Understanding Flags

> This documentation might be outdated. Please consult the Reactor manual at
> https://docs.skaarhoj.com/11_advanced_topics/flag-groups.html

Inside reactor we have support for setting flags true or false. These flags are based around how TSL works to some extent and provides a list of true/false states with 4 "bits" per index.

Flags in reactor consists of 4 bits per index, that can be set true or false individually, each of these flag bits can be used for different things but you will mainly see them used for providing tally and routing support in configs as they are a great way to store the intermediate state that comes in from a GPI pin or from a device like an ATEM switcher. Then use the stored state to send the Tally to the cameras

The 4 "colors/bits" main use cases in the default configs for each color are:

Red	Mainly used for Program Tally
Green	Mainly used for Preview Tally
Blue	Not widely used in default configs yet
White	Mainly used for storing what camera to Route on a mixer/videohub

These are not limited to this use case, but this is the current convention we use internally.

Define flag groups and sync them up

When you use flags you will need to add them to your project as "Flaggroups" with a name and an Index.
One Important note is that ANY config that you have loaded in the project that uses the same Group ID will be synced even if you have different names for them. This means that you can "sync" up tally or routing between 2 or more controllers running in the same reactor just by changing the flaggroup ID for "Tally" to be the same on both panels. This is also a very simple way to add a GPIO or Tally box to an existing system/setup.

This also brings the problem of if you happen to set your Routing group to the same ID as your Tally group, funny stuff might happen. But since we are using different Bit's for it, you should be fine and not risk having "crosstalk" between them. For simplicity we have made sure to make them on different group ID's in all default configs.

Here is an example from a PTZ Extreme about how these groups are defined on Index 10 and 11:
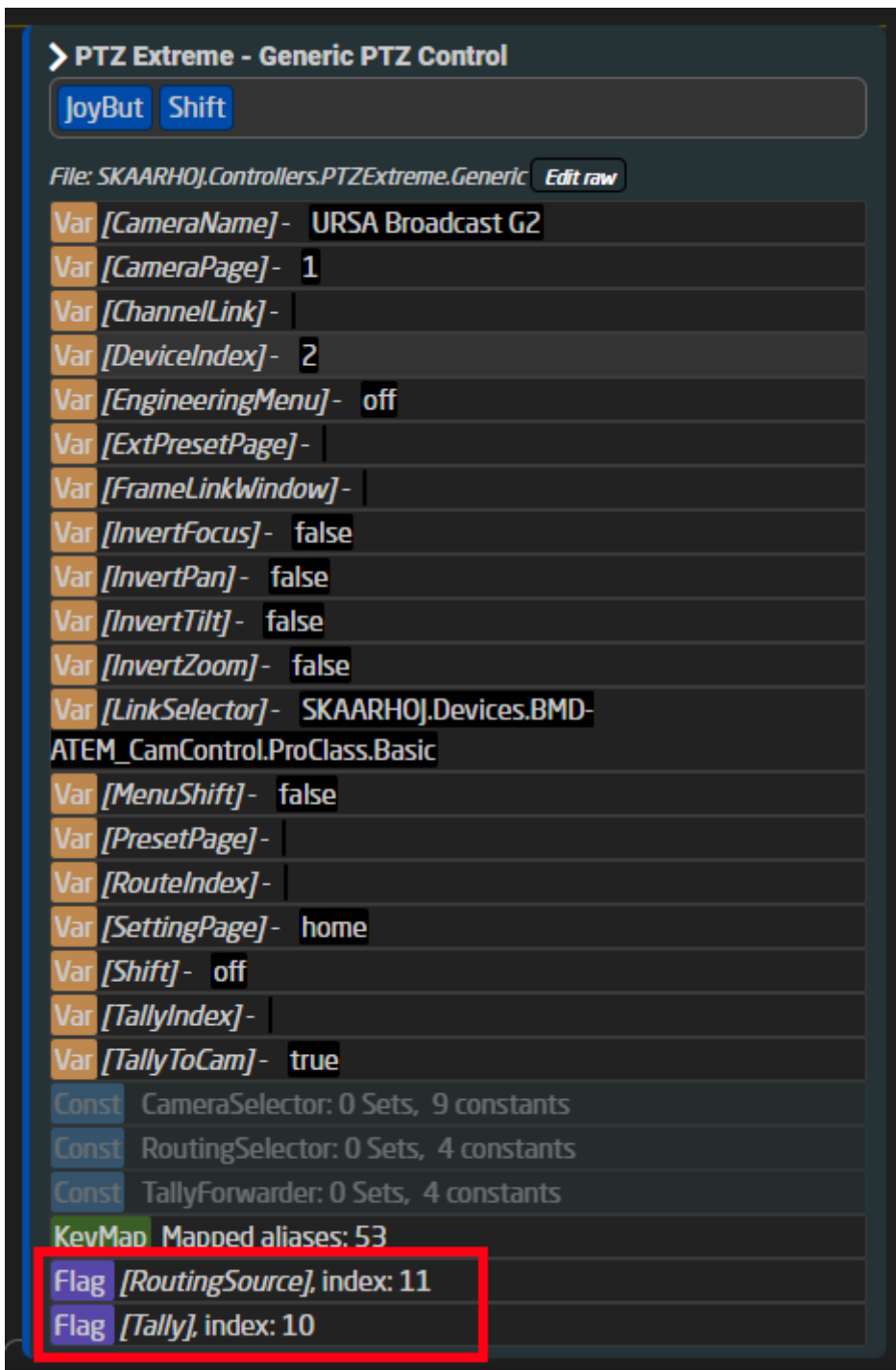
How to set a flag

The simplest way to set a flag is to use the normal "SKAARHOJ:HoldDown" behavior on a button/GPI pin or in a virtual trigger where you might use the tally status reported by your switcher.

