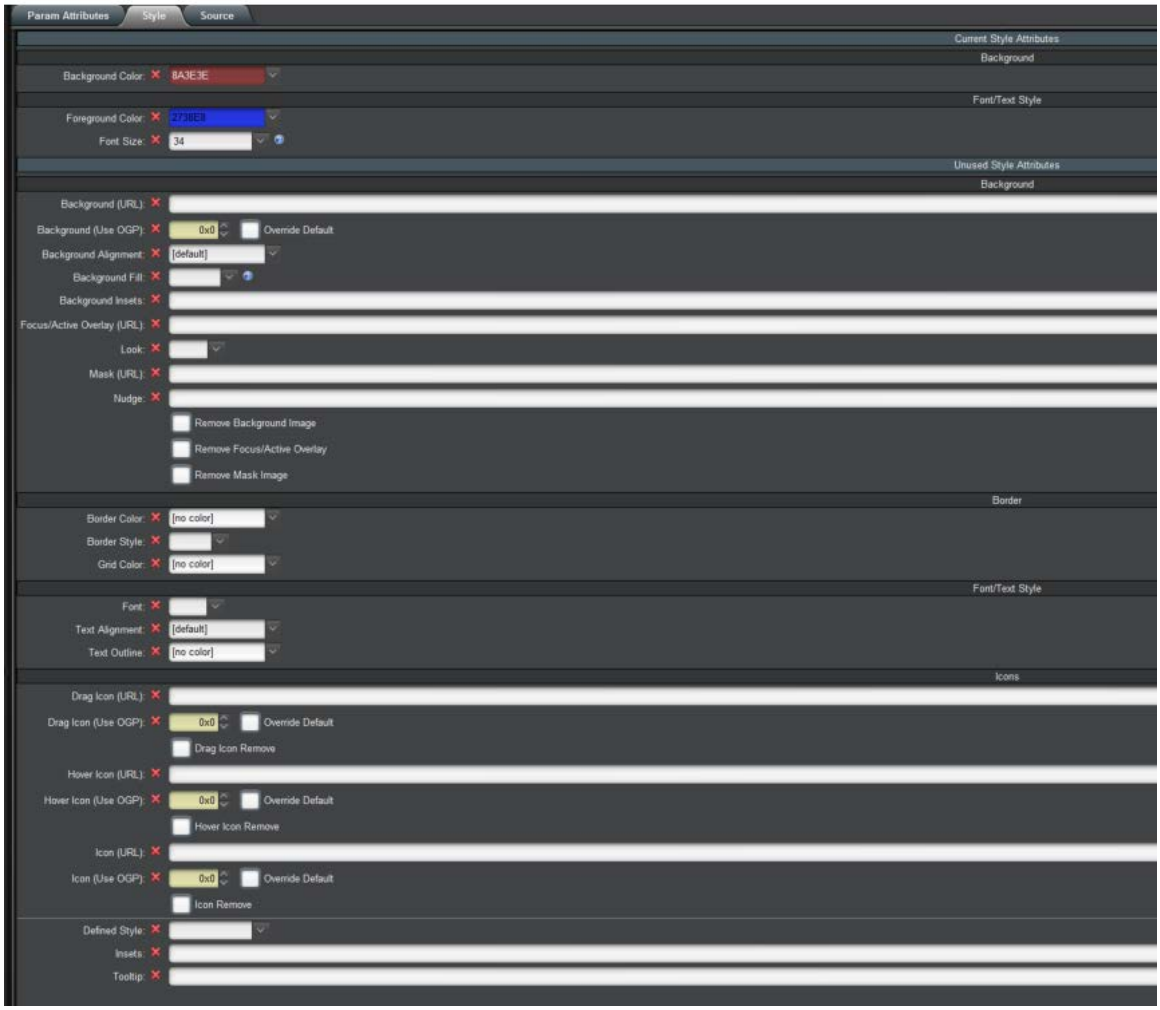


There are a lot of style changes that you can make.

We will not go into every item in detail, but, some important items should be pointed out here:
-In the image above, NO style elements have been added, notice that the values are all

blank.

-If a Style element is used, its position in this menu is moved to the top, this allows you to at a glance see which style elements have been set, the following image shows a Style page for when the Font Size, Foreground Color and Background Colour have been set, notice they are all at the top of the UI page:



-Clicking on the red ‘X’ clears that style element

-Font size has a dropdown listing some sizes such as ‘Normal’, ‘Small’ etc.. You can also type a number here directly to get a particular font size.

-Background URL. Most every element can have an image attached to it. Use of images in Custom Panels is highly encouraged as it makes the panels much more ascetically pleasing as well as helping understand functionality better.

-Defined Style, this will be covered later in training, suffice to say Defined Styles will save you a *lot* of work. Use them!

Source Tab:
Example Source code for our selected parameter:
<param expand="true" oid="1804" showlabel="false" style="size:34;bg#8A3E3E;fg#2738E8;" widget="slider-vertical"/>

Generally, you will not be editing this source by hand for most features as GUIs have been provided for you that edit/create this source for you.

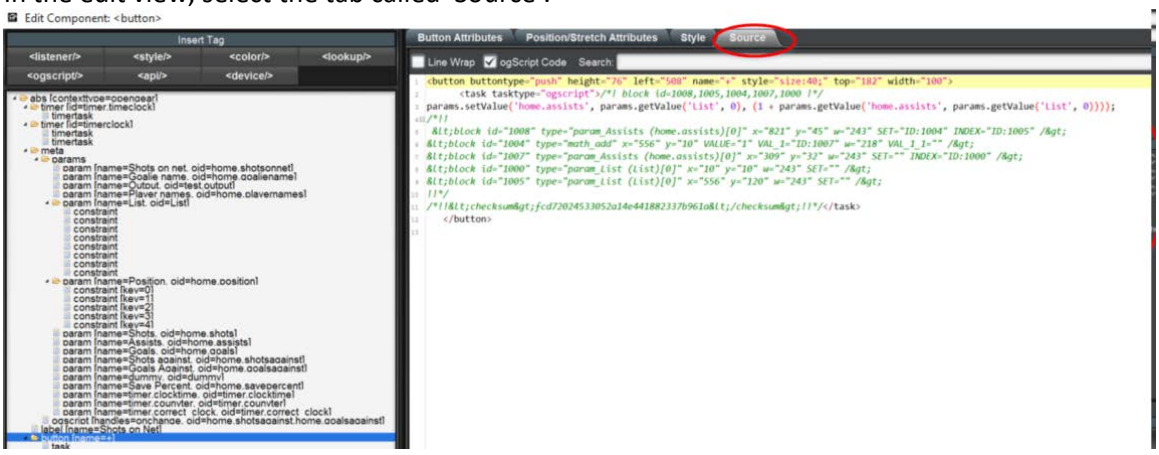
Viewing the Source Code for CustomPanels

Custom Panels are XML files. The editors that you work with are graphical front ends that modify that XML source files.

Advanced creators of CustomPanels may prefer in some situations to modify the XLM text directly.

Some new/advanced features may not yet have graphical front ends to create that modify those text elements in which case editing the XLM file directly will be the only method to access those features.
To view/edit the source directly for the file or an element get into the editing tool by double clicking on a panel element as normal.

In the edit view, select the tab called ‘Source’.



The example above is showing the source code for the selected button.

The source code shown will for every element in the Custom Panel for the selected item in the Tree view and lower. If the top level item is selected the entire source code for the Custom

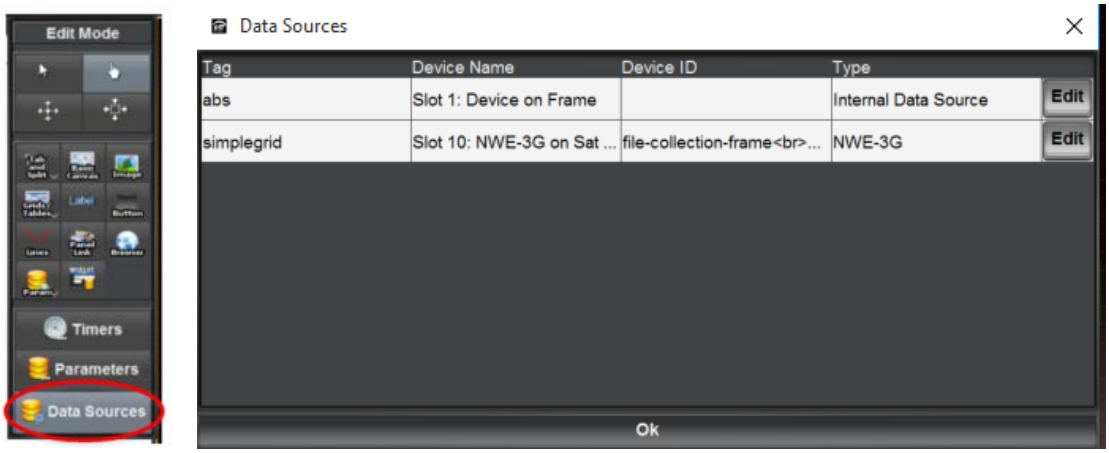
Panel is shown to you.

Data sources, pointing your panel at different cards / frames / products

When Creating a Custom Panel you may not be working at the final location of the Custom Panel with the exact equipment that the Custom Panel will be controlling.

For example, Ross Video employees may create a Custom Panel to control your Carbonite at our location. When that Custom Panel is built it will be done using our Carbonite in our lab.
When this Custom Panel is deployed to a customers location, it will be used on a different physical Carbonite at a different IP address.

When an item on the Custom Panel is created it has a reference in it telling it *which* physical Carbonite, Gear Card, etc… it is to control. These can be viewed by selecting the 'Data Sources' button in the edit pane.



In the above example the panel has two data sources:
-Internal Data Source (which is the parameters you created on this Custom Panel)
-Slot 10: of the NEW-3G product (an openGear card in an openGear frame).

To have this panel work exactly as it was created but using a different physical gear card simply press the 'edit' button on the row of the device to change.

Note: it is possible to change where the parameters in a file are stored / pointed to, but, changing this after you have created your Custom Panel for Internal Data Sources would be a very rare thing to do!

Advanced Parameter Note:
You can set a panel to not save itself back to your data files.

There are two reasons for this:

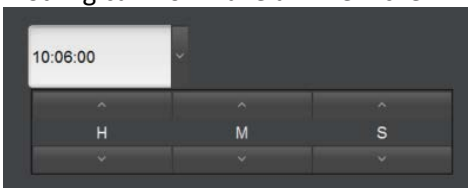
-This keeps your panel marked as not 'dirty' for things like clock timers

-This allows you to control that some changes are not to be permanent.

