Blue Pill / Reactor

March 2021

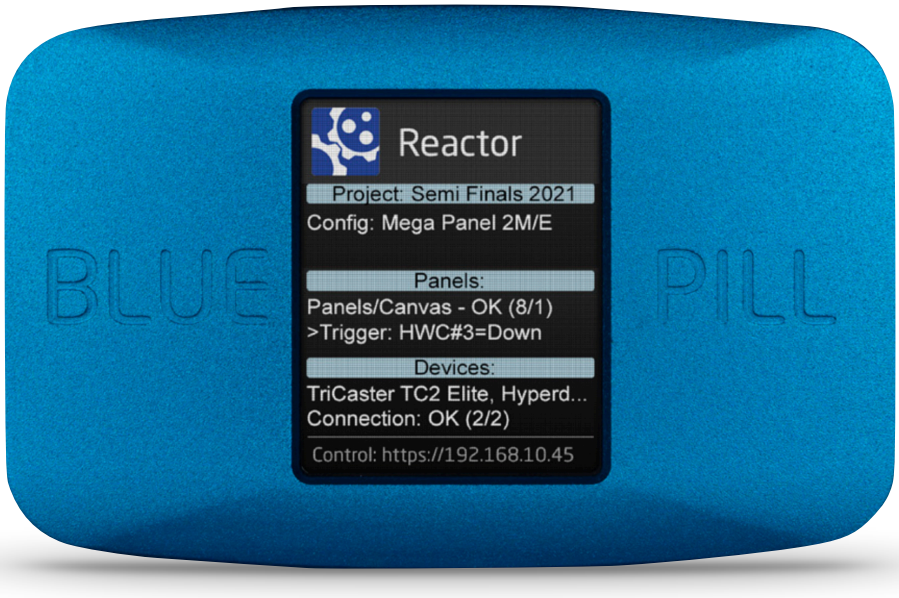
Congratulations on heading into the future of media production control with SKAARHOJ! This document is the first written source that describes the core concepts of the Blue Pill platform and the Reactor software running on it.

Blue Pill generally refers to our new line of software technologies while it is also a specific hardware product which is essentially a small Swiss Army Knife server product with some bells and whistles.

Device Cores are a well known concept from our UniSketch OS where they represent features that gets compiled into the firmwares on panels. In the Blue Pill world, device cores are binaries running on the underlying skaarOS (our custom industrial Linux build).

Reactor is an application running on Blue Pill which connects to panels, connects to device cores and through configurations make them work together. In this sense, Reactor can be considered a similar technology to what UniSketch is via its configuration web UI. In principle Reactor is just one potential panel management application that could run on Blue Pill, but in reality it's likely to be the only game in town for quite a while. The point is: The system is more modular, scalable and flexible that ever.

This document goes through the basics to the advanced sides of SKAARHOJ's new and exciting technology, Blue Pill / Reactor. Enjoy.



Contents

Introduction

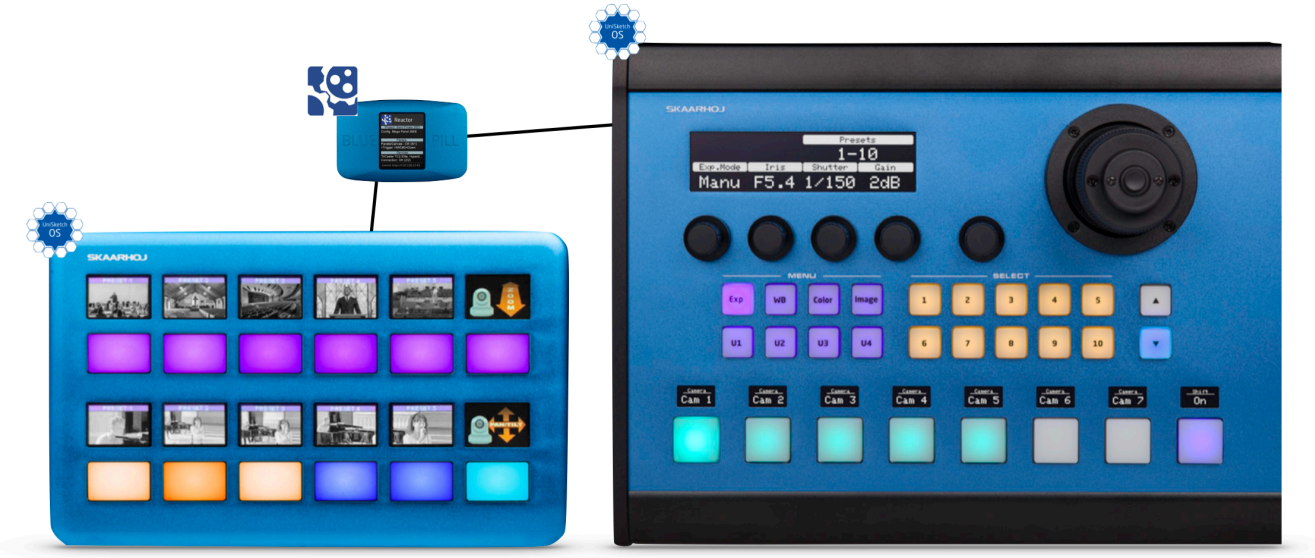
Modularity Reimagined

Blue Pill enhances the UniSketch ecosystem by combining multiple controllers into a single seamless controller. For example, a PTZ Pro can be combined with a Frame Shot and have the Frame Shot show thumbnails for preset recall. The selected camera on the PTZ Pro will drive the preset selection on the Frame Shot because to the Blue Pill they are one single panel.