When setting flags we are looking at an IORefrence structure that looks like this:

Flag:Tally/Red/Var:TallyIndex/
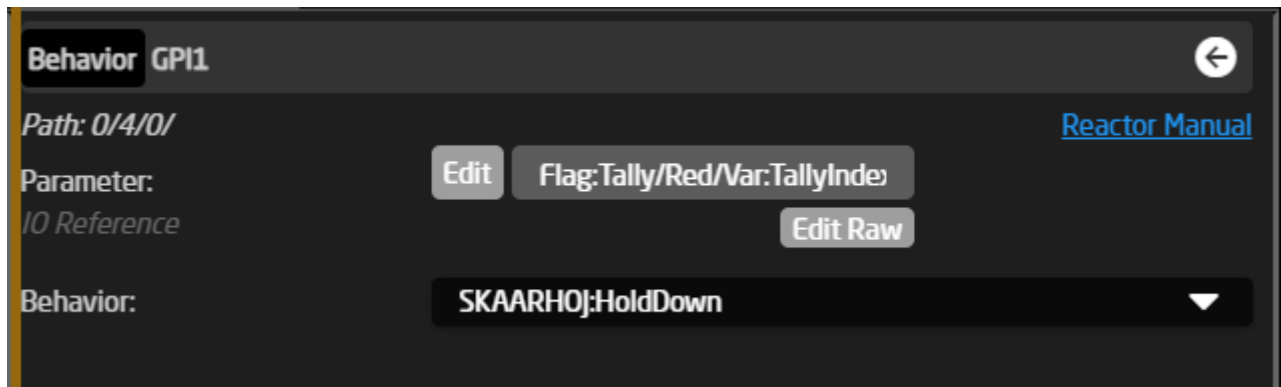
So to break it down we have:

Flag:[Group Name]/[Color]/[Index]

Often you will see us use a variable for the index. In most cases we use it together with a constant set where the user can fill in what routing input or tally index it should follow. For a mixer setup the tally index would correlate with what input number is active on the mixer.

Some examples of this in use

Here is a couple of examples of how these setups look, they are posted here in json format as it's the simplest way to get a condensed look at the setup:

From an RCP using the GPI input to set the red tally flag on the selected camera:
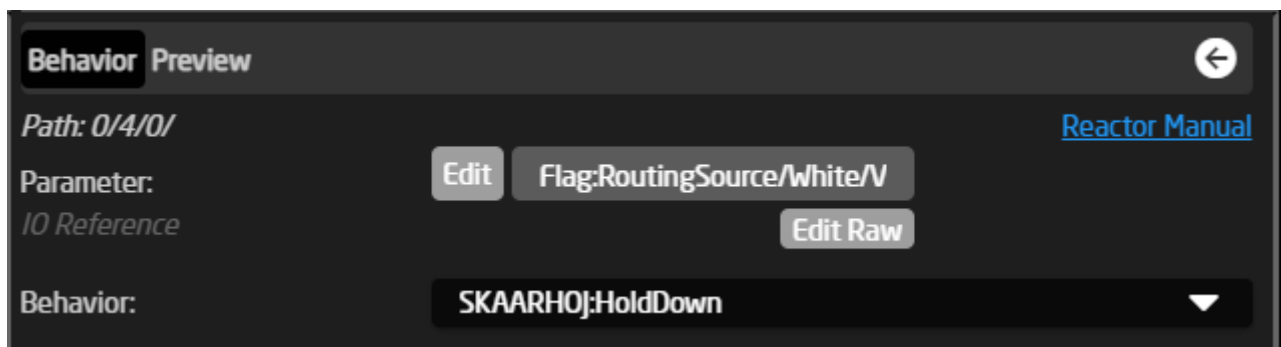

```
{
    "ParentID": "SKAARHOJ:HoldDown",
    "IOReference": {
        "Raw": "Flag:Tally/Red/Var:TallyIndex"
    }
}
```