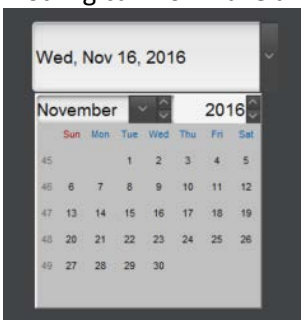
Simply set the checkbox 'Do not save changes made to parameter value' in the Edit Parameter UI

Some newer String widgets

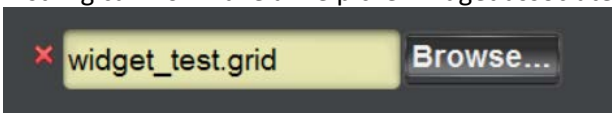
A String can now have a Time Picker Widget

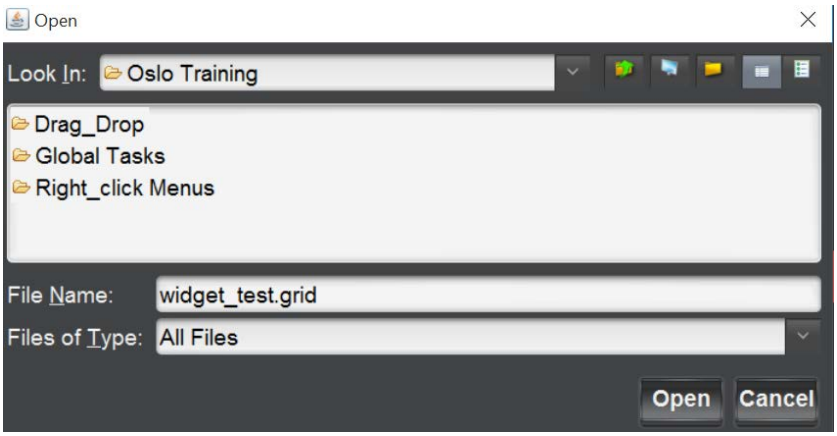


A String can now have a Time Picker Widget

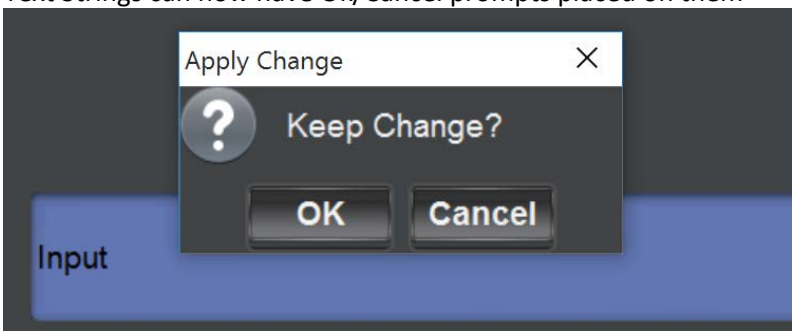


A string can now have a file picker widget associated with it

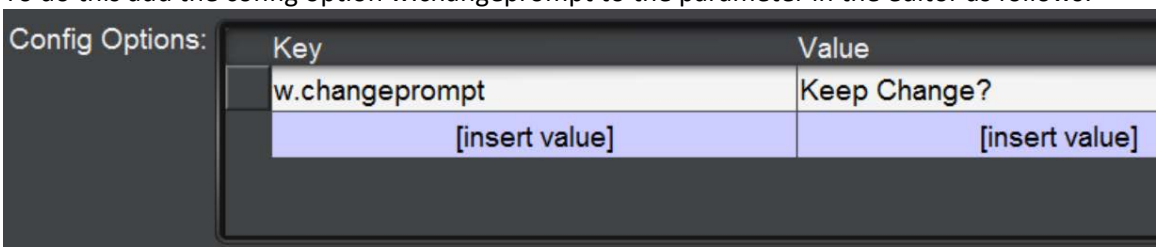




Text Strings can now have Ok/Cancel prompts placed on them

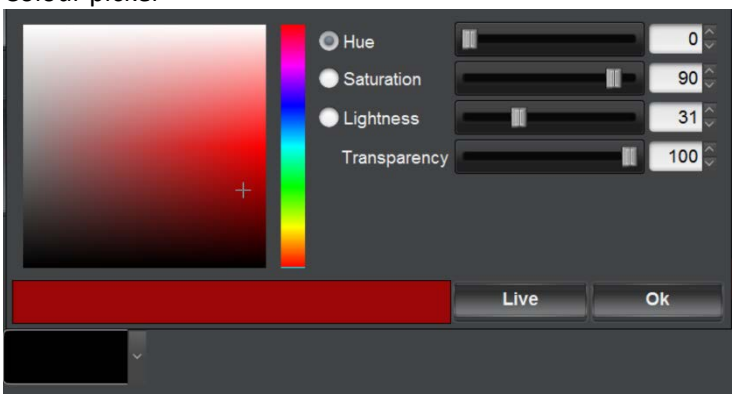


To do this add the config option w.changeprompt to the parameter in the editor as follows:



Some newer integer widgets

Colour picker



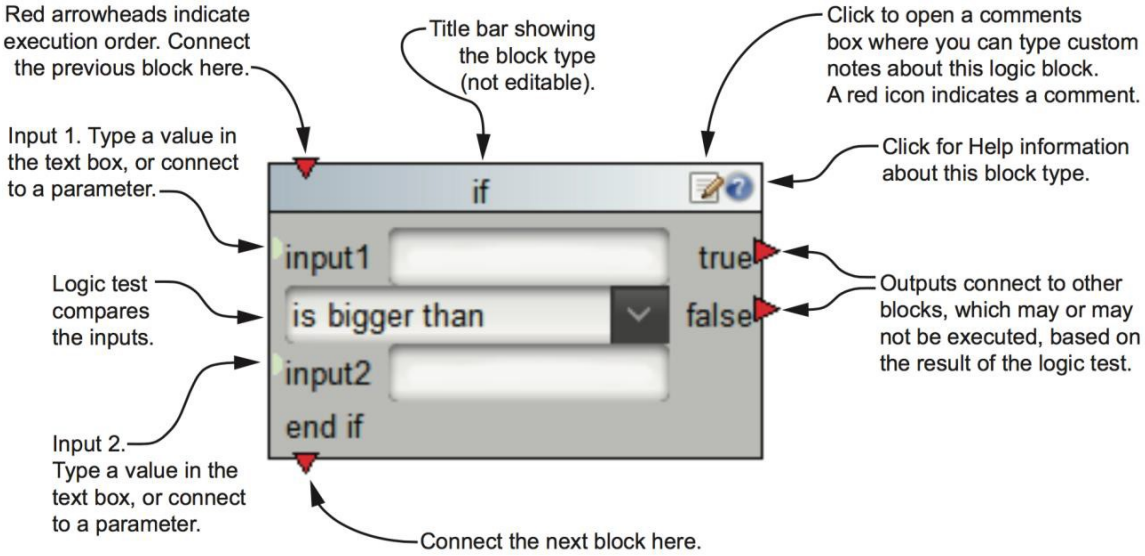
Widget Config options

An absolutely great new tool for CustomPanel creators!

Widgets have had additional configuration available by manually adding them to custom panels. For most users, knowing these exist as well as knowing how to add them has made these features unavailable to most creators of CustomPanels.
The Param Attributes tag now has an editor for all config options.

Example showing an instance of it for the ‘Date Picker’ widget:

The ‘?’ icon to the right of the ‘Config options’ table gives a popup that shows ALL of the options for each widget, a snippet of it is:


Red arrowheads indicate execution order. Connect the previous block here.
Title bar showing the block type (not editable).
Click to open a comments box where you can type custom notes about this logic block. A red icon indicates a comment.
Input 1. Type a value in the text box, or connect to a parameter.
Click for Help information about this block type.
if
input1
true
Logic test compares the inputs.
is bigger than
false
Outputs connect to other blocks, which may or may not be executed, based on the result of the logic test.
input2
end if
Input 2. Type a value in the text box, or connect to a parameter.
Connect the next block here.

CustomPanel Integrations

DashBoard CustomPanels integrate with XPression, Carbonite Switchers, openGear cards, other networked devices and more.

Integrating with XPression

The Ross XPression Character Generator has two main methods of integration with DashBoard.

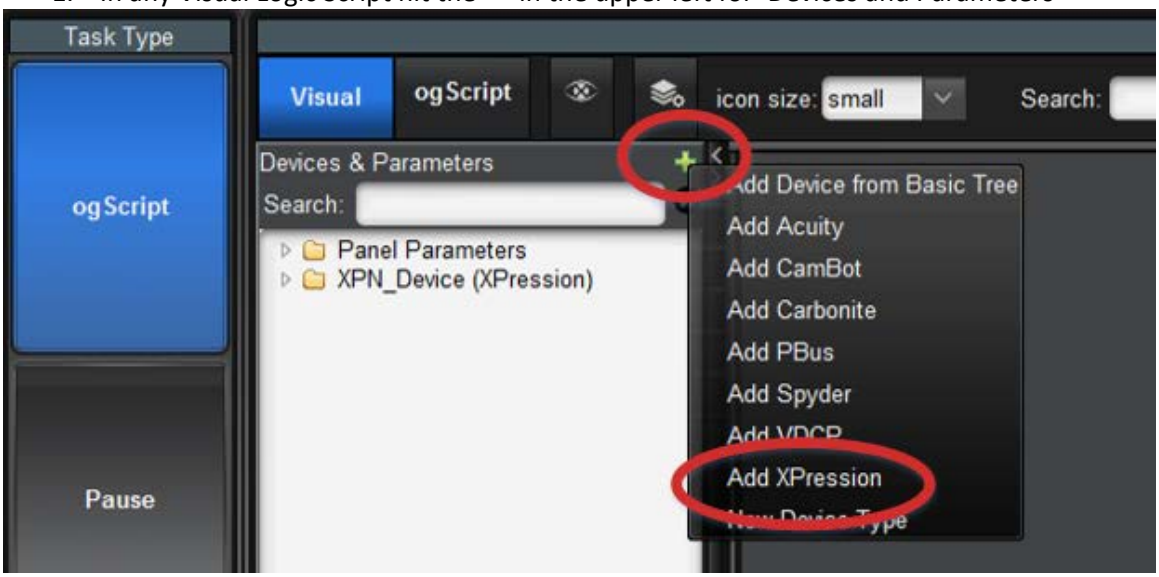
1: Sending commands through Ross Talk Commands
2: Sending Data to XPression using Datalinq

Both methods work independently from each other (and can be used at the same time as well!)

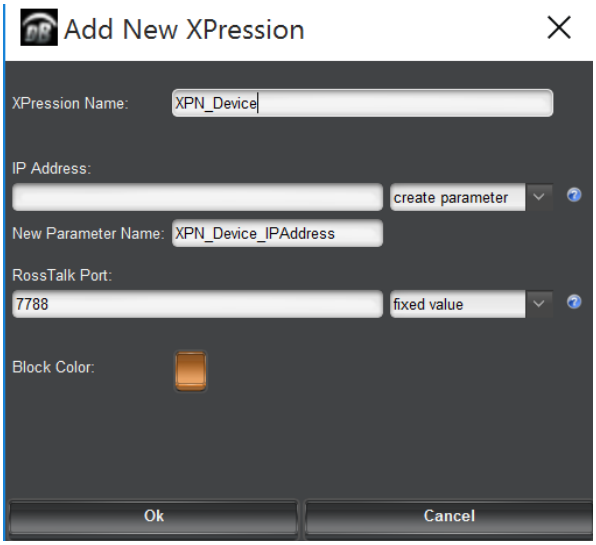
To send a RossTalk command

To send commands through a RossTalk command, the easiest way is using Visual Logic.
In a custom panel, the first thing you need to do is let that custom panel know that you want to connect to an XPression engine. (if you don't have any visual scripts, quickly add a button to your panel and then add a task to it).

1. In any Visual Logic Script hit the '+' in the upper left for 'Devices and Parameters'



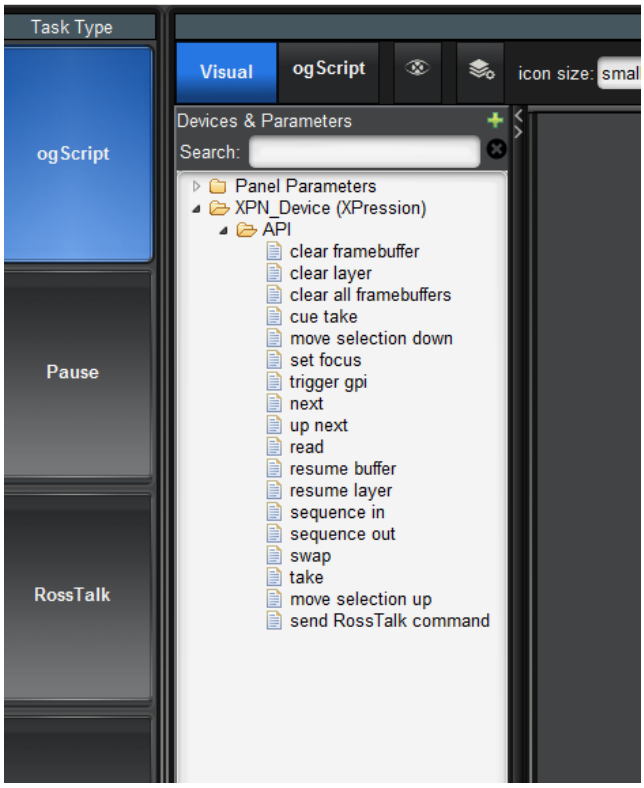
You will now get a popup asking you to name this XPression as well give it an IP address:



We suggest you select the ‘create parameter’ option so that you can easily reuse this panel with XPressions at different IP addresses, you can however hard code it to a fixed value if you prefer.