Blue Pill talks to UniSketch panels via the Raw Panel device core (TCP Server) on UniSketch. This device core has been around for a long time as a way for third party developers to interface with UniSketch controllers. With this road tested technology, Blue Pill takes control of the UniSketch panels and offers unprecedented possibilities that UniSketch could never do on its own. This includes:

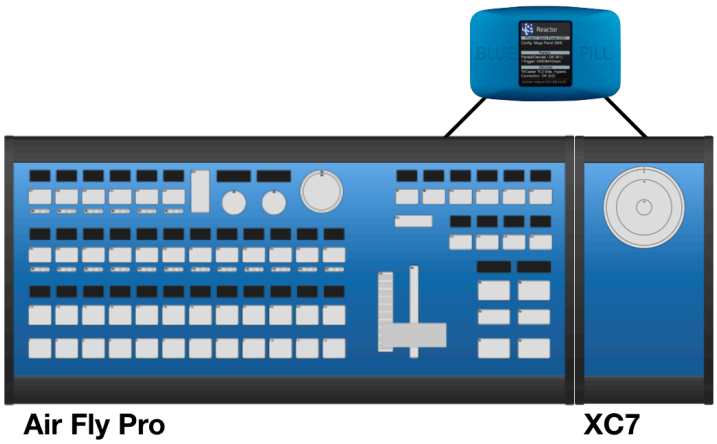
- Virtually unlimited SKAARHOJ panels combined (seamless modularity)
- Virtually unlimited devices being integrated (cameras, switcher, routers, etc.)
- New integrations not available on UniSketch (like from Canon, RED, Panasonic etc.)
- System provided features such a presets (macros / time lines), virtual triggers etc.

Use of the word *unlimited* is meant to imply that the software is designed in a limitless way, but obviously memory and CPU power will always limit any computer system in reality. But the clever networked design of the Blue Pill platform means there will always be ways to distribute the workload in advanced and extreme cases. For example; Blue Pill device cores talk with Reactor over a network socket and usually does so on the same device (the Blue Pill) but if need be, this can also happen over actual network fibers and cables and even public internet if desirable.

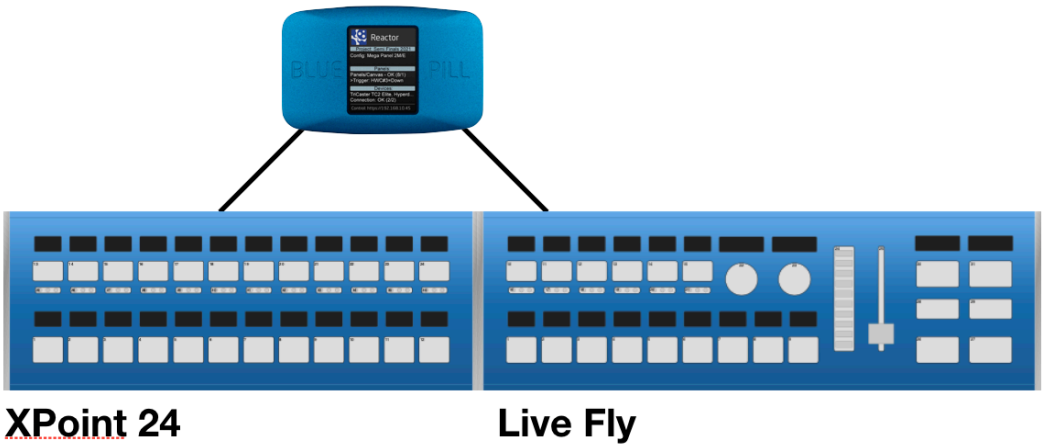


Apart from the Frame Shot making a perfect wing man to SKAARHOJ's legendary PTZ controllers, other SKAARHOJ products receive a whole new potential by expanding existing products.

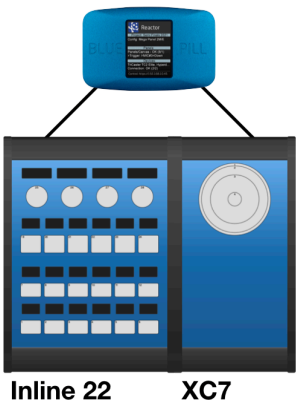
Combine an Air Fly Pro with a XC7 joystick to add PTZ control to a switcher panel.

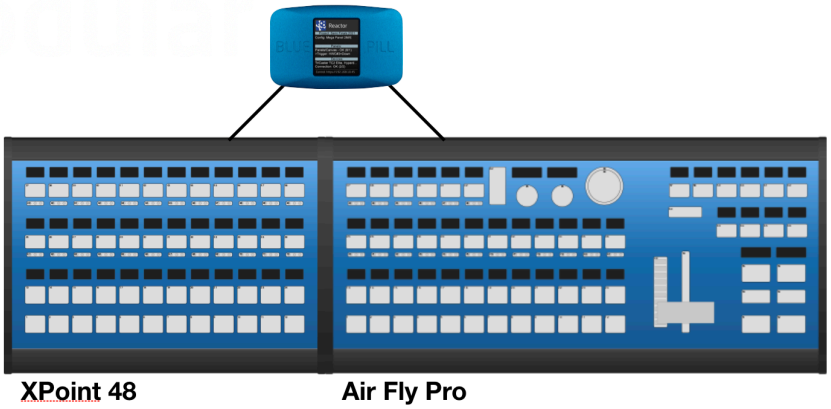


Extend a Live Fly with XPoint 24 to have direct access to more switcher sources:



Combine Inline 22 and XC7 to have a full PTZ controller! Or Add XPoint 48 to Air Fly Pro to access more sources directly - like for the Live Fly example above. And of course add XC7 to this combo too.