And here for the preview button (same as pressing the joystick) on the same RCP, that sets a Routing Flag instead:


```
{
    "ParentID": "SKAARHOJ:HoldDown",
    "IOReference": {
        "Raw": "Flag:RoutingSource/White/Var:RouteIndex"
```
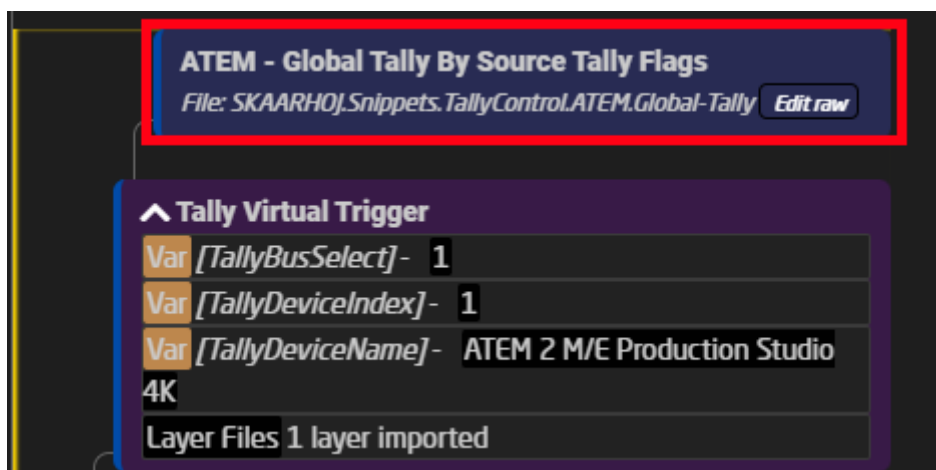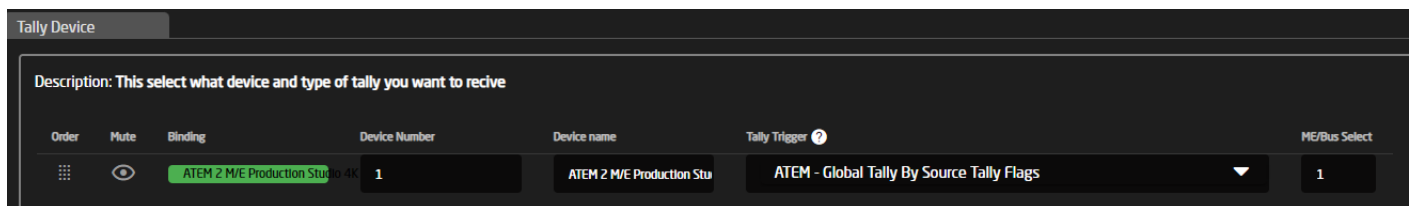
```
    },
    "FeedbackDefault": {
      "DisplayText": {
        "Title": "Preview"
      }
    }
  }
```

A more extreme setup of using a Virtual Trigger for setting the tally flags with tally status from an ATEM mixer:

This is the file you load when you select a tally device in the constant set:

Inside it has two Virtual Triggers that set the Program and Preview Tally and they do the same thing where they read the state of the tally parameter on the ATEM and store it into the flags on either the color red or green, depending on if it's the program or preview tigger.
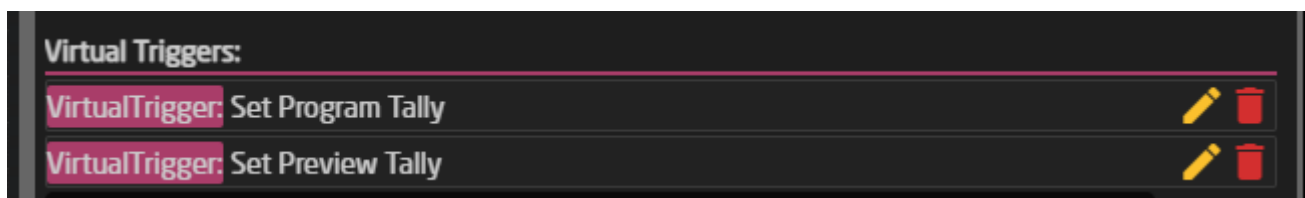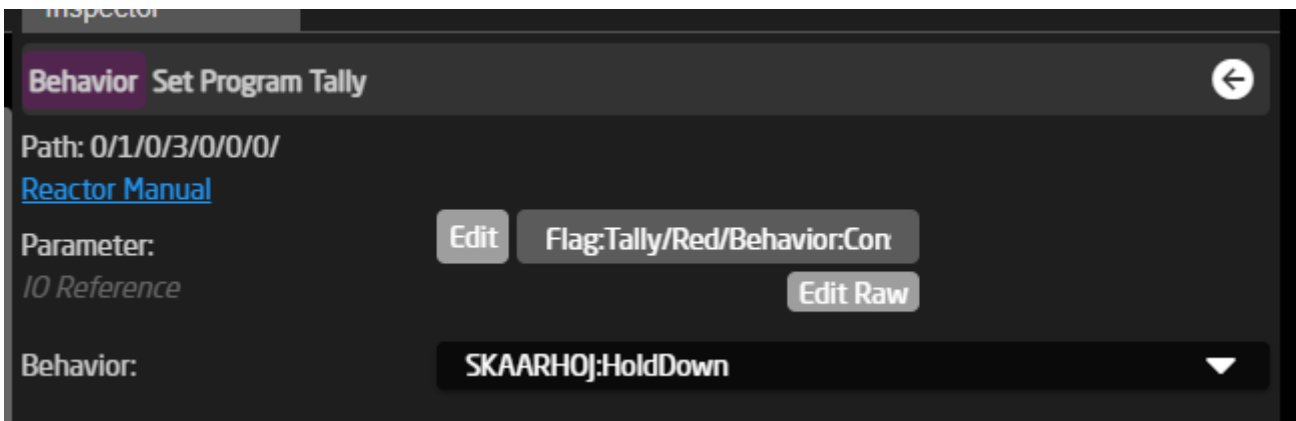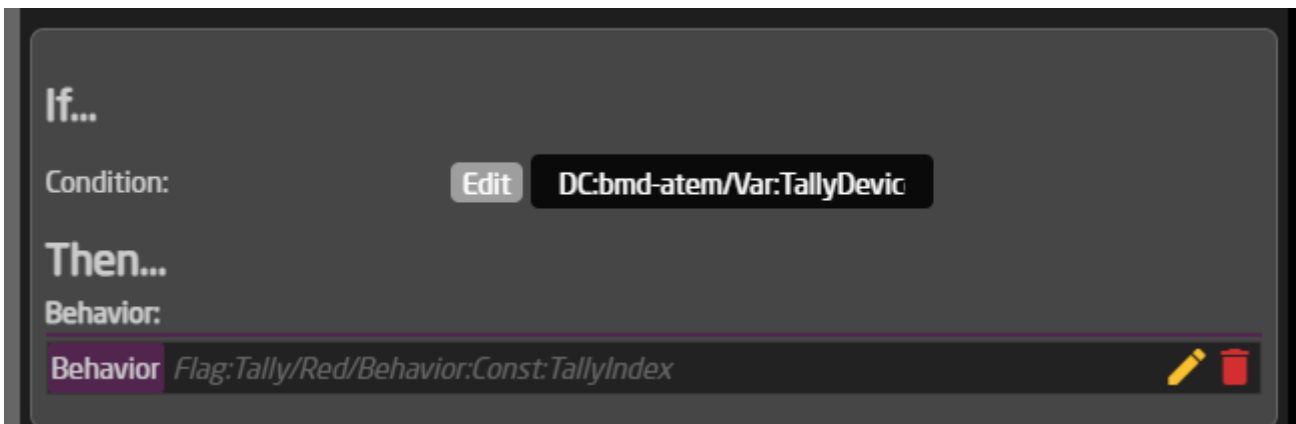
Full code for the ATEM Tally snippet:

```json
{
    "Name": "ATEM - Global Tally By Source Tally Flags",
    "MetaData": {
        "DeviceCoresInside": {
            "bmd-atem": []
        },
        "Priority": 150
    },
    "VirtualTriggers": [
        {
            "Name": "Set Program Tally",
            "ConstantSetReference": "CameraSelector",
            "Mode": "Binary",
            "BinaryActiveIf": "DC:bmd-
atem/Var:TallyDeviceIndex/TallyBySourceTallyFlags/Behavior:Const:TallyIndex/1 == true",
            "AnalogIOref": {},
            "HoldGroupFinalValue": {},
            "Behavior": {
                "ParentID": "SKAARHOJ:HoldDown",
```

```
            "IOReference": {
                "Raw": "Flag:Tally/Red/Behavior:Const:TallyIndex"
            }
        }
    },
    {
        "Name": "Set Preview Tally",
        "ConstantSetReference": "CameraSelector",
        "Mode": "Binary",
        "BinaryActiveIf": "DC:bmd-
atem/Var:TallyDeviceIndex/TallyBySourceTallyFlags/Behavior:Const:TallyIndex/2 == true",
        "AnalogIOref": {},
        "HoldGroupFinalValue": {},
        "Behavior": {
            "ParentID": "SKAARHOJ:HoldDown",
            "IOReference": {
                "Raw": "Flag:Tally/Green/Behavior:Const:TallyIndex"
            }
        }
    }
  ]
}
```