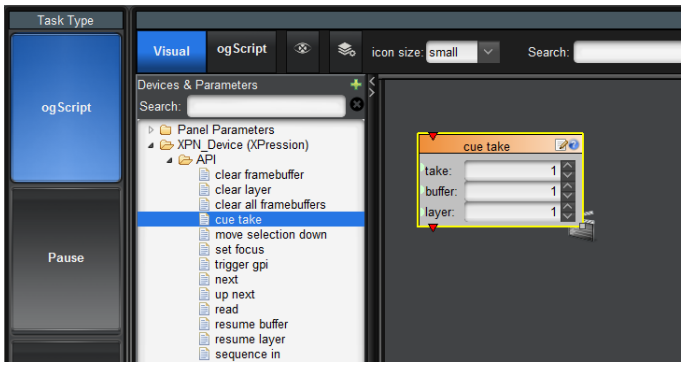
If you do use create parameter you will need to put that parameter onto your custom panel and assign it the correct IP address before your commands will work.

You will now have in your scripting tree the XPression Device, expanding the device by clicking the node in the tree allows you to get to the APIs.

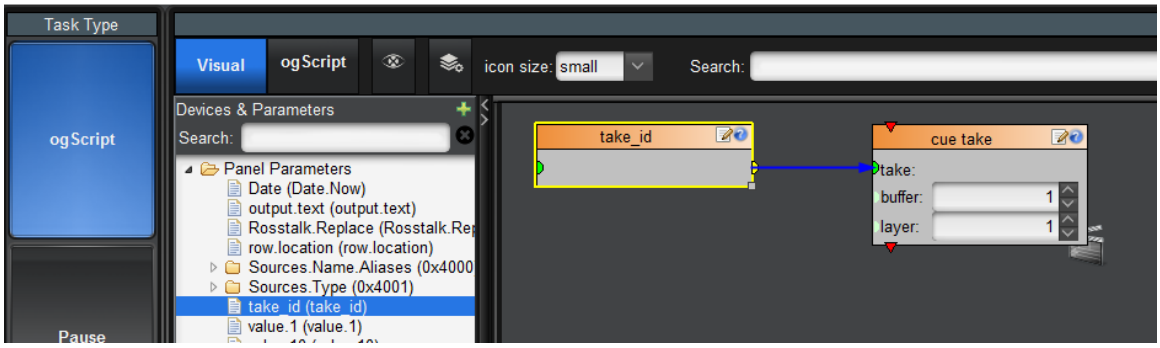


These commands can be simply dragged onto your Visual Logic script.

Add Task

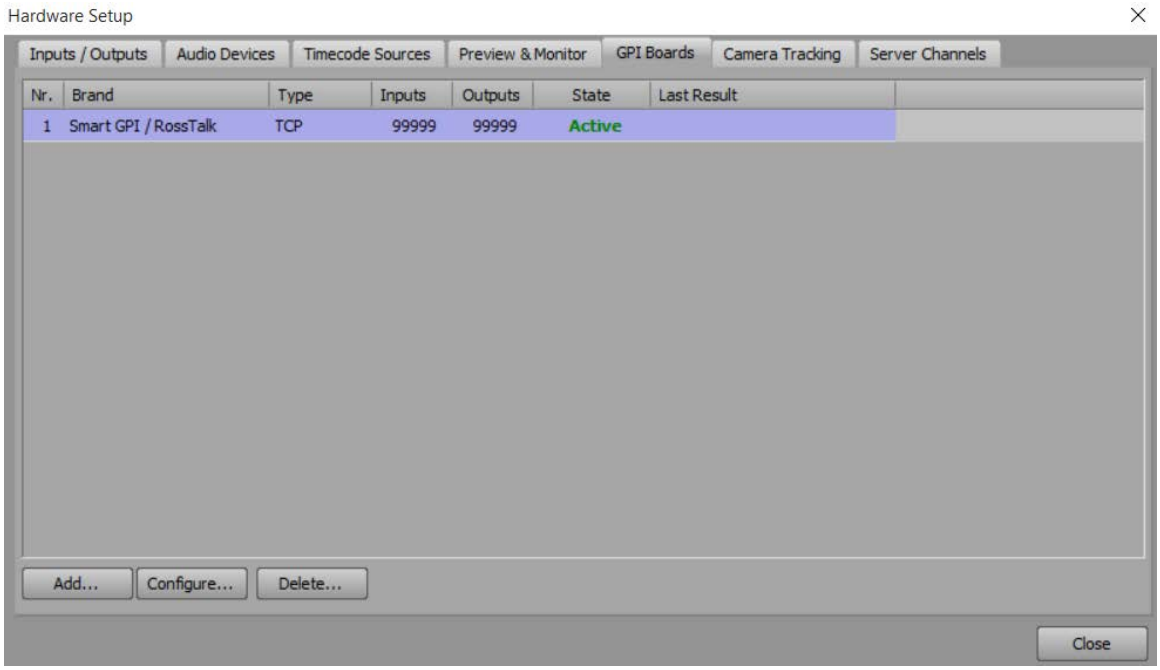


Parameters in the block can now be hardcoded in the block, or, custom panel parameters can be hooked into them:



Note that XPression needs to be configured to receive these messages.
In XPression under Edit\Hardware Setup go to the 'GPI Boards' tab.
You will need to have a Smart GPI/RossTalk element added and Active.

Example:

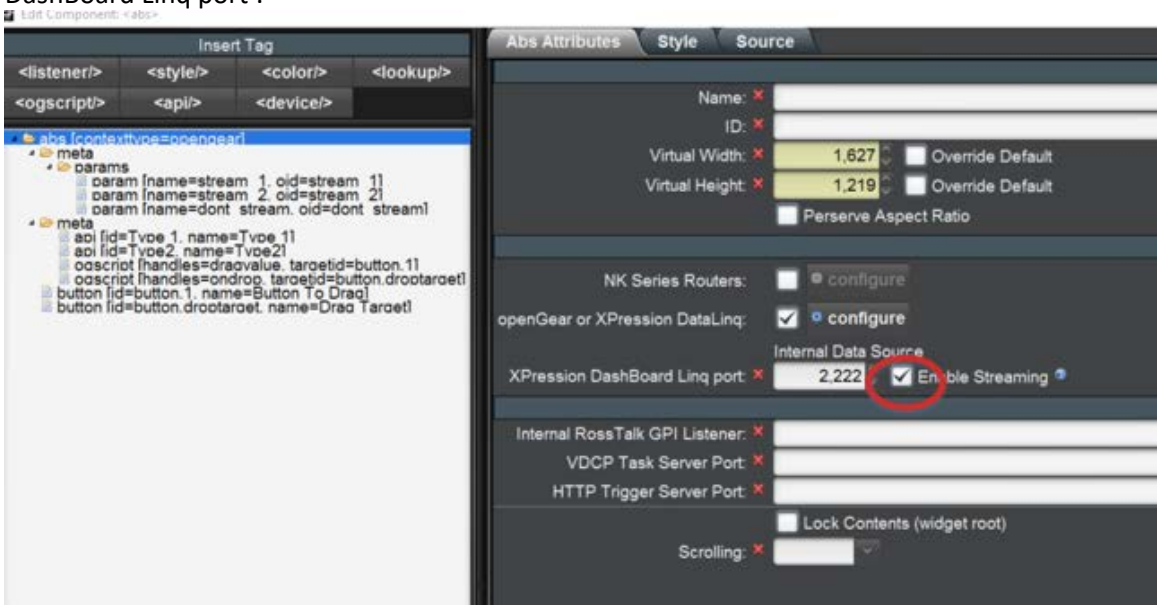


To Send Data to XPression Using Dataling

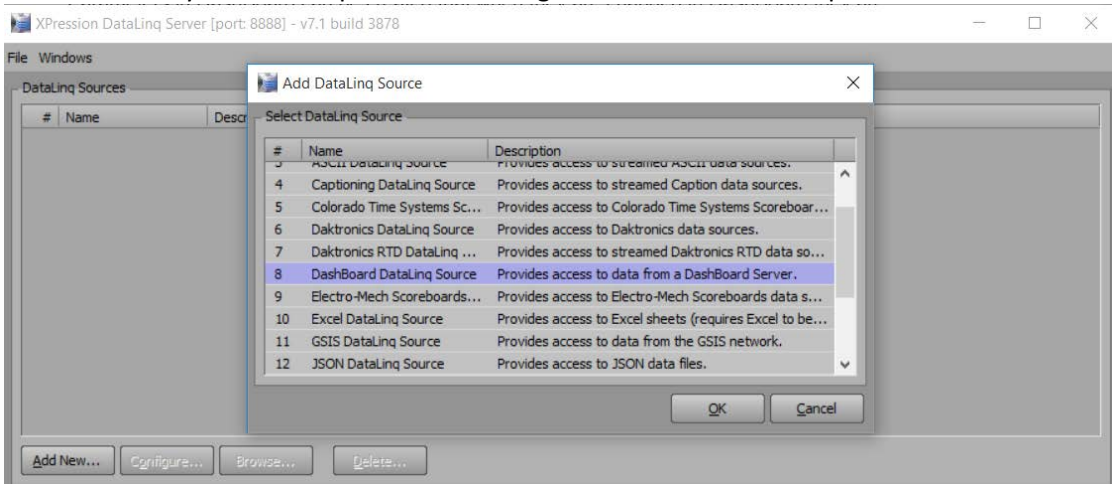
You can use a Dataling integration between DashBoard and XPression. Parameters in DashBoard can be created so that when they are changed in DashBoard they are instantly rendered live in XPression.

1. Create parameters in DashBoard.
2. Set your panel in DashBoard to stream.

In the top level node in Edit mode Click the ‘Enable Streaming’ option for ‘XPression DashBoard Linq port’.

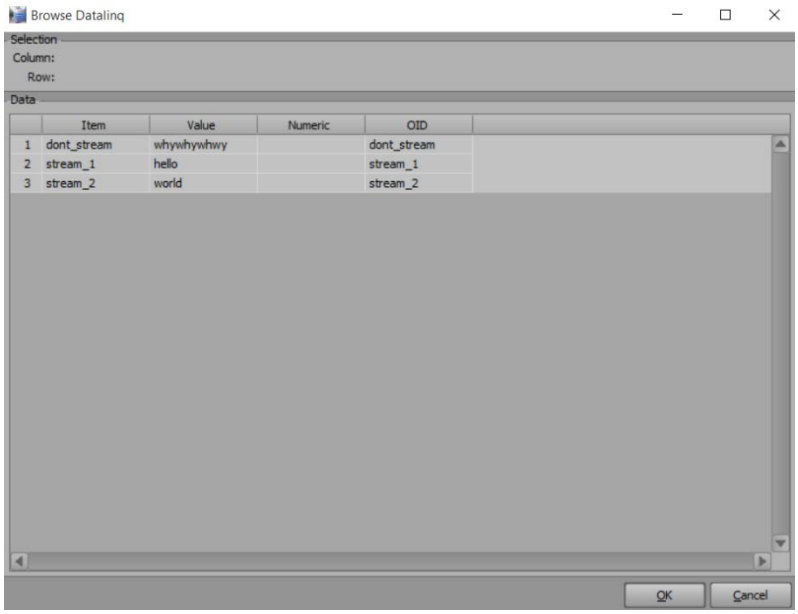


3. Make sure your Datalinq server is running then add a DashBoard panel to it.



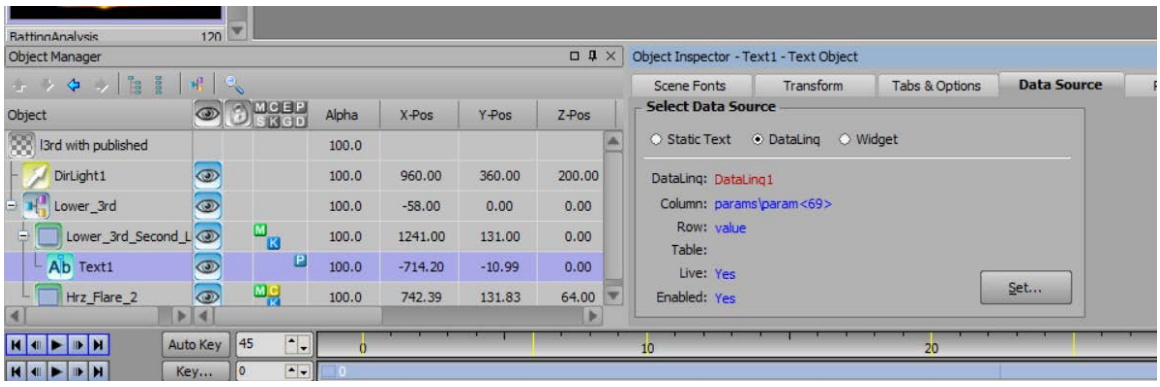
Once it is configured, to test that it is running select the ‘Browse’ button on the Datalinq server, you should see all of your parameters and their current values.