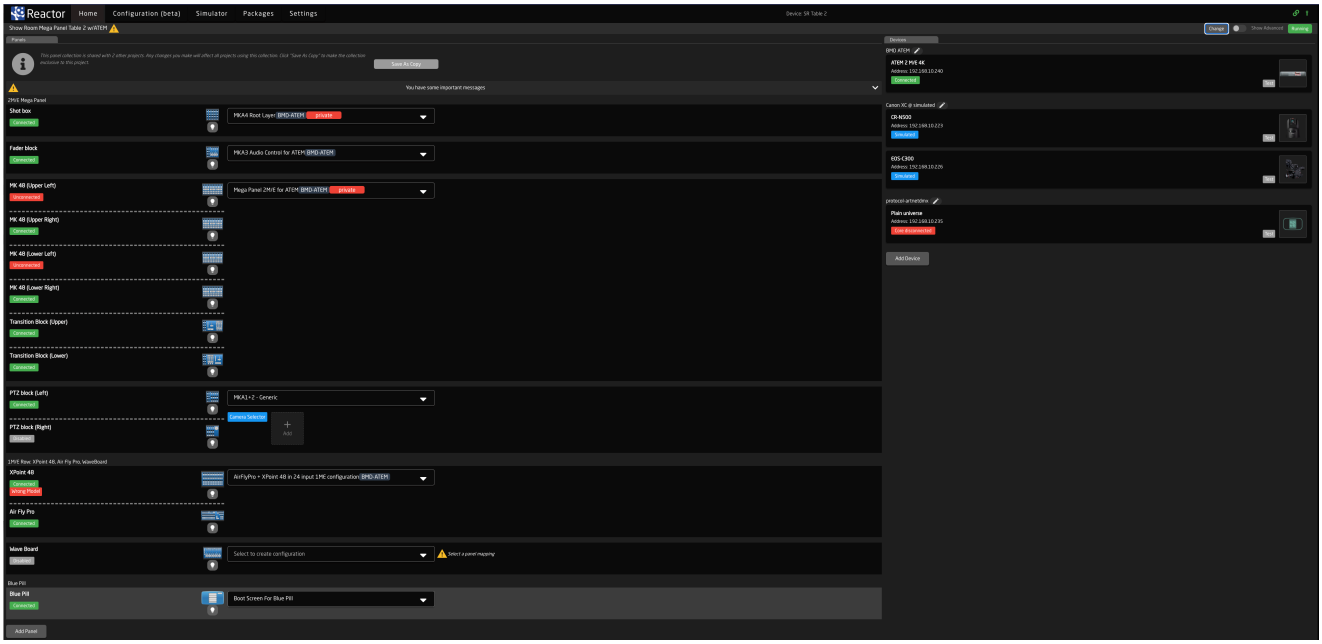




A really good example is the SKAARHOJ mega panel which consists of typically 8-10 individual controllers brought together by Blue Pill in this way:

Reactor

The Home screen of Reactor currently looks like this, though from user feedback we are still in a stage of refinement so it is subject to change. It consists of a **Project** title at the top and two sections with **Panels** and **Devices**. Importantly, within the Panels sections is panel **Mapping** and set up of **Constant Sets**.



- **Projects** - are individual combinations of panels, devices, and configurations that can easily be switched between or edited as needed. With Blue Pill it is possible to even share aspects of projects with other projects on the same Blue Pill or even export them to be used in other set ups.
- **Panels** - or a Panel Group - consists of one or more panels organized in one or more groups. A group of panels can be in either an abstract way (like all the panels in Control Room A) or in a very mechanical sense (the panels in a mega panel, arranged at specific X-Y coordinates to reflect their physical positions adjacent to each other). Every panel has an integer ID (Panel ID) by which it is referenced in the configuration. Within panels the mapping or hardware component assignments are also selected.
    - **Mapping** - define a virtually unlimited layer structure on which hardware component behaviors (HWC behaviors) define the response of a given hardware component on a panel. Layer visibility driven by conditions from a system such as the value of a variable or device core parameter determines which behaviors are active at any time. Configurations can be split out into files which can be reused over and over again in different contexts to multiply the effect of the time spend on a single configuration. This is true both for the individual end users who configure their Blue Pill, and to the level of SKAARHOJ's expert engineering and support teams when they design embedded libraries of configurations for controllers and popular devices. Configurations that can be imported are found in four categories:
        - Embedded configurations are distributed with Reactor software releases.
        - Local configurations are sharable on the same Blue Pill across individual configurations (coming soon)
        - Private configurations are included layer configurations which are only accessible to a single configuration. Trying to save changes to an embedded, subscription or local configuration will result in a copy being made as a private configuration. This system design allows a user to maximize the benefit of standard configurations while maintaining full flexibility to customize the controller at will.

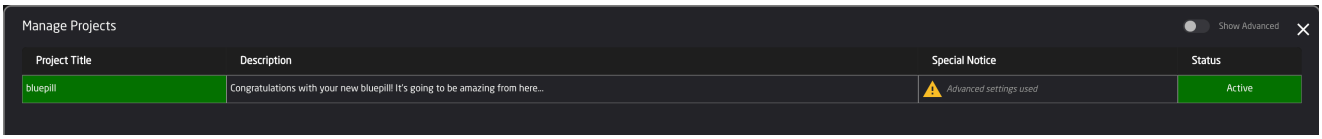
- **Constant Sets** - allows for flexible mapping of inputs for certain configurations to provide flexibility and speed. You will often find these used as camera and input selectors that allows for easy mapping of cameras and inputs, and where you can drag and drop the order and rename buttons on the fly.

- **Devices** - consists of one or more devices such as cameras, switchers, routers, software endpoints etc. They are organized by the **device core** they are controlled by. Unlike UniSketch where each model of VISCA camera from different manufacturers has its own device core, in Blue Pill a device core can contain specific support for *many* models. There is for example only one VISCA device core in Blue Pill, but it has information on more than 30 VISCA cameras from different manufacturers and can make integrations for each one. Likewise there is only one Panasonic PTZ device core but it supports about ten Panasonic PTZ camera models.