How to read a flag

Reading a flag is very similar to setting a flag, but now we look at the feedback value of it instead and compare it against if it's true or false.

The simplest way to read a flag is to use the "SKAARHOJ:Output" behavior on a LED or GPO pin or in a virtual trigger where you might use the tally status reported by your switcher.

Some examples of this in use

Here is a couple of examples of how these setups look, they are posted here in json format as it's the simplest way to get a condensed look at the setup:

First the simplest way of reading it is for a GPO pin on an RCP for providing routing support externally:

```
{
    "ParentID": "SKAARHOJ:Output",
    "IOReference": {
        "Raw": "Flag:RoutingSource/White/Var:RouteIndex"
    }
}
```

Another use case for a Tally LED on a Blue Pill or RCP where you will see a setup like this, where it reads both the Red and Green Tally bit:


```
{
    "Name": "Tally LED - Selected",
    "Description": "Show tally for selected tally index (in TallyIndex variable, using Tally Flag group)",
```

```
"IOReference": {},
"FeedbackDefault": {
    "Intensity": "Dimmed"
},
"FeedbackConditional": {
    "20": {
        "ActiveIf": "Flag:Tally/Green/Var:TallyIndex == true",
        "Color": {
            "ColorCode": "GREEN"
        },
        "Intensity": "On"
    },
    "30": {
        "ActiveIf": "Flag:Tally/Red/Var:TallyIndex == true",
        "Color": {
            "ColorCode": "RED"
        },
        "Intensity": "On"
    }
}
}
```
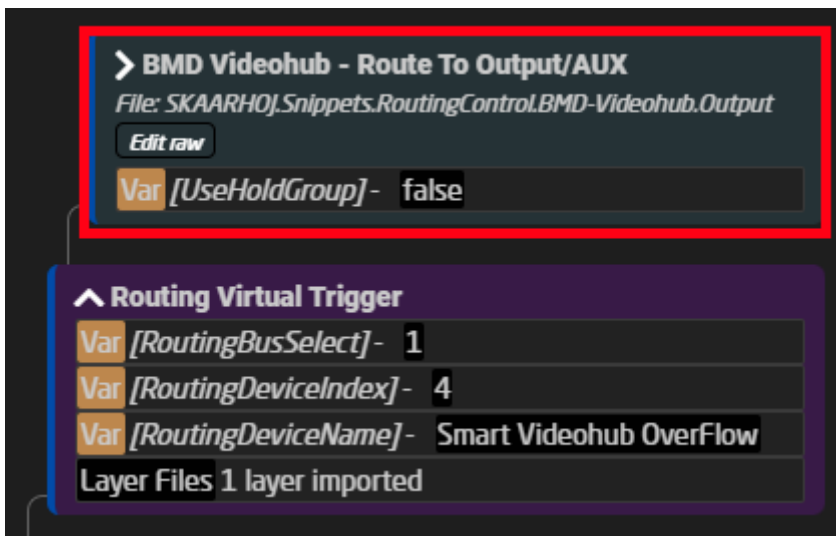
The last two main use cases that we will look at is reading a flag and then sending some command to a BMD Video hub for providing routing control, this is using virtual triggers:

This is the file you load when you select a routing device in the constant set:
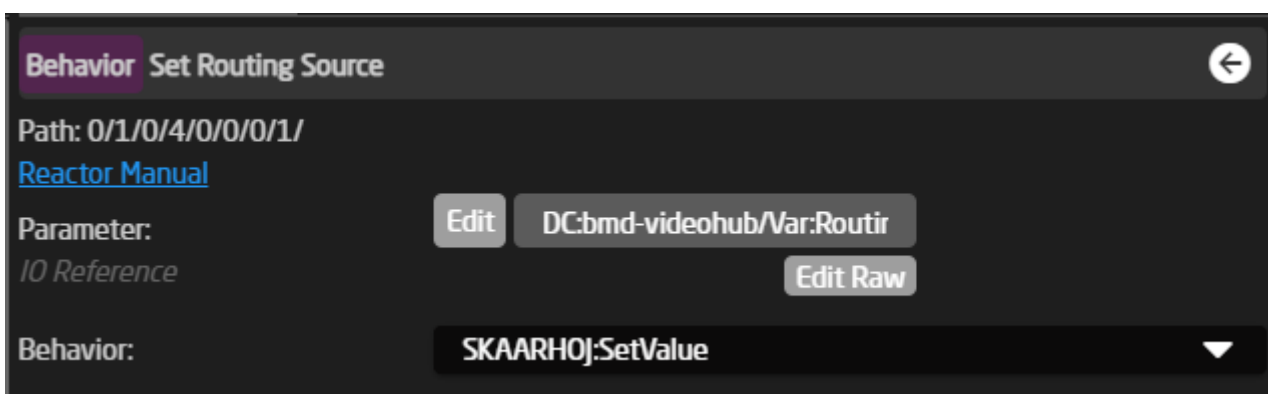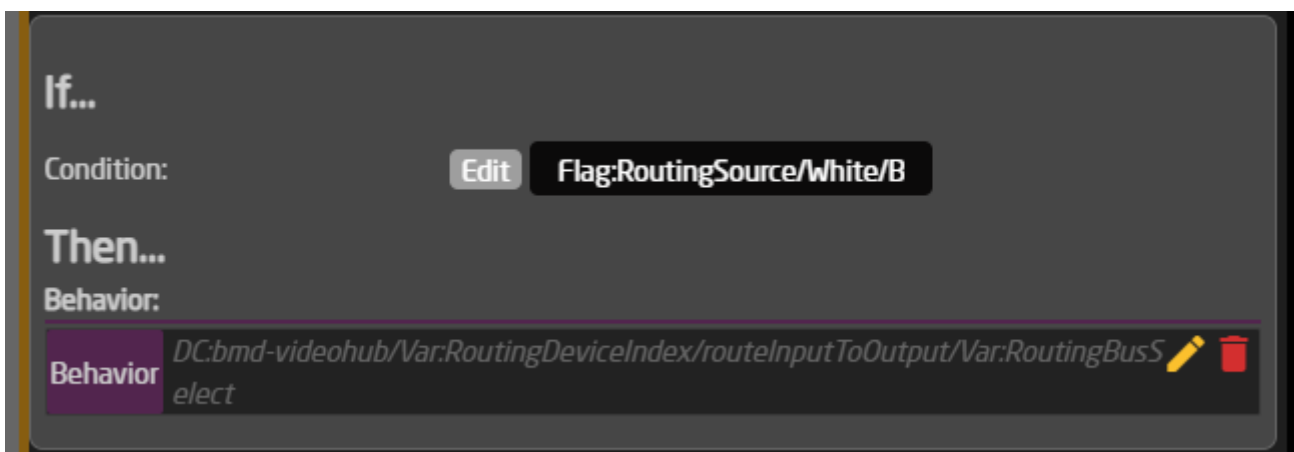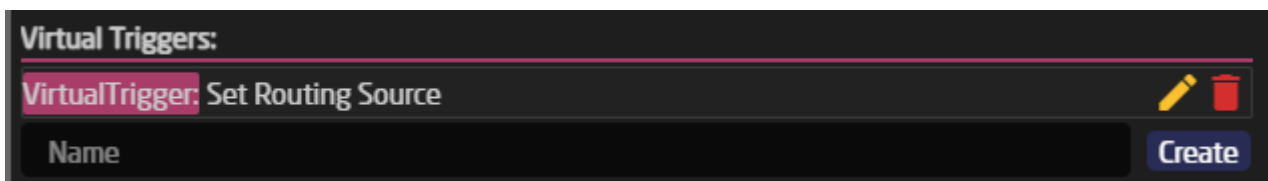
Inside it has virtual triggers that read the stored routing flag and sends the routing selected and sends it to the parameter on the Videohub.


```
{
    "Name": "Basic V-Trigger",
    "ActiveIf": "Var:UseHoldGroup:Current == false",
```

```
    "VirtualTriggers": [
      {
          "Name": "Set Routing Source",
          "ConstantSetReference": "CameraSelector",
          "Mode": "Binary",
          "BinaryActiveIf": "Flag:RoutingSource/White/Behavior:Const:RouteIndex == true",
          "AnalogIOref": {},
          "HoldGroupFinalValue": {},
          "Behavior": {
            "ParentID": "SKAARHOJ:SetValue",
            "IOReference": {
                "Raw": "DC:bmd-videohub/Var:RoutingDeviceIndex/routeInputToOutput/Var:RoutingBusSelect"
            },
            "EventHandlers": {
                "trigger": {
                  "BinarySetValues": {
                      "Raw": "Behavior:Const:RouteIndex"
                  },
                  "IOReference": {}
                }
            }
          }
      }
    ]
}
```