Example:



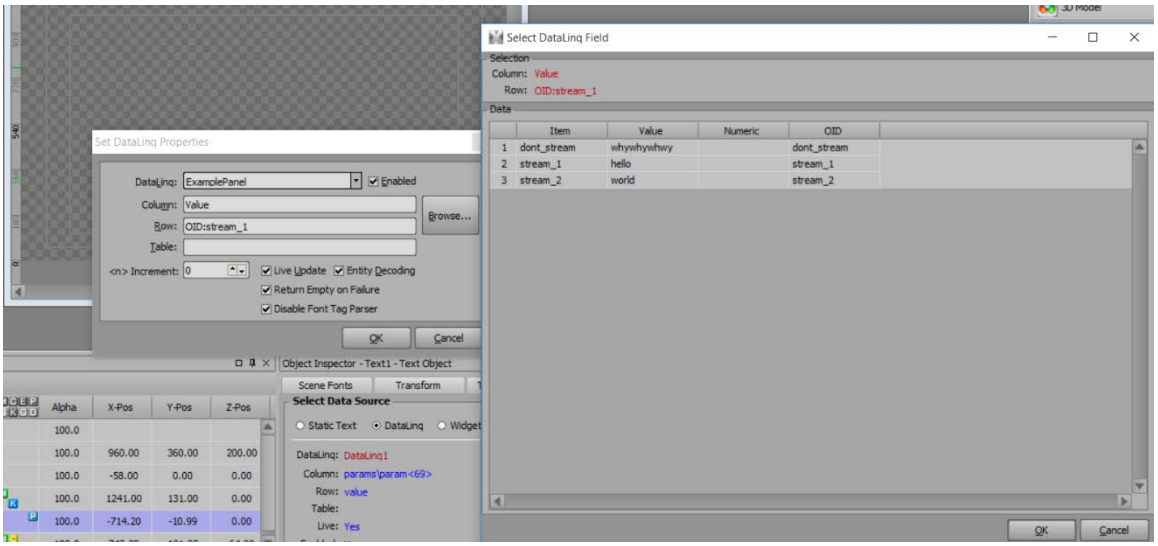
4. In your XPression project you need to attach these items to your Scenes.
For the item you want to connect, in XPression select the item in the Object Manager, and, under the 'Data Source Tab' select 'Datalinq'.

Example:



5. Select the Datalinq field.
Click the **Set** Button, select the DashBoard Datalinq source and then the field.

Example: (with the Browse button selected as well showing the XPN Datalinq browser)



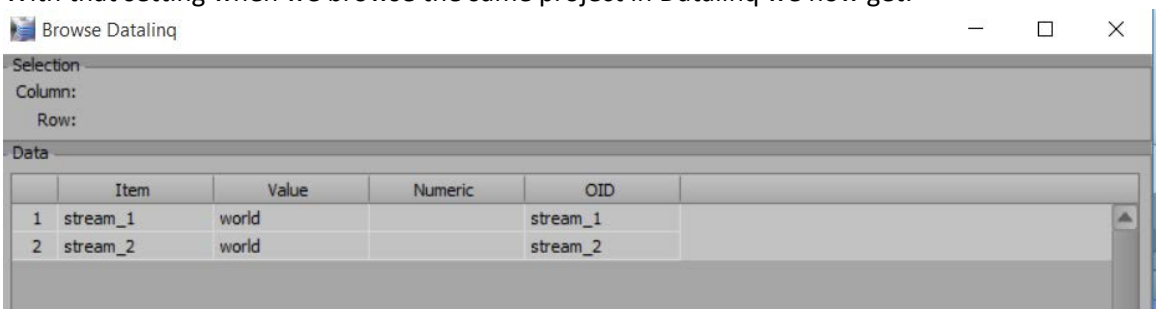
Any changes made to the Parameter in DashBoard will now be pushed and rendered live in Xpression.

Note that you can use this with numbers, strings, and elements such as Drop down lists etc..

Advanced side note:
You may not want to stream all parameters from DashBoard to XPression, if nothing else, it clutters up the choices inside of the Datalinq server as well as the selector in XPression.

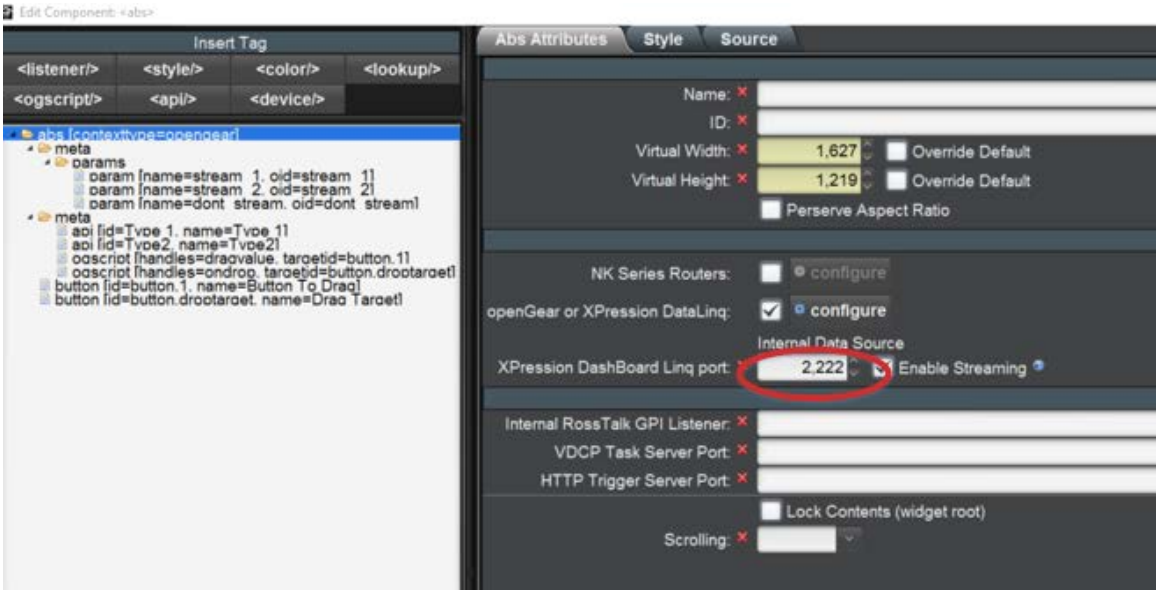
To make a parameter **not** be sent to XPression you need to add by hand the text stream=”false”
into the parameter declaration in your panel, here is an example:
<param access="1" maxlength="0" name="dont_stream" oid="dont_stream" stream="false" type="STRING" value="whywhywhwy" widget="default"/>

With that setting when we browse the same project in Datalinq we now get:



***Advanced side note*:** If you want multiple CustomPanel to talk to your Datalinq server you can do that simply by having each CustomPanel use a different port to talk on.

When you configure your DashBoard Custom Panel, simply override the default Datalinq port under ‘XPression Datalinq port’ on the same line that you enable Streaming to Datlinq.



Integrating with Carbonite switchers

The Ross Carbonite switcher integrates easily with DashBoard.

1: Sending commands through Ross Talk Commands using Visual Logic
2: Creating Custom Panels by dragging UI elements onto a custom panel

Integrating with openGear cards

The Ross XPression Character Generatour has two main methods of integration with DashBoard.

1: Sending commands through Ross Talk Commands
2: Sending Data to XPression using Datalinq

Integrating with generic networked devices

The Ross XPression Character Generator has two main methods of integration with DashBoard.

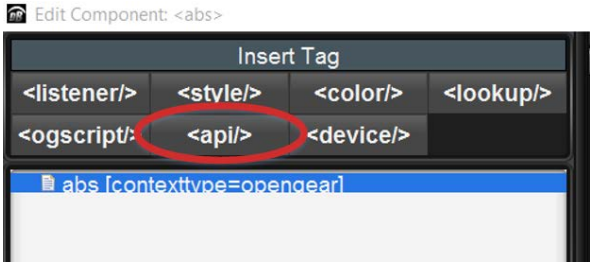
1: Sending commands through Ross Talk Commands
2: Sending Data to XPression using Datalinq

Global APIs, Tasks, and Functions in a CustomPanel

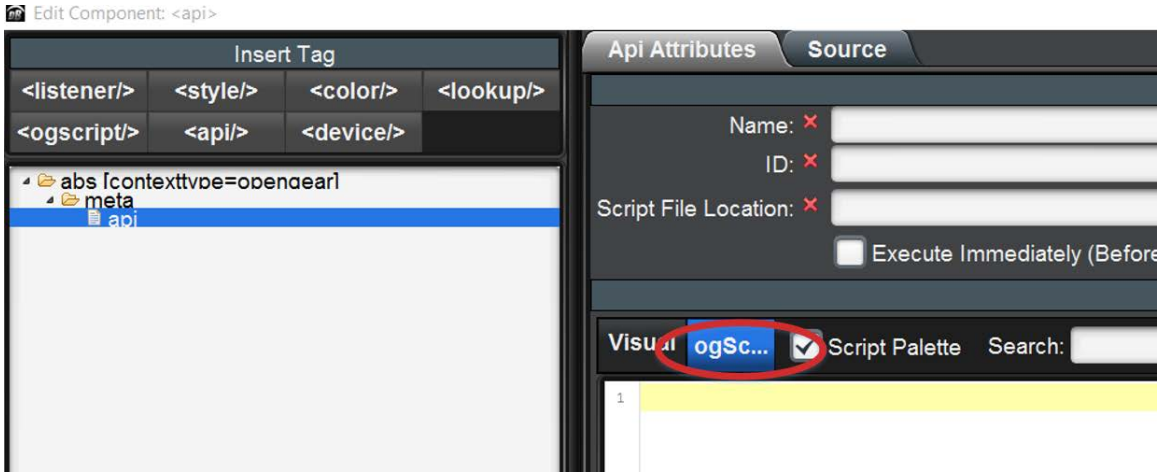
Creating your own functions in a CustomPanel is a common task that you will want to be able to do.

To create a function in a CustomPanel

1. In your custom panel with the top level node selected in Edit mode press the '<api/>' button.



You will now have a 'meta' and API tag, you will want to select the **ogScript** button in the right hand pane:



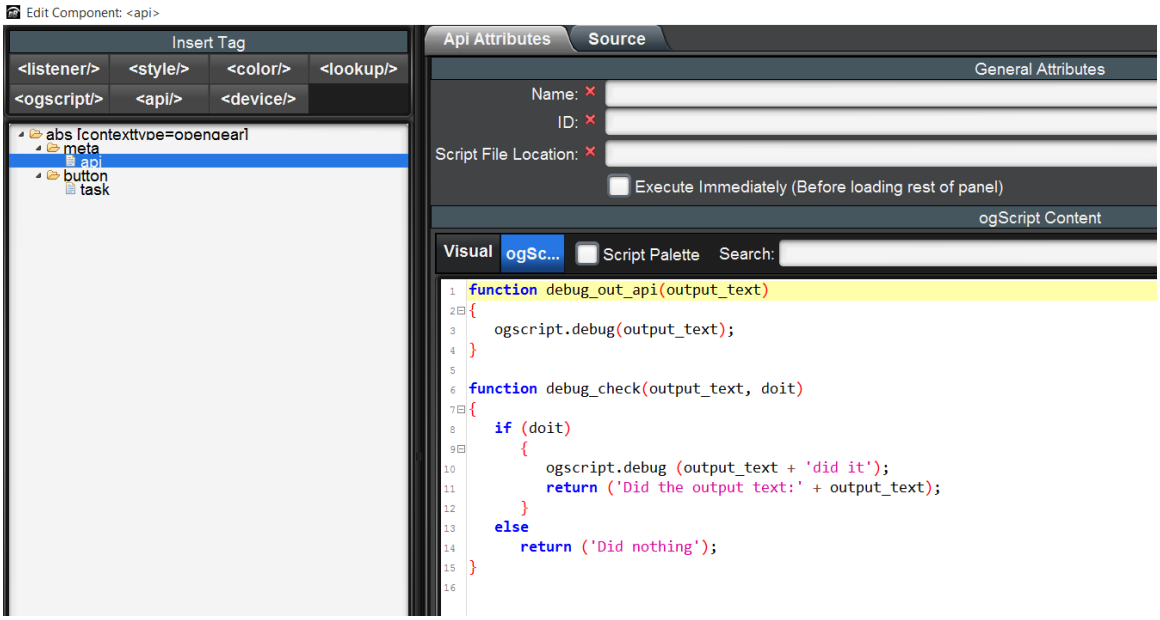
2. You are now ready to add your functions.