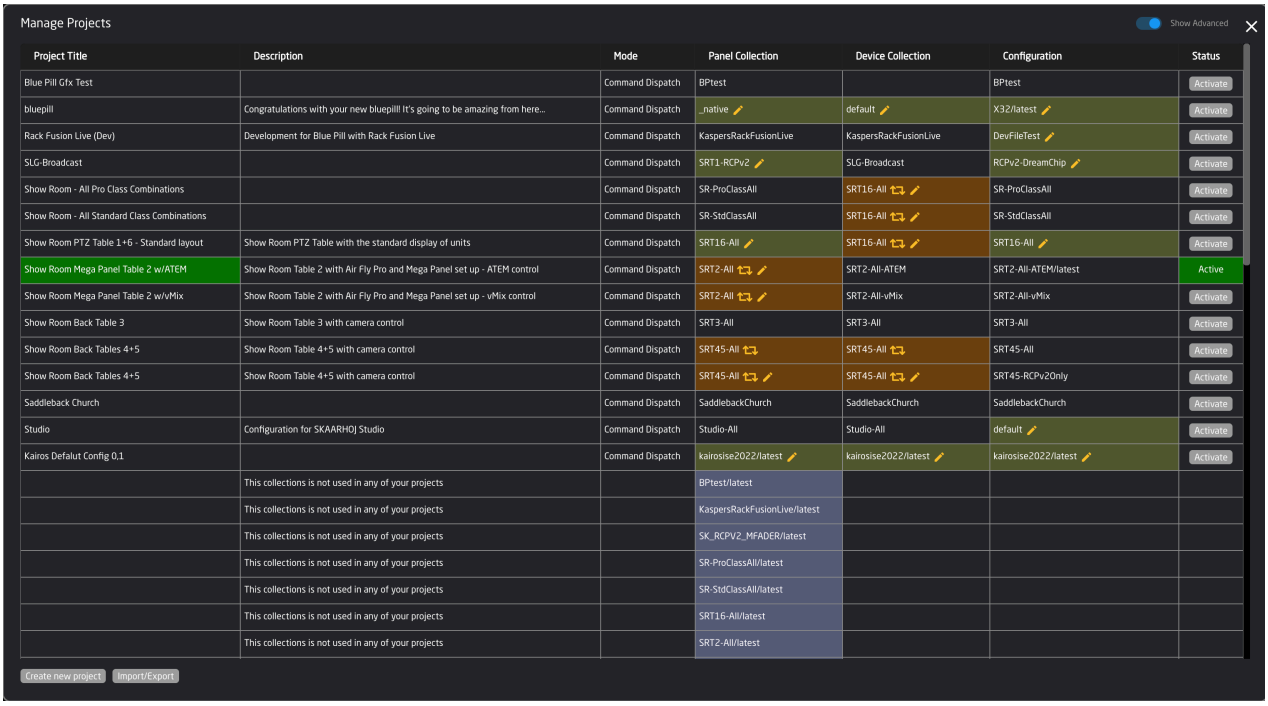
Projects

Projects are the unit that have a title, description and references to a panel collection, a device collection and a configuration. This design provides the flexibility to use the same set of SKAARHOJ panels but quickly change to another location of devices with other IP addresses for example. Or keep the same devices and panels, but run a different configuration serving a different user group of operators.

When a Blue Pill ships it has a default configuration is as below. It is set in simple view with only the basic information available.



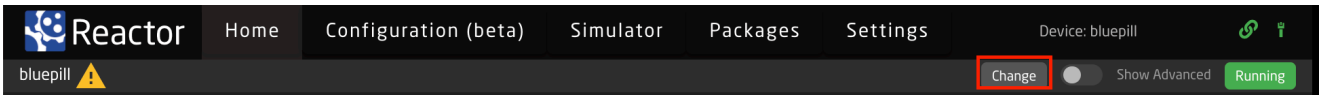
As time goes by a user may develop more and more configurations which are essentially combinations of existing and unique panel collections, device collections and configurations. Seen below is what it would look like in advanced view, where all the details are available.



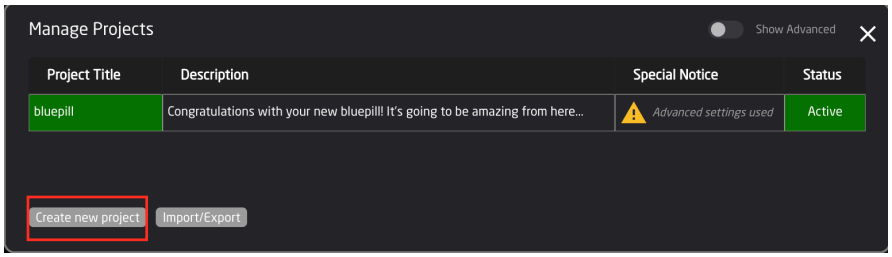
Unlike UniSketch where a change of configuration and device cores needed an online connection to SKAARHOJ server, on Blue Pill everything is stored locally in Reactor. Only software upgrades or installations of non-existing device cores and applications need communication with the server.

Creating Projects

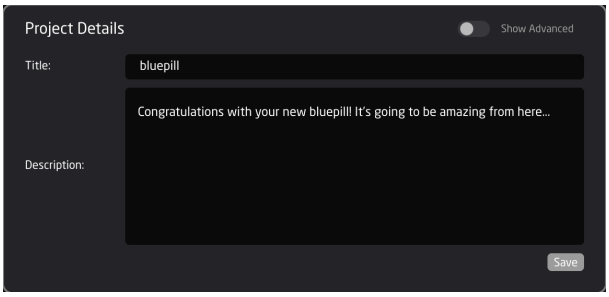
To create a new project click Change Project within Reactor Home.



Select Create new project on the projects pop up.

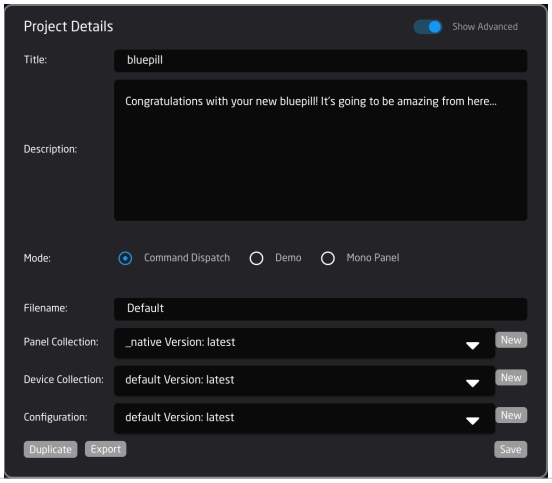


There are two main parts of the project: Name and Description.