The last main use case to show of here is doing the same as for routing but for forwarding the tally states directly to any camera connected. We will look at the setup for the Canon PTZ cameras in this example:

This example is taken from an ETH-GPIO Link, but it's the exact same code used on all PTZ controllers.
Inside the Camera Selector you will have a field for the "Tally Forward Config", this should auto fill for you and look something like this:


Description: Please Specify what cameras and indexes you want to be able to control from the GPIO Inputs and Outputs

| Order	Mute	Binding	Device Number:	Name/Label	Tally Forward Config: ❓	Tally Index	Route Index
⠿	👁	EOS-C300	1		Canon XC - Tally ▼	1	1
⠿	👁	CR-N300	2		Canon XC - Tally ▼	2	2
					NEW		

This will then load up like this inside the config, in something that should look familiar by now, as

it's the same setup as for the tally and routing setups mentioned above.
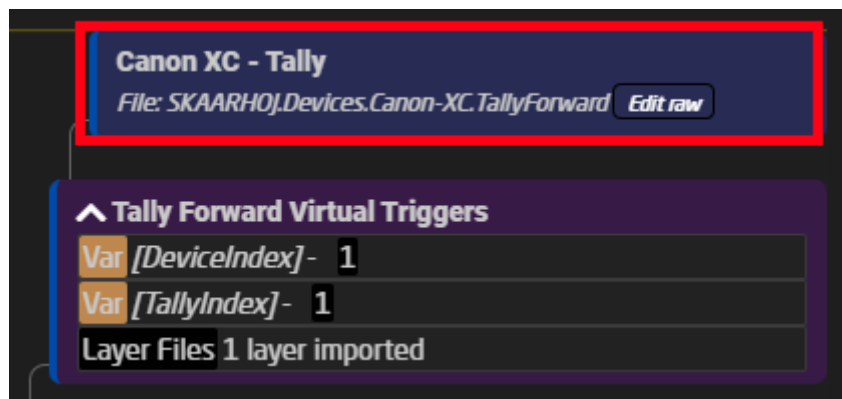
The big difference here is that this setup will very a lot depending on how the cameras handle tally, but one thing they will all have in common is that they have a trigger for disabling tally altogether, and that they all use the same flags, red and green for the tally feedback. On the Canon PTZ's you will see a total of 4 Virtual triggers. We will only look at the "Set Red" trigger.
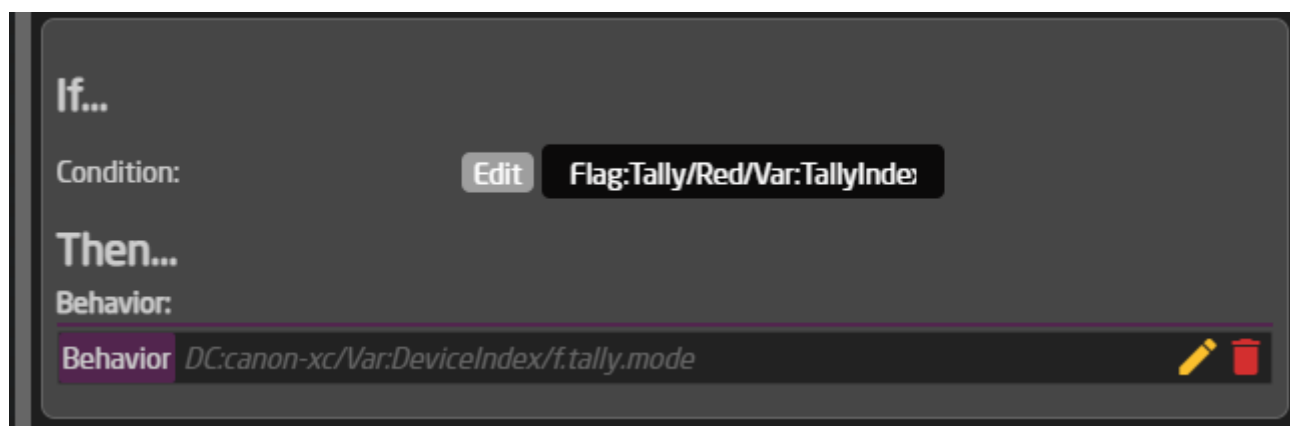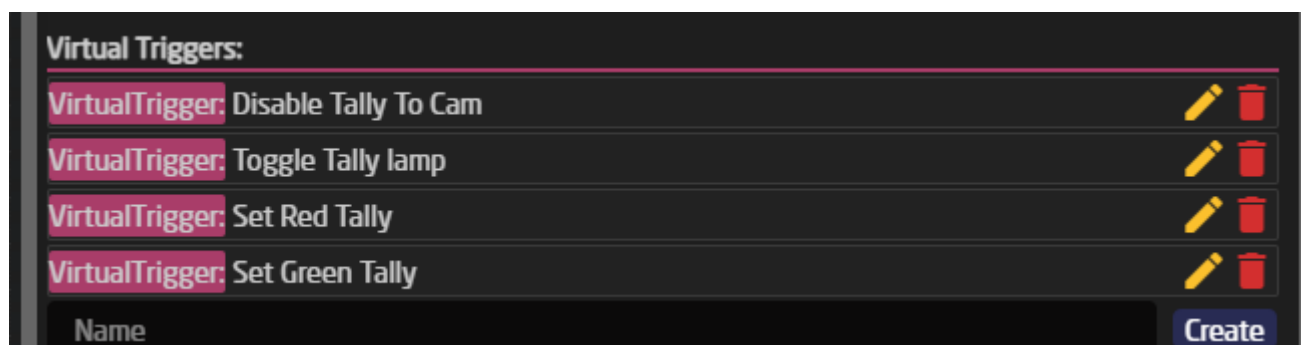
```
{
    "Name": "Set Red Tally",
    "ActiveIf": "Var:TallyToCam == true",
    "Mode": "Binary",
    "BinaryActiveIf": "Flag:Tally/Red/Var:TallyIndex == true",
    "AnalogIOref": {},
    "HoldGroupFinalValue": {},
    "Behavior": {
        "ParentID": "SKAARHOJ:SetValue",
        "Constants": {
            "MatchValue": {
                "Values": [
                    "1"
                ]
            }
        },
        "IOReference": {
            "Raw": "DC:canon-xc/Var:DeviceIndex/f.tally.mode"
        }
    }
}
```