This is best done currently by typing scripts by hand.

Every function should follow the JavaScript conventions example:

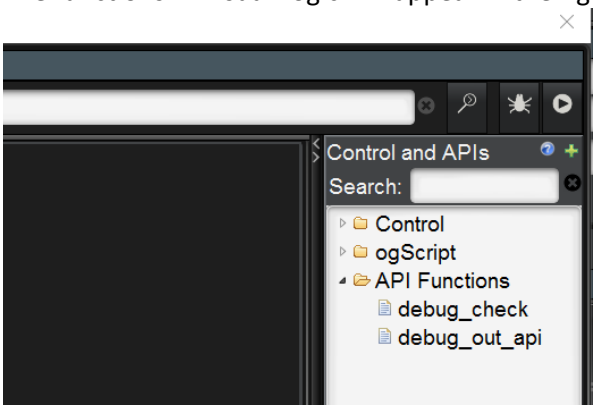
Function *name* (*parameter, parameter)* // the number of parameters is up to you
{
// content
}

Here is an example showing two functions, one with a return value:

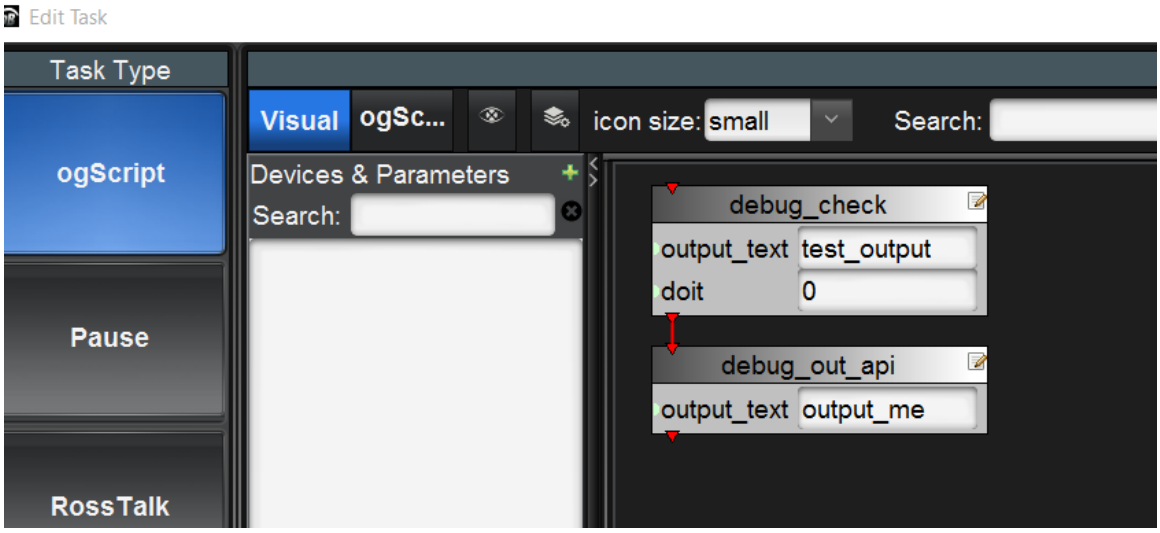


These functions can be used either in script or in Visual logic.

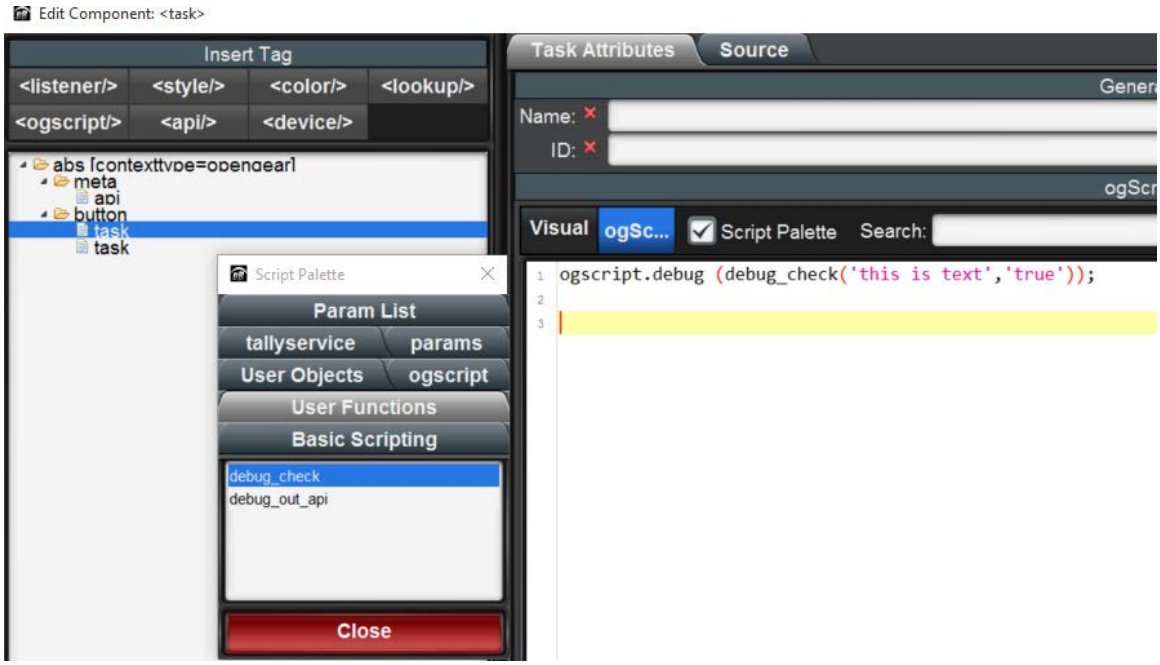
The functions in Visual Logic will appear in the right hand pane under the 'API Functions' node:



3. You can now drag these on and use them in Visual Logic or type them in by hand in another script:



Or in Script:



(note that your functions are also visible in the Script Palette).

Advanced side note: You can have multiple API blocks withing your Meta tag for organization if you want.

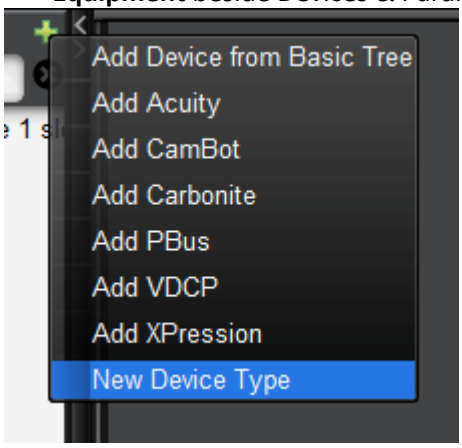
Creating Simple Devices

Some devices have the ability to be controlled using simple protocols that do not require elaborate handshakes.

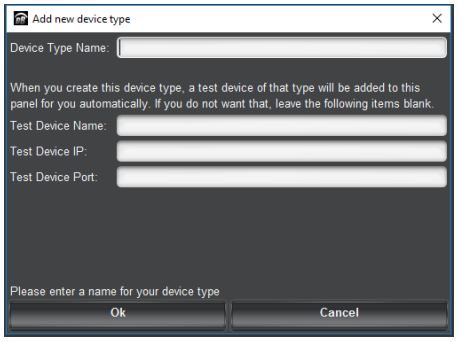
For these protocols device definitions can be made that have them appear as visual logic blocks.

To create a new device type

1. In the **OgScript Content Editor**, click the green plus icon **+** and select **Add New Equipment** beside Devices & Parameters in the top left.



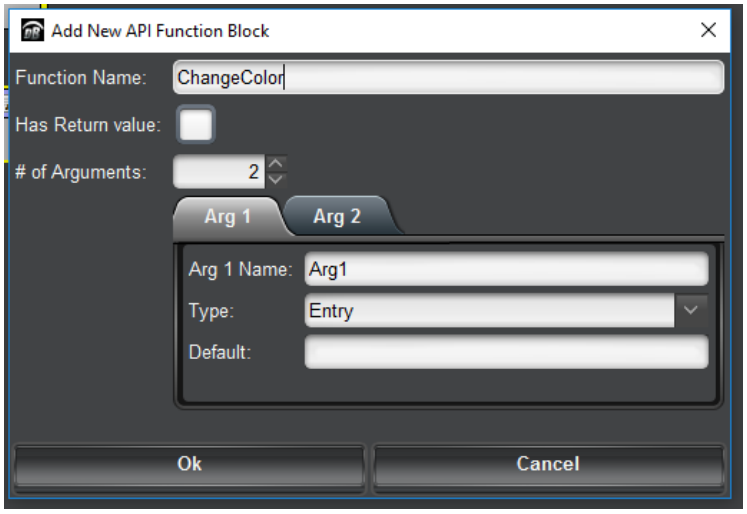
2. This opens a pop-up window where you enter the **Device Type Name** and the test device name along with the **IP Address** and **Port**.



This adds a new device to the **Devices & Parameters** list on the left side of the interface. By default the new device has two functions: **Get IP Address** and **Get Port**.

3. To add new API commands to the device right click on the API folder and choose **Add API Function Block**.

The **Add API Function Block** opens a pop up where the new function is given a name, whether it has a return value, and how many arguments are used for the command. Each argument can be named and there are four styles of argument type (**Entry**, **Spinner**, **CheckBox** and **Dropdown**). A default value can be provided.



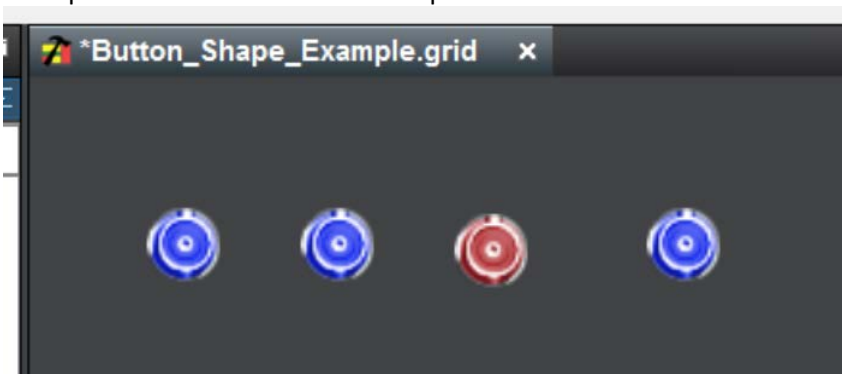
Once the arguments are added you are then in a Visual Logic view where you can define what the API specifically does. The new API is available in the accordion box for the device in the left hand menu.

The API file is stored (by default) in the DashBoard directory under VisualLanguage/blocks/Devices so that file can be shared with others as desired.

Modifying Button Shapes

Buttons can have their shape modified to be any image and shape instead of the simple rectangle.

Example of some buttons in the shape of a BNC:



Here is what the **Style** settings for these buttons look like:

Note: There is an override for the background colour in the **Toggle On** and **Toggle Off** settings.

Overlaying UI components to ‘drop’ data

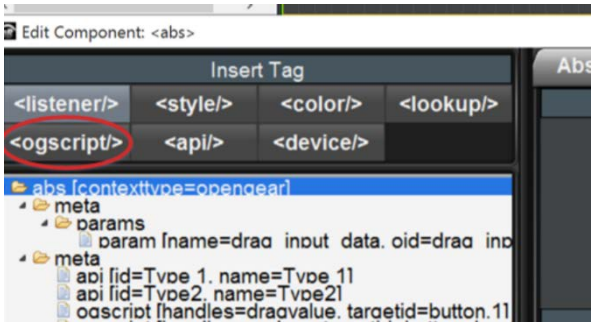
You can drag a UI element over another element, and when you let go of the mouse it sends data to the UI element that is beneath.