Name	Customizable project name
Description	Optional description of the project for reference

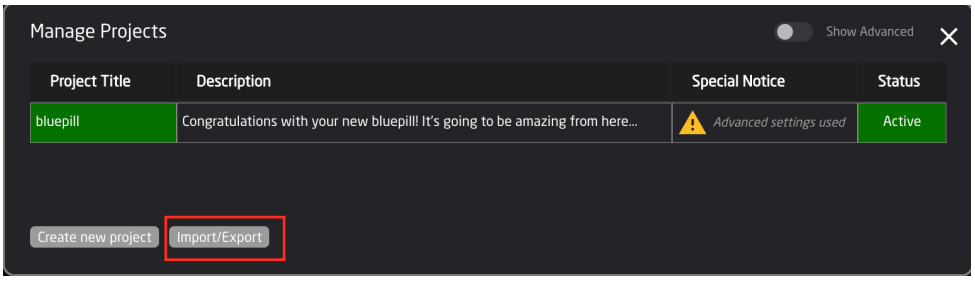
Advanced brings with it additional project options.



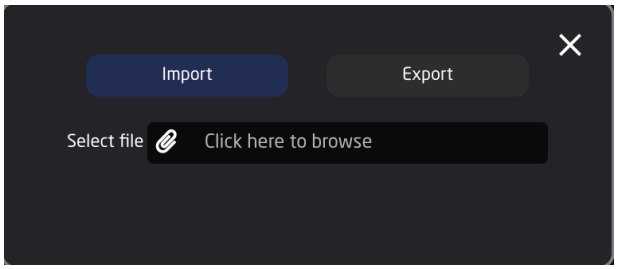
Mode	1. Command Dispatch for normal operation 2. Demo runs a color splash across the connected controllers 3. *Mono panel description coming soon*
File Name	Master name for the file when making new collections, configurations, and exports.
Panel Collection	Allows for the sharing of panel collections across projects on a Blue Pill
Device Collection	Allows for the sharing of device collections across projects on a Blue Pill
Layer Configuration	Allows for the selection of a specific layer files in the configuration across projects on a Blue Pill
Duplicate Project	Makes a copy of the project.
Export Project	Creates an export project file. Allows for the exclusion of various parameters like: Exclude Panels, Exclude Devices, Exclude Config, and Disable Config Consolidation

Import/Export

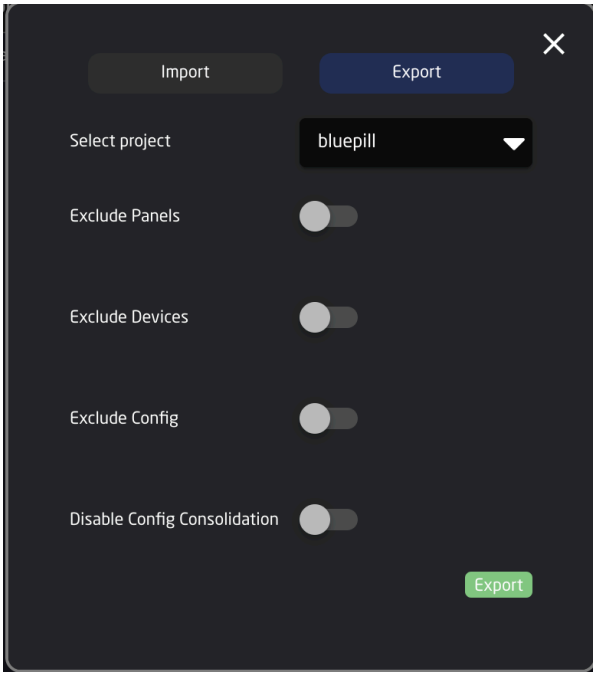
It is also possible to import and export projects to more easily share between different Blue Pill devices. This option is available next to create new projects in the Manage Projects pop up.



Import allows for a simple file upload.

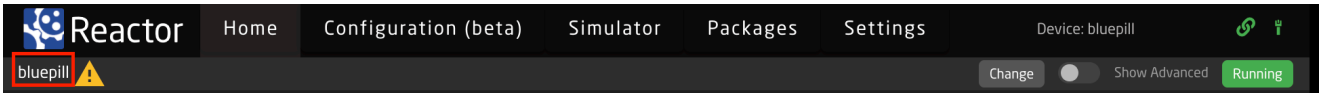


Export allows for the selection of what to include or exclude from the file. This can be the panels, devices, or configuration. This is specific project based and not for the entire Blue Pill.