As seen above, it's basically the same thing the difference is only that here we have a "MatchValue" to select that it's the program we want to set on the Canon PTZ, if we want to swap the colors on the PTZ, then it's just a matter of changing this from a 1 to a number 2 and the reverse on the preview trigger. This is because of the options available on the camera, to look more at these option please refer to the parameter list for the Canon-XC core.

Here is the full JSON for this layer with all 4 virtual triggers, in their current form:

```json
{
    "Name": "Canon XC - Tally",
    "MetaData": {
        "DeviceCoresInside": {
            "canon-xc": []
        },
        "Priority": 100
    },
    "VirtualTriggers": [
        {
            "Name": "Disable Tally To Cam",
            "Mode": "Binary",
            "BinaryActiveIf": "Var:TallyToCam == false",
            "AnalogIOref": {},
            "HoldGroupFinalValue": {},
            "Behavior": {
                "ParentID": "SKAARHOJ:SetValue",
                "Constants": {
                    "MatchValue": {
                        "Values": [
                            "0"
                        ]
                    }
                },
                "IOReference": {
                    "Raw": "DC:canon-xc/Var:DeviceIndex/f.tally"
                }
            }
        },
        {
            "Name": "Toggle Tally lamp",
            "ActiveIf": "Var:TallyToCam == true",
            "Mode": "Binary",
            "BinaryActiveIf": "Flag:Tally/Red/Var:TallyIndex == true || Flag:Tally/Green/Var:TallyIndex == true",
            "AnalogIOref": {},
            "HoldGroupFinalValue": {},
            "Behavior": {
                "ParentID": "SKAARHOJ:HoldDown",
                "IOReference": {
```

```
              "Raw": "DC:canon-xc/Var:DeviceIndex/f.tally"
            }
          }
        },
        {
          "Name": "Set Red Tally",
          "ActiveIf": "Var:TallyToCam == true",
          "Mode": "Binary",
          "BinaryActiveIf": "Flag:Tally/Red/Var:TallyIndex == true",
          "AnalogIOref": {},
          "HoldGroupFinalValue": {},
          "Behavior": {
            "ParentID": "SKAARHOJ:SetValue",
            "Constants": {
              "MatchValue": {
                "Values": [
                  "1"
                ]
              }
            },
            "IOReference": {
              "Raw": "DC:canon-xc/Var:DeviceIndex/f.tally.mode"
            }
          }
        },
        {
          "Name": "Set Green Tally",
          "ActiveIf": "Var:TallyToCam == true",
          "Mode": "Binary",
          "BinaryActiveIf": "Flag:Tally/Green/Var:TallyIndex == true && Flag:Tally/Red/Var:TallyIndex == false",
          "AnalogIOref": {},
          "HoldGroupFinalValue": {},
          "Behavior": {
            "ParentID": "SKAARHOJ:SetValue",
            "Constants": {
              "MatchValue": {
                "Values": [
                  "0"
                ]
```

```
                }
            },
            "IOReference": {
                "Raw": "DC:canon-xc/Var:DeviceIndex/f.tally.mode"
            }
        }
    }
  ]
}
```