To do this, the UI elements must first be set to be draggable. This means that when they are moved over another UI element they can ‘drop’ information which can then be used by the UI element that they were dragged onto.

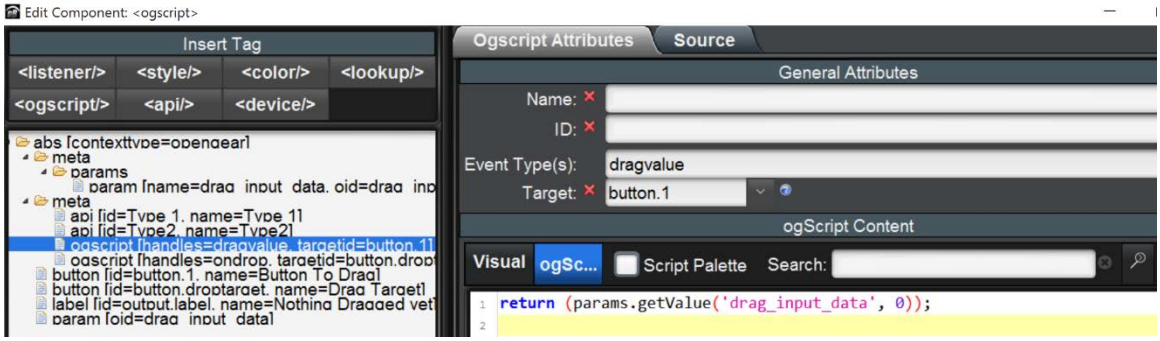
UI elements can also be set as drop targets.

To set a UI element as draggable

1) In your Custom panel you need to create an <ogscript/> function and set some config data for this ogscript function.
   a) In your custom panel in edit mode with the top node selected click the <ogscript/> button



   b) In your new ogscript function you need to set the event type to ‘dragvalue’



   c) In the ‘Target’ section you need to pick the ID of your UI element.

Note: this is not the name of the element but the ‘ID’, if you haven’t set the ID of that UI element you need to do that before it will appear in the ‘Target’ drop down list.

d) Type the script for the information that this UI element will drop.
The value that you put in the “return” section will be what is dropped.

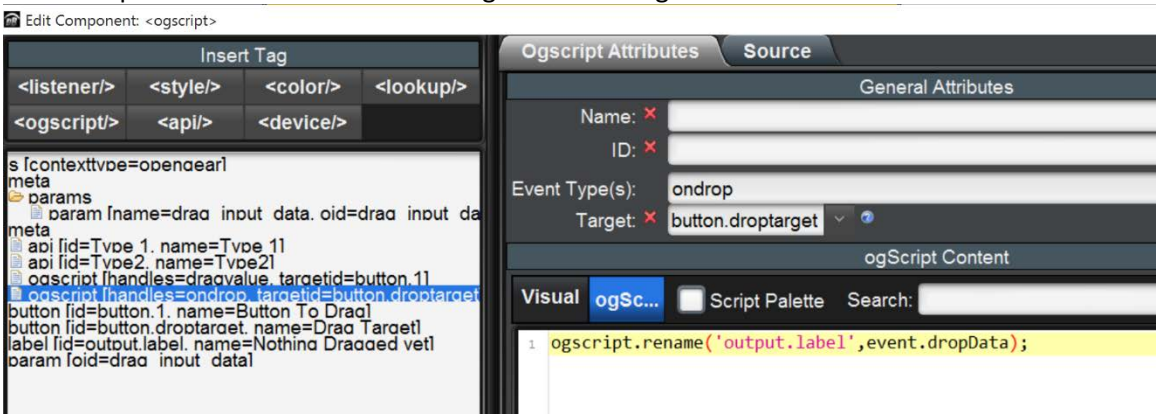
The example above gets the information from a parameter on the custom panel.

2) You need to set the UI element to be able to receive a drop target. This is very similar to the previous step, in this case though the event type is ‘ondrop’.

The function that you write here will be able to receive the drop data and then do something with it.

Important note: the drop data will appear in a variable called **event.dropData**.

The example below shows the data being used to change the name of an on screen label.

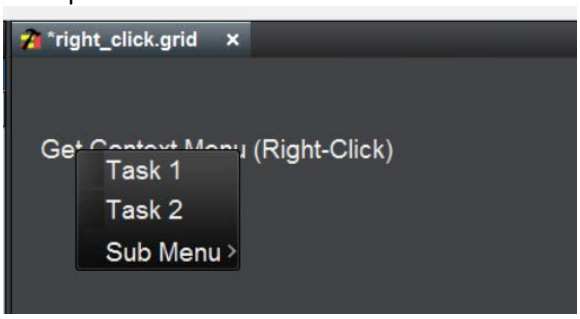


Advanced note: You can override the look of the drop icon as you drag it in the Style section of the UI element that is to be dropped.

Creating Right-click Context Menus

UI elements in your custom panels can have different popup menus appear when you right click on them. These right click menus can have tasks attached to them.

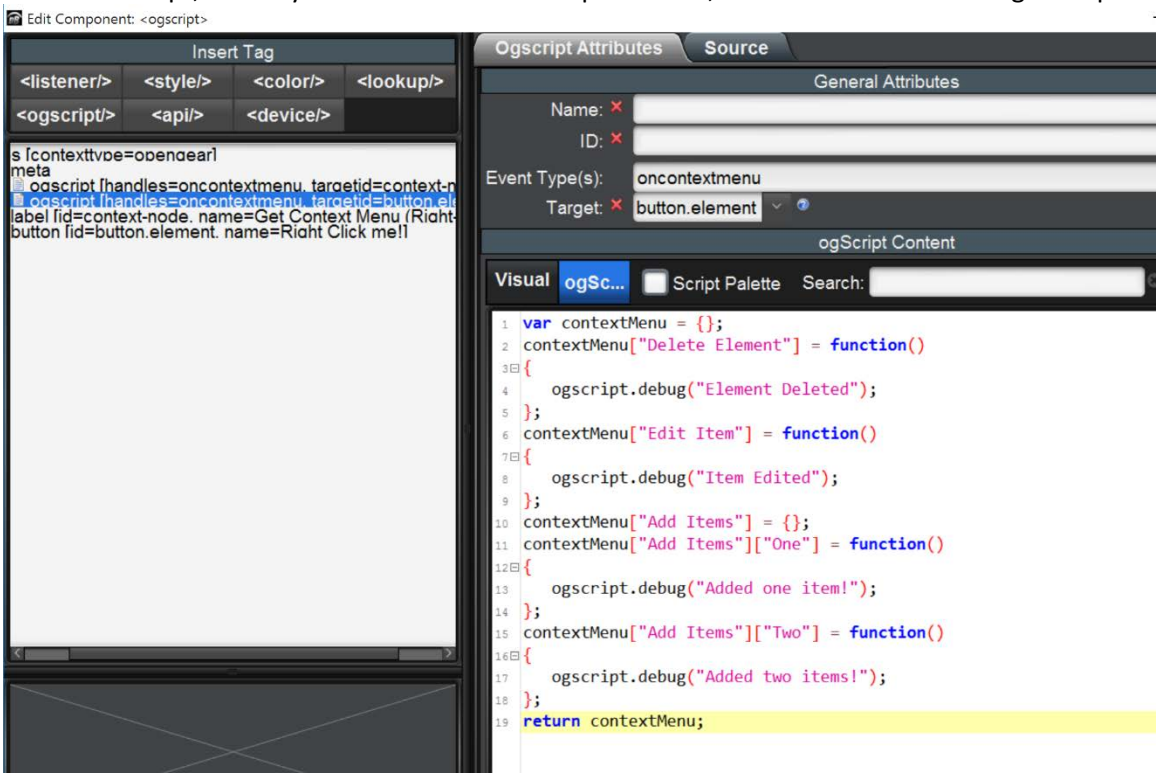
Example UI of this on a label:



Doing this involves attaching an 'oncontextmenu' item to the ID of the UI element.

1. In your custom panel, at the top node, select <ogscript/>

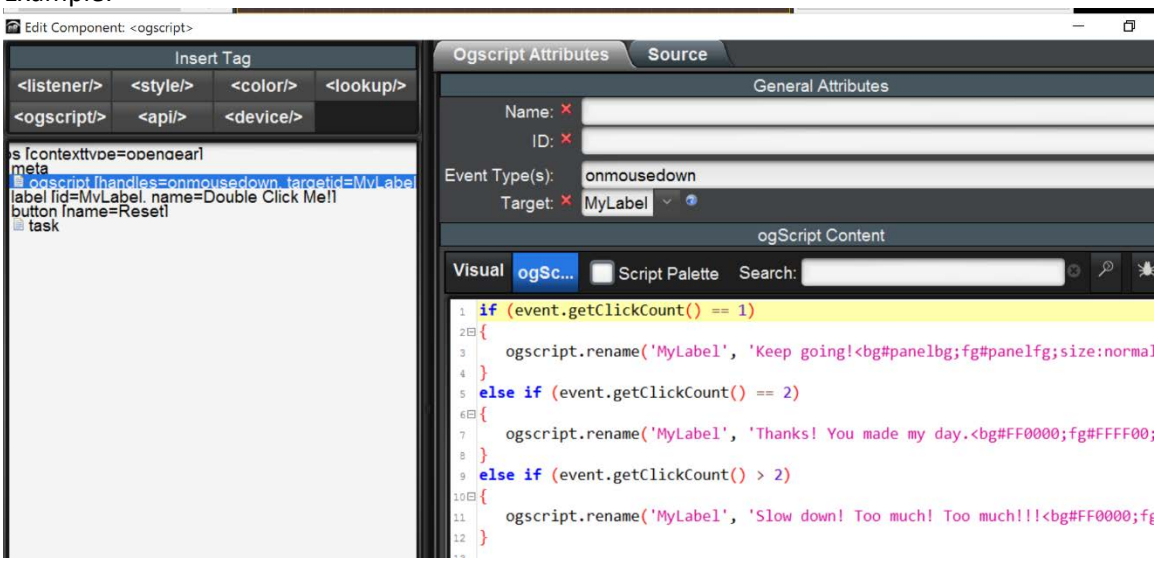
2. In the script, define your menus and add script to them, as shown in the following example:



Setting Special Mouse Actions for UI Elements

If you want to handle double clicking a UI element as a special action you need to register a function for that UI element for the ‘onmousedown’ event.

Example:



The example above shows registering a task against the UI ID ‘MyLabel’.

Notice the use of the “event.getClickCount()” to determine if it was a single click, double click, or more. The regular action of clicking on that element will also be run.

Remember, to register a function, have the top level node selected in your panel in the tree on the left and press the <ogscript/> button.

Only UI Elements that you have given an ‘ID’ to will appear in the ‘Target’ drop down list.

Creating Parameter Structures to Group Data

DashBoard allows you to create parameters that are structures.
Structures allow you to group your data together instead of having to make several parameters and remember that they are linked.

From the online help:

param (struct)

Compound parameters may be defined through the use of the STRUCT param type. A **struct** contains a collection of parameters. **Structs** may not be nested. **Struct** must have a **constrainttype** of STRUCT. Members of the **struct** are declared through `subparam` tags within the `value` tag.

A struct may also use another param as a template to pre-populate the member sub-param declarations. This is done through the `templateoid` attribute.

Syntax

```
<param constrainttype="STRUCT" oid="oid"
type="STRUCT" attributes>
   <value>
    <subparam suboid="sub-oid" sub-param-
attributes/>
    <subparam suboid="sub-oid" sub-param-
attributes/>
    . . .
   </value>
</param>
```

Attributes

Attribute	Values	Restrictions	Description
oid	*String*	Required	The OID of the parameter (can be used to override an existing parameter).
access	0		Parameter is read-only in DashBoard
	1		Parameter is read-write in DashBoard
name	*String*		Parameter Name
widget	Positive integer	Must be a valid widget hint	Defines the default widget hint for the param.
type	STRUCT		Must be set to STRUCT.