Editing Projects

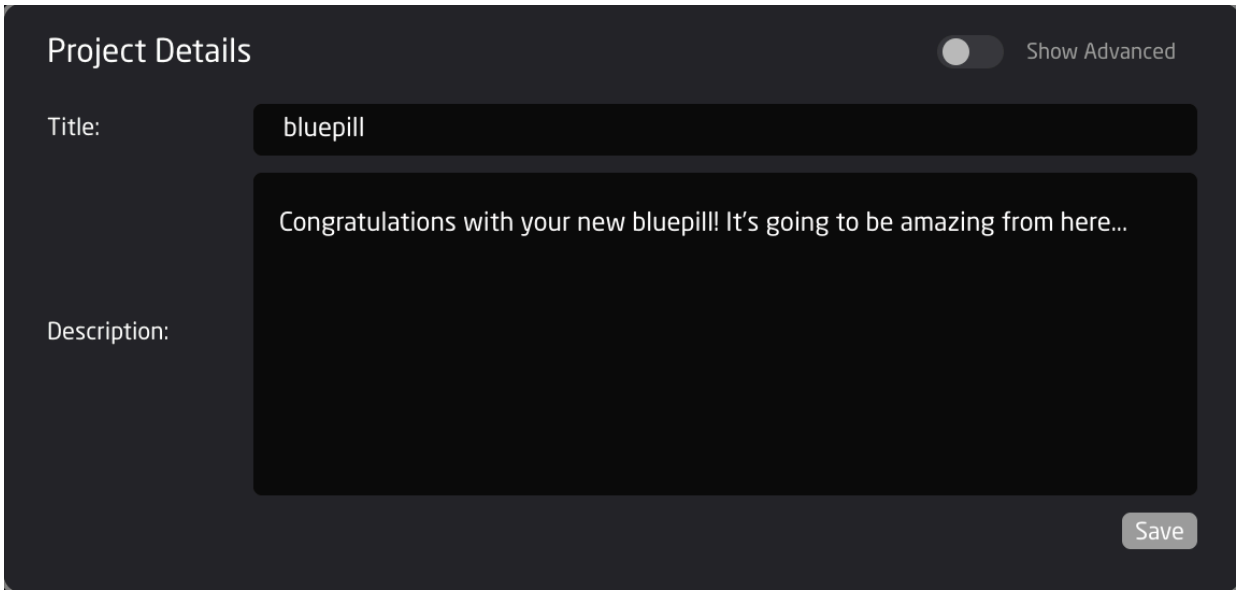
To change the details of the project click the project name within Reactor Home



Or within any of the project detail boxes on the projects pop up.

Project Title	Description	Special Notice	Status
bluepill	Congratulations with your new bluepill! It's going to be amazing from here...	*Advanced settings used*	Active

It is then possible to customize the name and description of the project. Press Save.

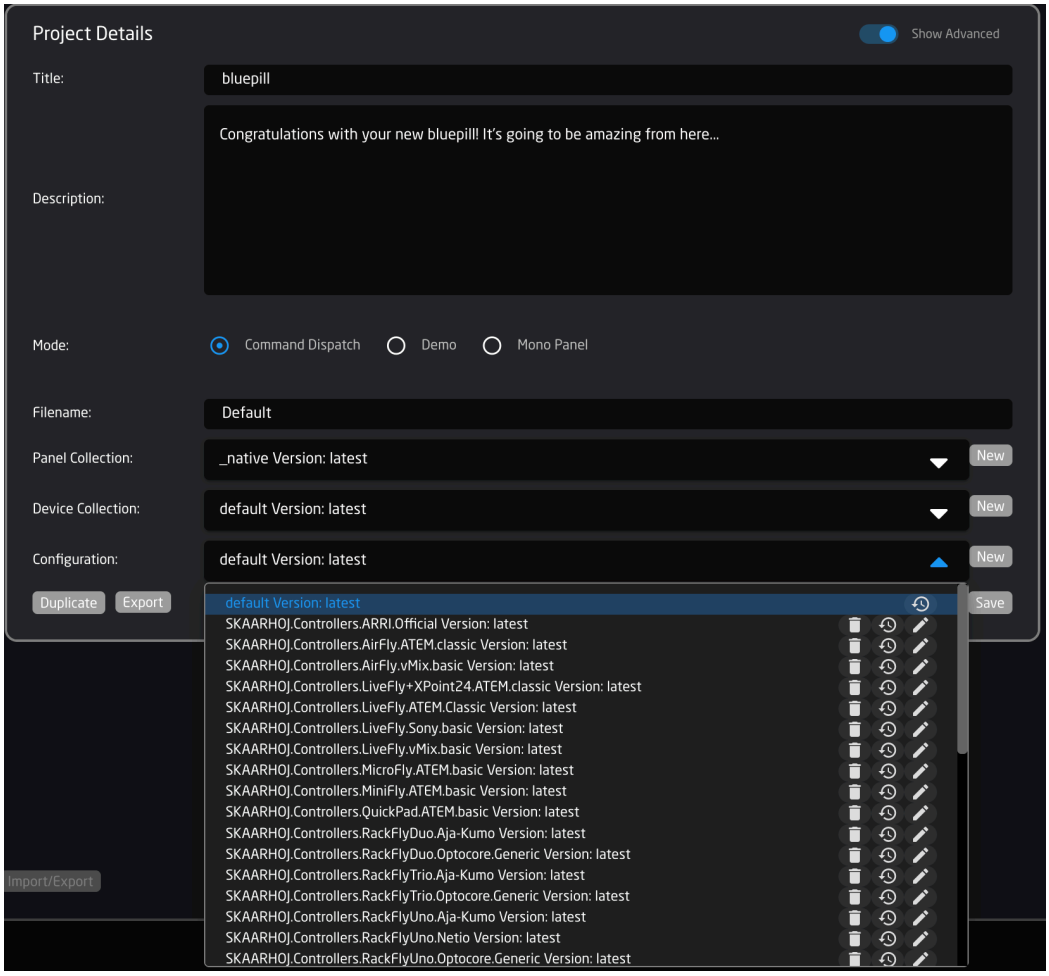


Enabling Advanced view brings about extra options for editing the project outside of the basic details, the same as would be seen when creating a new project.

Layer Configuration Version Look Up

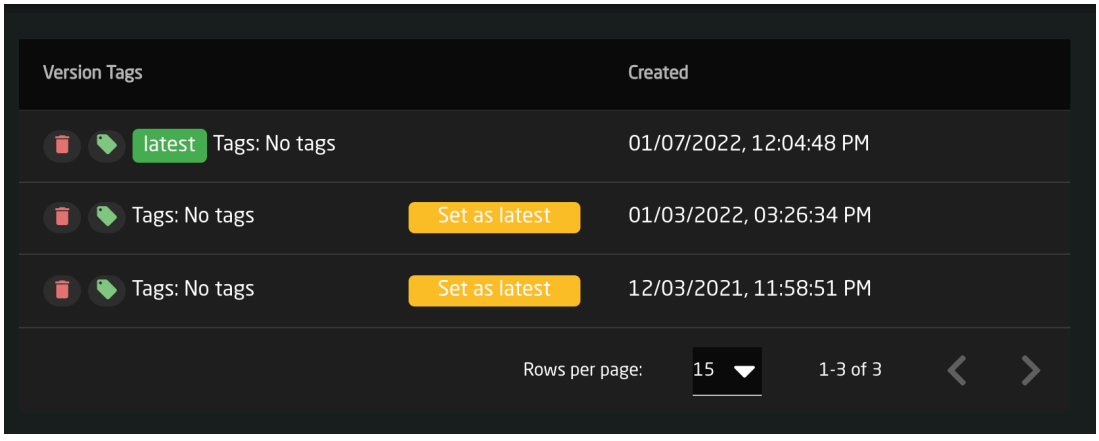
In the layer configuration section, it is possible to access autosaved versions of the configurations. This is useful when a lot of changes have been made and it is necessary to roll things back a little.