Flags used in Routing Triggers

Routing Triggers in SKAARHOJ PTZ Controller configurations utilize a flag named "RoutingSource". This flag is used to communicate between the camera selector and a virtual trigger, providing information about the input source to be routed to a specific video switcher or hub when a camera is selected. The "RoutingSource" flag index varies across different controller configurations. For instance, the index for a PTZ Extreme is set to "11":
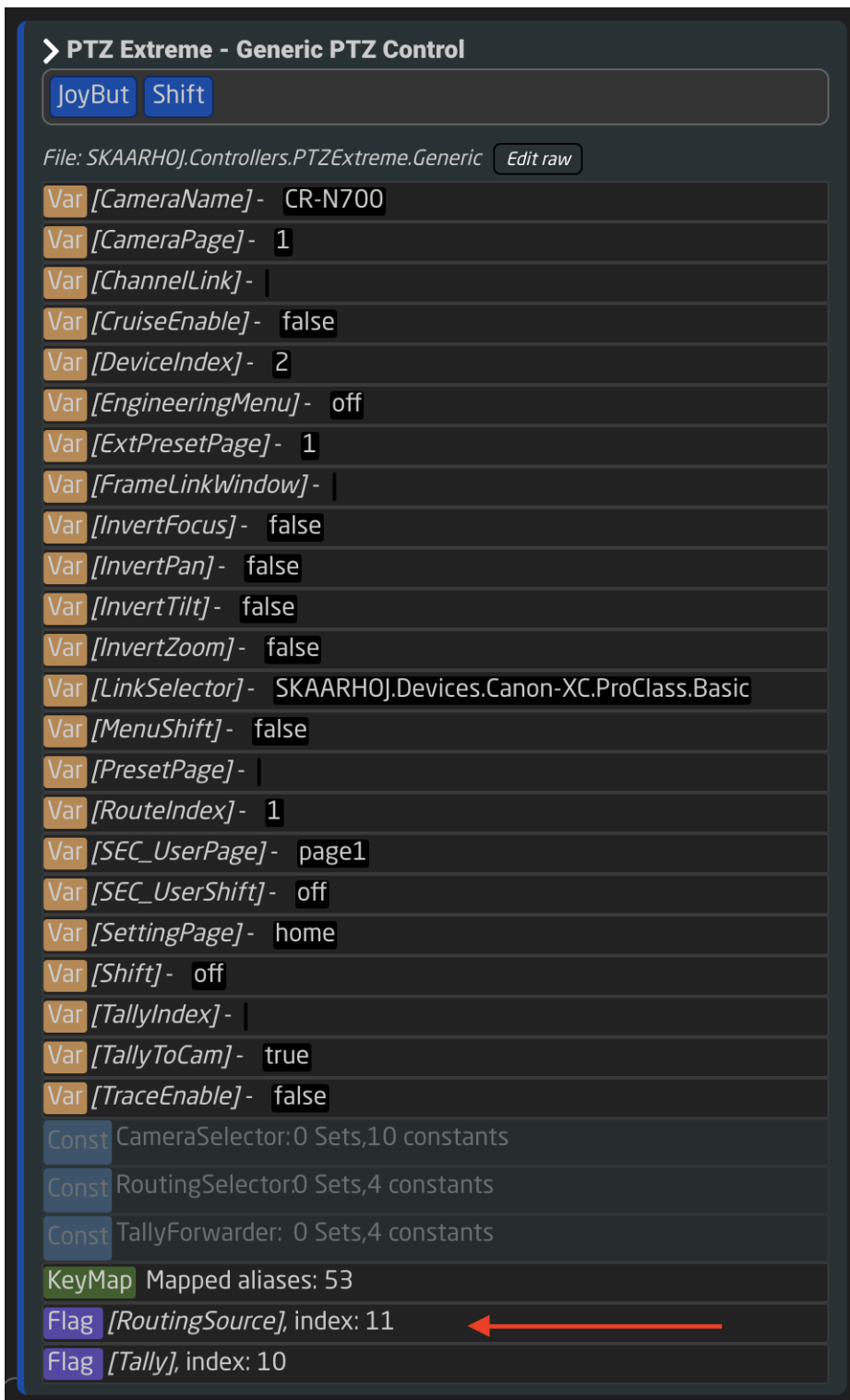
When managing multiple PTZ Extremes with the same configuration, a conflict arises as each controller's Routing Trigger setup uses the same index (11). This duplication can lead to operational issues. That same is true for other controllers and their configurations being shared.

Solution to the Index Conflict

The solution lies in creating a flag by the same name in the tree on a parent layer to the shared configuration. The "RoutingSource" flag (a "TreeDweller" property in the configuration tree) is not set with the "Always Define" flag. This allows for an alternate definition of "RoutingSource" on the parent layer of the configuration. When defined on this parent layer, it prevents the value inside

the PTZ Extreme Configuration from overriding. Thus, for any additional configurations of similar controllers, you can assign a different index number to the "RoutingSource" flag by setting it on the parent layer. See below:

Adding a new flag is done from the tree by clicking the layer name and adding a flag in the Inspector:

Inspector

Flag *[]* - RoutingSource

Layer: Default PTZ Extreme Configuration

Name: RoutingSource

Group Index: 111

Description:

Always define: ☑

Revision #6
Created 7 October 2022 10:02:14 by Andreas Hauge Thomsen
Updated 7 November 2024 13:34:21 by Lukas Bachschwell