Attribute	Values	Restrictions	Description
structtype	*String*		Defines the structure type. Specifies a dependency of a widget upon a global struct parameter with matching **structtype**. Currently this type checking is restricted only to PanelBuilder UI; a custom widget will only be available in PanelBuilder if a parameter exists with matching **structtype**.
templateoid	*String*		Specifies a template struct parameter to pre-populate the subparams.
constrainttype	STRUCT		Must be set to STRUCT
value			Container for **subparam** elements.
subparam	*param*	May not be a nested struct param	Member parameters, declared using the same syntax as a param declaration, with the exception that its oid is specified in the attribute **suboid**.

Default values shown in **bold**.

Example

The following declares a struct parameter.

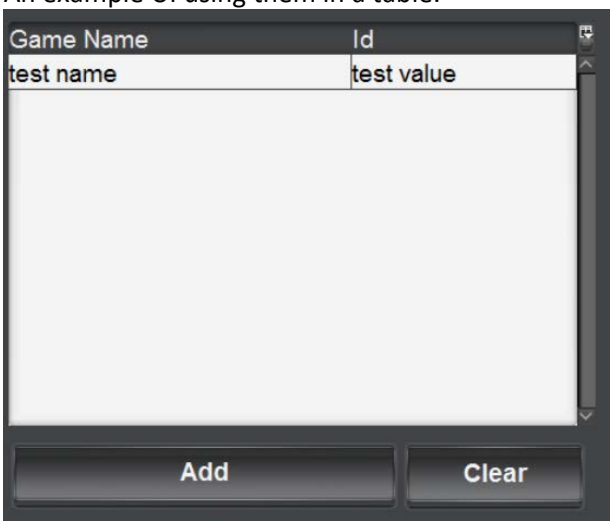
```
<param access="1" constrainttype="STRUCT" name="Clip
Info" oid="clipInfo" type="STRUCT" widget="36">
   <value>
    <subparam name="Clip Name" suboid="ClipName"
 type="STRING" value="Test"/>
    <subparam name="Director" suboid="Director"
 type="STRING" value="Test"/>
    <subparam name="Air Date" suboid="AirDate"
 type="STRING" value="Test"/>
    <subparam name="Author" suboid="Author"
 type="STRING" value="Test"/>
   </value>
</param>
```

The following declares an array of struct params, using the previous example as its template. Note that any attributes specified explicitly will override the values provided in the template.

```
<param access="1" constrainttype="STRUCT" name="Clip
List" oid="clipList" templateoid="clipInfo"
type="STRUCT_ARRAY" widget="36">
<value>
   <subparam suboid="ClipName" value="Winter is
Coming"/>
```

```
   <subparam suboid="Director" value="Tim Van
Patten"/>
   <subparam suboid="AirDate" value="April 24,
2011"/>
   <subparam suboid="Author" value="David Benoiff
&amp;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;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;amp; D.B. Weiss"/>
</value>
</param
```

An example UI using them in a table:



Publishing a Simple Web Page

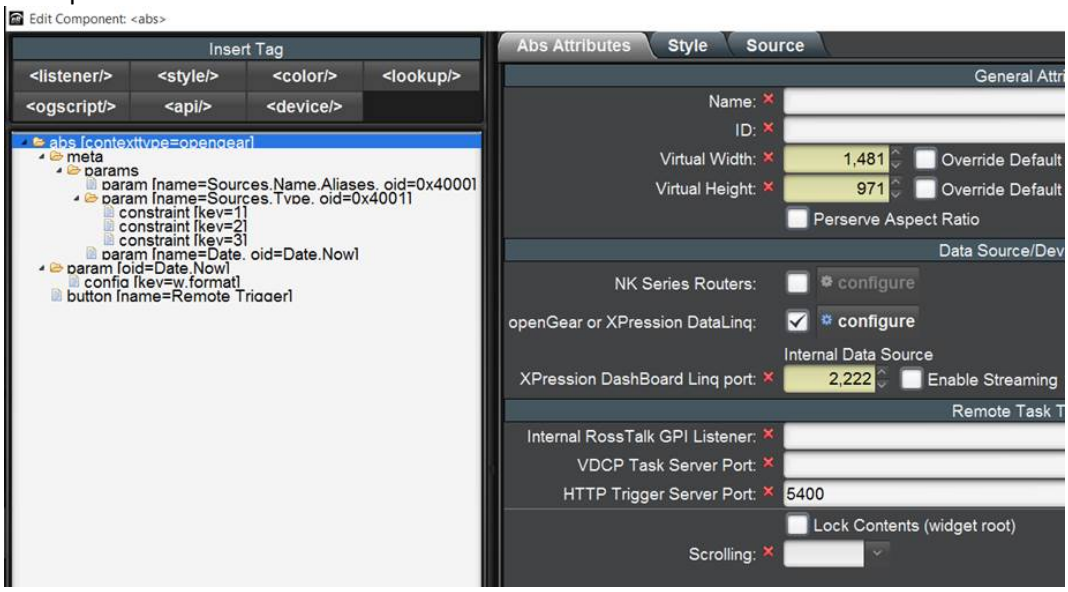
DashBoard allows you to publish from it a simple web page that is nothing more than a series of buttons. This is for those users who want to be able to trigger CustomPanel events from items such as an iPad which do not support Java and thus DashBoard.

If you want to do that:

1. Set the panel to publish the web page

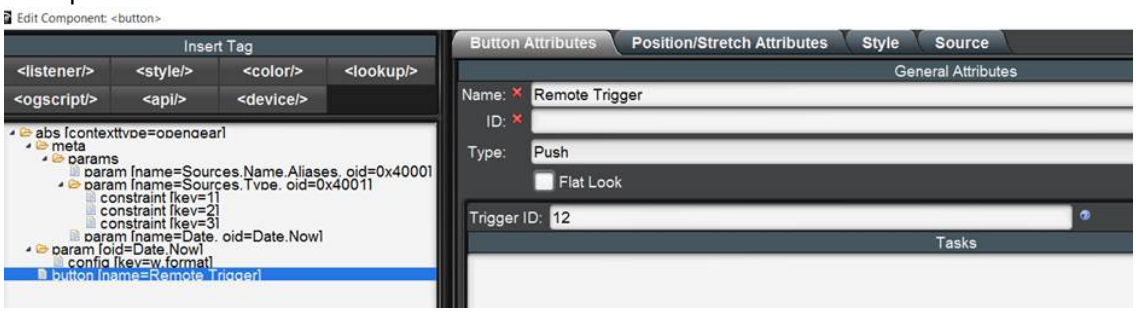
   In the top root node of the panel set the 'HTTP Trigger Server Port' value:

   Example:



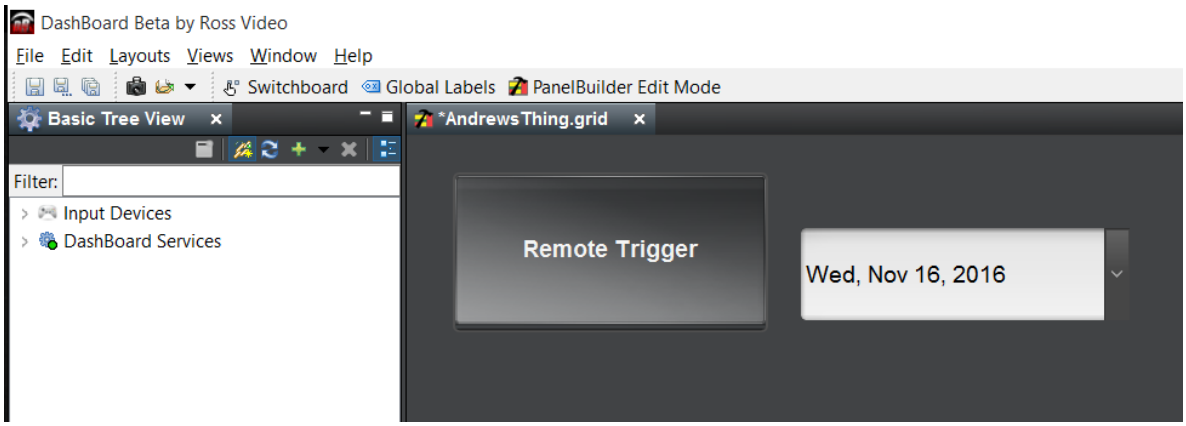
2. Every button that you want to publish needs a 'trigger ID'.

   Example:

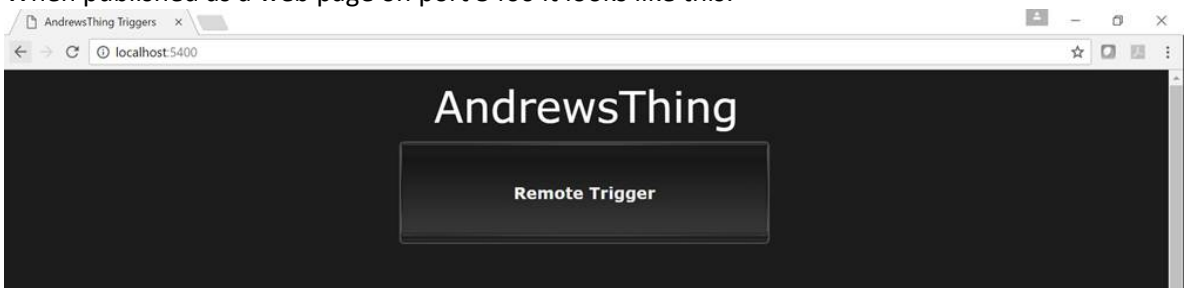


3. Look at it in your browser.

   I had a Custom Panel called 'AndrewsThing' with one button on it as follows:



When published as a web page on port 5400 it looks like this:



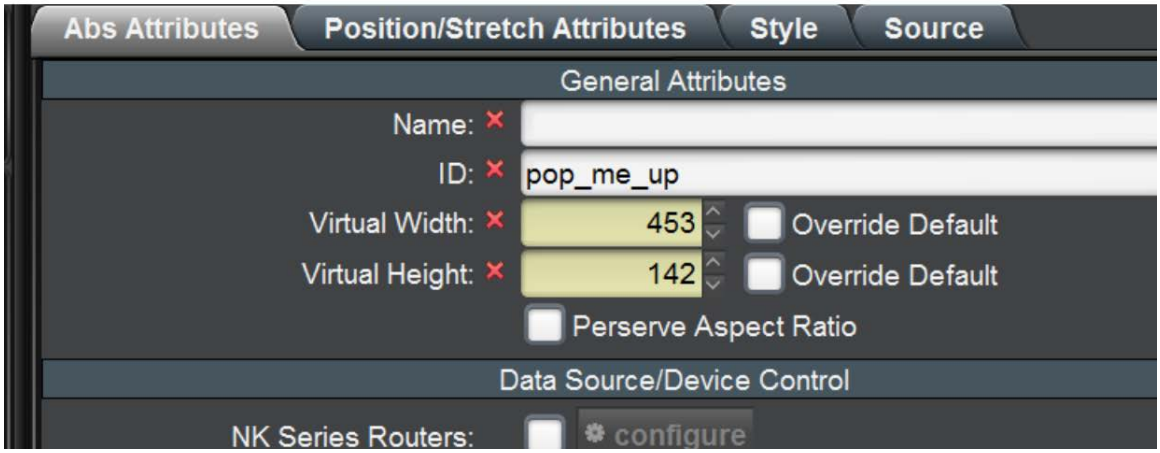
Creating Popup Windows

Often you may want a UI popup window to appear when you select a button.
For example, you might want a popup to appear when you select a button marked ‘Edit’.

This window may have lots of parameters in it, or only one.

1. Create an ABS (Blank Canvas) that will be the popup window on your custom panel.
- Make sure to give it an ID in the editor on the ‘ABS attributes’ page.

Example with an ID called ‘pop_me_up’

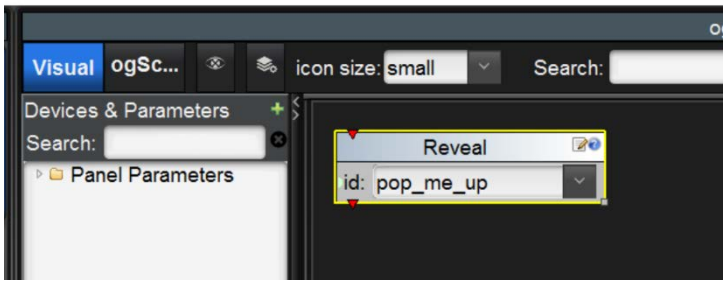


- Make this ABS **not visible** by default, this needs to be done in the source code.
- Add the text visible="false" to your declaration for the ABS.