When in the Layer Configuration section, click on the little clock symbol next to the layer to see previous versions.



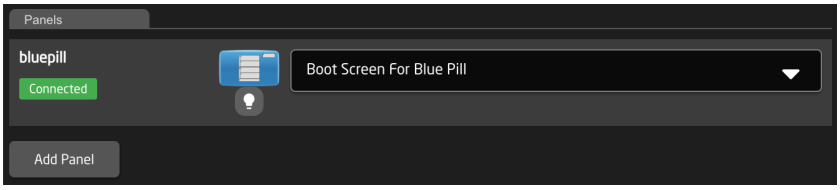
Available versions are: every change from the last 24 hours and as 1 file per day for 10 days.

It is also possible to add tags or comments to individual versions. When a tag or comment is added to the version, it will no longer be deleted based on the default time scale and will persist.



Panels

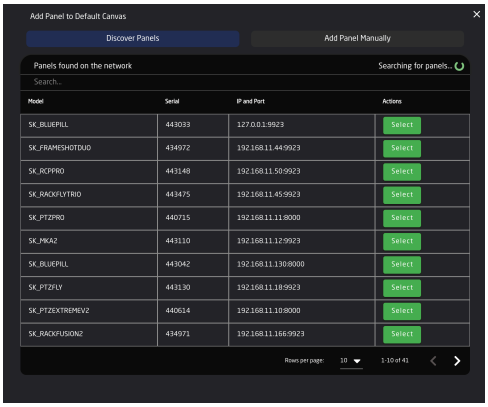
The Panels section is where new SKAARHOJ panels are added to the panel collection of the project. By default the Blue Pill will already be in the Panels section, this is because the Blue Pill is technically also a panel.



There are two ways a panel can be added, Auto Discover and Manually.

Auto Discover

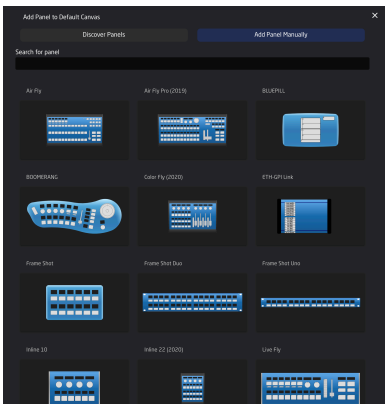
Panels can easily be discovered by mDNS look ups on the same subnet the Blue Pill is on. This makes it very easy to discover and include a panel in the configuration. mDNS will usually search the current subnet, but with a correctly configured Blue Pill, panels on other subnets can easily be included too as long as the IP and port is known to the user.



Pressing SHIFT + the green Select button will allow for multiple panel selection.

Manually Added

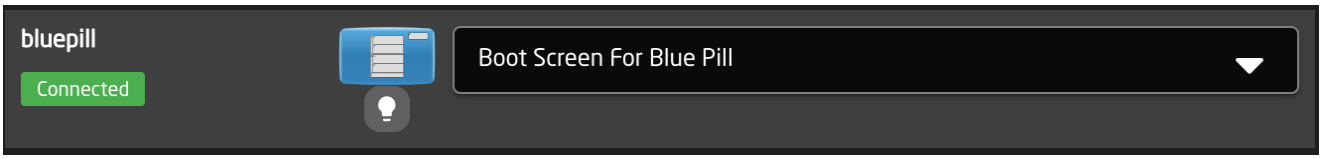
Panels can be picked from SKAARHOJs included library of products. Following a manual choice of panel, the user will have to enter the IP address himself including any constraints desired. See the Panel Details section for set up.



Pressing SHIFT + the green Select button will allow for multiple panel selection.

Panel Details

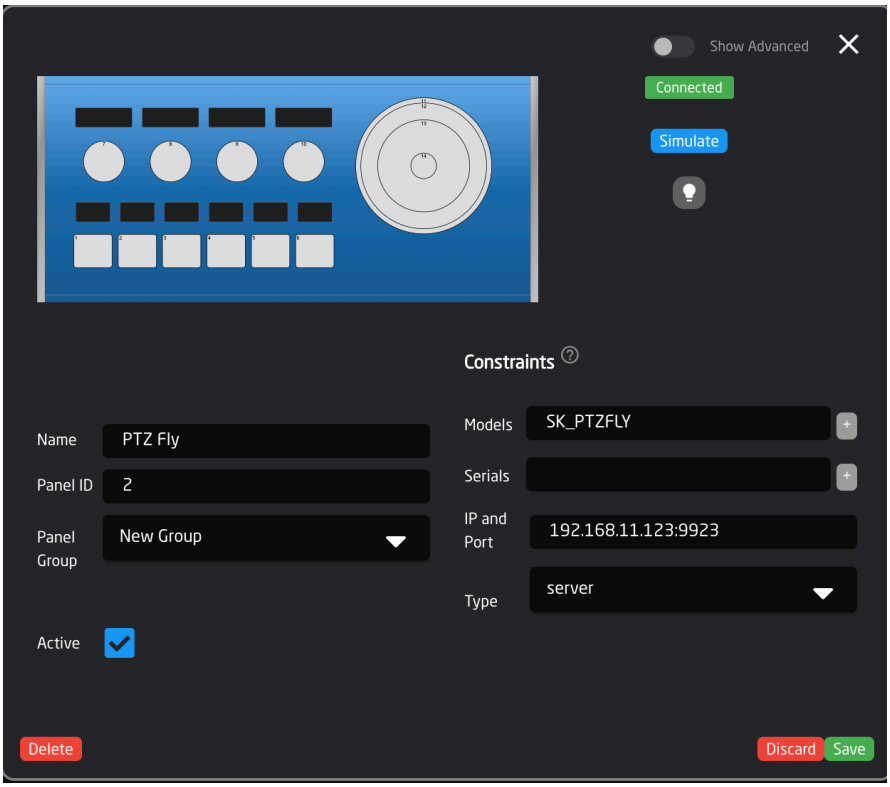
The immediate view of a panel in the Home Screen show its name, connectivity status, mapping, and any available constant sets. There is also a small light bulb button for lighting up the panel for quick real world identification.