Example for the above: (ignore the style elements etc..)
<abs height="142" id="pop_me_up" left="35" style="bdr:thick;bg#CA0505;" top="143" visible="false" width="453" />

2. To make this visible you will use a script.

A simple example would be adding a script to a button that when pressed shows this window.



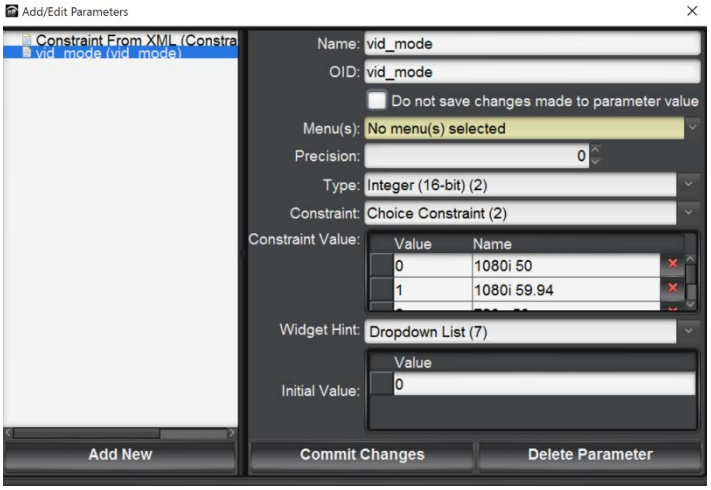
You will need to use a similar script with ‘Hide’ instead of Reveal to have this window no

longer be visible.

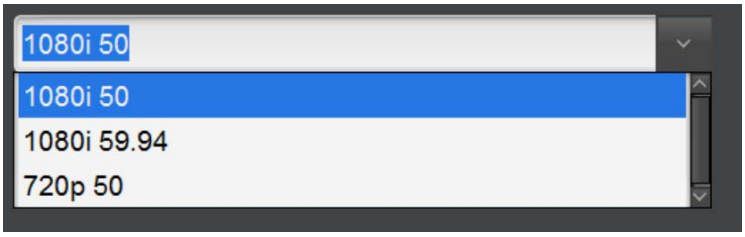
Updating Constraints

A constraint on a variable limits the input a user can put in.
Example: Min / Max value on an integer.

More common is an integer being used for a dropdown list.
Every element in the dropdown list has a numeric value and text to show for it.
Example configuration:



What it looks like in the UI:



(constraints are also used to create sets of toggle buttons etc. Same configuration, different widget).

You may want to update your constraint using script while your panel is running.
A common example is reading data from an XML file and then updating a constraing based on the XML information such as making a drop down for team names.

To update a constraint: (done in scripting by hand)
Step 1: create a temp array. Example:

```
var constraintData = [];   //Placeholder for the new constraint data
```

Step 2: Add the new constraint item to the array

constraintData.push({"key": 1, "value": “droplist item”); //Push the key/value pair
// the Key should be different for each item you push.
// Do this changing the number (1 above) and “value” (droplist item) above for each entry in your new constraint

Step 3: Attach this new constraint to your data
var constraint = params.createIntChoiceConstraint(constraintData);   //Create the constraint
params.replaceConstraint("OID Name", constraint);   //Replace the constraint</task>
// the text “OID Name” would be the ID of the parameter you created earlier.

Parsing XML Files

Data on your computer in an XML file is a common source of information to a custom panel.
Side note: Excel can be configured to save its information into an XML file.

1. Open the file and create an XML document object from it.
2. Create a search to get the information you want out of the file (as a node list).

You can use any of the javascript XMLDocument Object methods such as:
-getElementsByTagName()

Or run XPath queries using
-ogscript.runXPath();

3. Iterate through node list doing what you need with it. (often updating a constraint)

The following example shows parsing a very simple XML file and then updating the constraint for a dropdown list based on the result:

```
var xmlDocument = ogscript.parseXML('data.xml');        //Parse the XML
var nodeList = xmlDocument.getElementsByTagName("team"); //Get the tags we want to
include in the constraint

var constraintData = [];   //Placeholder for the new constraint data

if (constraintData.length == 0)  //Placeholder if we don't have anything (OPTIONAL)
{
  constraintData.push({"key": 0, "value": "Nothing"});
}

for (var i = 0; i <         nodeList.getLength(); i++)         //Loop through each tag we found
{
  var node = nodeList.item(i);                        //Get the node
  constraintData.push({"key": parseInt(node.getAttribute("ID")), "value":
node.getAttribute("name")});  //Push the key/value pair
}


var constraint = params.createIntChoiceConstraint(constraintData);   //Create the constraint
params.replaceConstraint("Constraint_From_XML", constraint);        //Replace the
constraint</task>
```

Using putObject and getObject

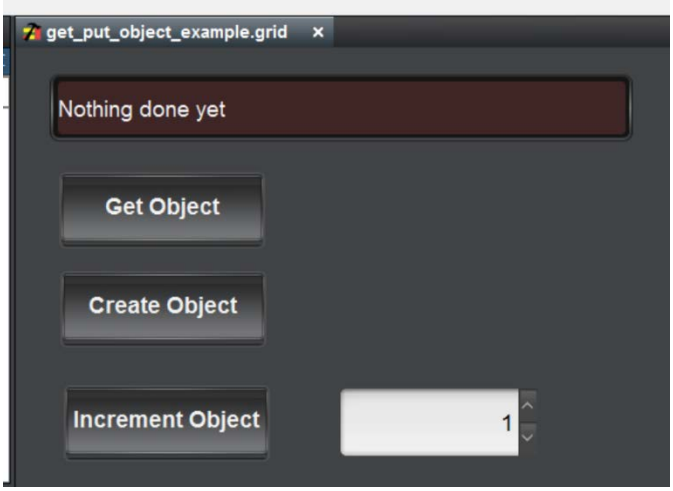
Javascript objects can be created on your panel and then stored for later retrieval and use. Think of it as a dynamically created global object.

Example use from the help file is to parse an XML document and then store it so that you don't have to keep re-parsing it.

The ogscript.putObject('Key','object') creates the global object and stores it with the name that you provided in 'Key'.

When you want to get the object, you simply pass in the name to ogscript.getObject('Key');

Here are some quick code examples using the following Custom Panel



Get Object code:

```
var test= ogscript.getObject ('globalNumber');

if (test == null)
  params.setValue('output.text', 0, 'Object does not exist');
else
  params.setValue('output.text', 0, 'Object found, number is: ' + test.number);
```

Create Object code:

```
var number_object = {name:"Oslo", number:20};
ogscript.putObject('globalNumber',number_object);

params.setValue('output.text', 0, 'Object Created with value of 20');
```

Increment Object Code:

```
var test= ogscript.getObject ('globalNumber');

if (test == null)
  {
  params.setValue('output.text', 0, 'Object does not exist');
  return;
  }

test.number += params.getValue('increment.value', 0);
ogscript.putObject('globalNumber',test);

params.setValue('output.text', 0, 'Object Updated to value: ' + test.number);
```

GPI Events

For DashBoard, a GPI is a very simple message that can be sent:

- Within a DashBoard Custom panel
- Between DashBoard Custom panels
- External GPI Ross Talk messages can be translated into DashBoard GPIs

A GPI Event is a DashBoard event with two values:

- Key (name of the GPI) (usually a number, but it does not have to be).
- State of the GPI

Many uses of GPIs do not use the State of the GPI. I.e. receiving the GPI message is enough to trigger an event on a device.

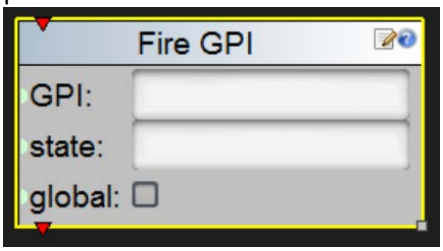
Simple use:
Send / Receive GPI #3.

More advanced: (use a name instead of a number)
Send / Receive GPI TRIGGER_LIGHTS

Even more advanced (use a name and a state 'off' here)
Send / receive GPI TRIGGER_LIGHTS OFF

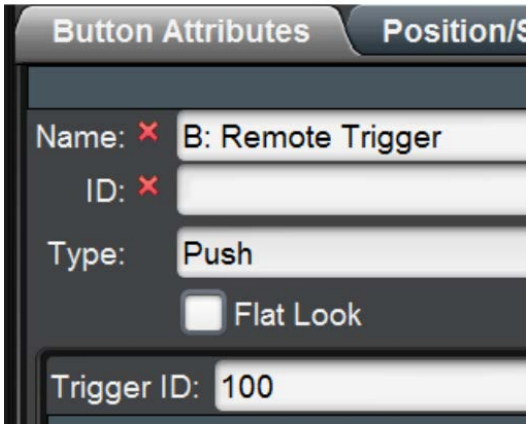
** Most uses of GPIs use the simple version **

Sending a GPI in DashBoard is most easily done using Visual Logic, as shown in the image and procedure below:

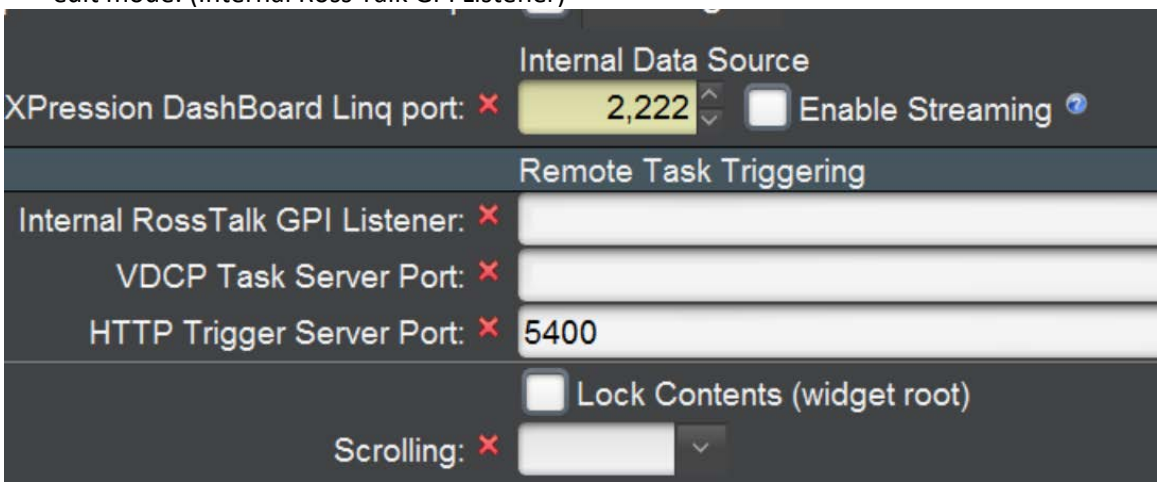


To send a GPI in DashBoard using Visual Logic (the easiest method)

1. To have a UI element run its task when a GPI is received simply set the 'Trigger ID' to be the GPI that you want it to act upon. The following example shows a button that will run its tasks when it receives GPI '100':



2. To enable DashBoard globally to receive GPI messages you need to set the preferences item: Window\Preferences\Ross Talk GPI Listener
3. Enable the checkbox. (If you change the port you will need to make sure external senders use the correct port).
4. You can also enable specific CustomPanels to listen for GPIs in the ABS attributes page in edit mode: (Internal Ross Talk GPI Listener)



Advanced Note: Keyboard Shortcuts in DashBoard can be created that send GPIs, and, do pretty much anything you can think of!

Using Message Builder and Parser

DashBoard comes with some helper objects you can use to create messages to send as well as parsing messages that are incoming.

```
CreateMessageBuilder()
CreateMessageParser()
```

Each of these creates an object with many functions to help you create and receive messages.

Creating API Tables

It is possible to create a table where every single cell runs custom code both to draw itself and to respond to changes as well as mouse clicks.

This is a very advanced feature and will not be covered today, however, you should be awareness of its usefulness.