Clicking on the panel name will open up the details. Here the name, ID number, panel group and Active status of the panel can be changed. By default the controller name will be the same as the model name, the ID will be auto generated in order, panels will per default be in their own groups and the status will be active. A controller's status needs to be active for use.

Constraints are requirements regarding the panel. Here it is determined which models are allowed, which serials are allowed, which IP and port numbers can be used and if the panel is of the server or client type (Raw Panel protocol related). Empty constrains will allow any value, but consider that a configuration always maps functionality to hardware component IDs which will change with the panel model for obvious reasons, so it's rarely a good idea to omit constrains such as the allowed models.

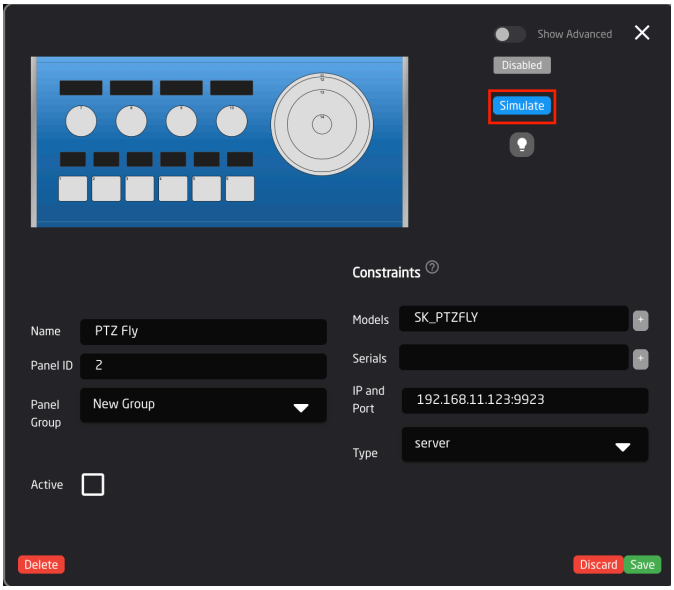
By default our controllers connect using port 9923. It is important to include the port number, separated from the IP address by a colon (:), for example 192.168.11.124:9923



From inside the panel details it is also possible to delete the panel or simulate the panel if it is not immediate available for testing.

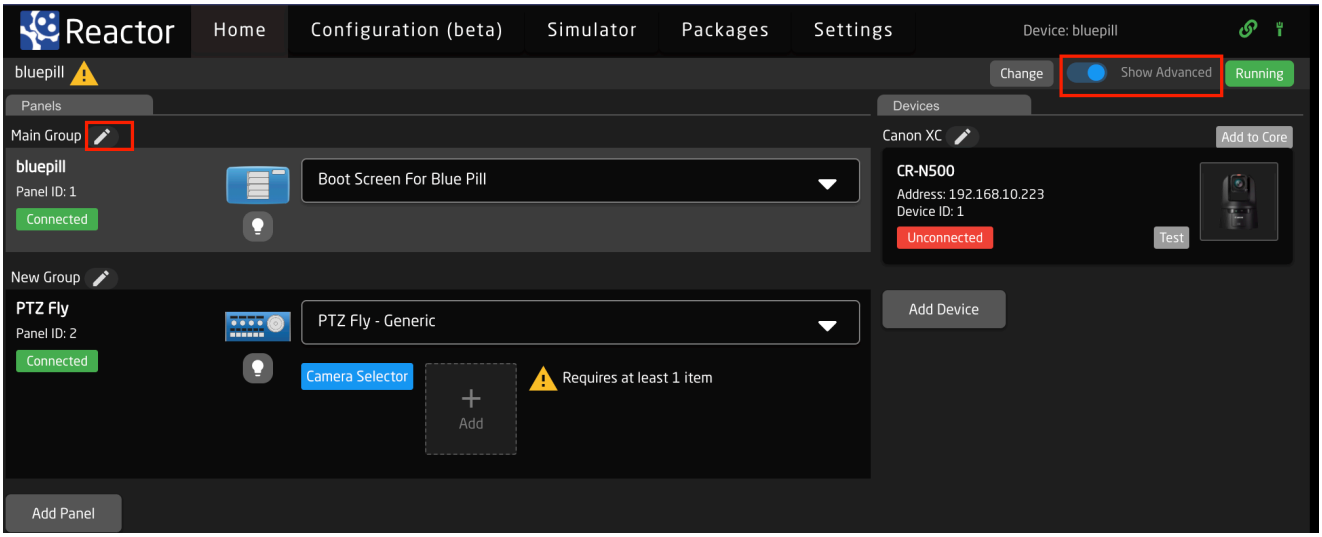
Panel Simulate

The Panel Simulate option available in the panel details allows for an interactive simulation of the panel's behaviors based on the configuration/mapping and devices without needing an established connection to the devices. Pressing Simulate will immediate relocate to the Simulator page for this panel, it is best to wait on simulating a panel until mapping has been selected and devices added.



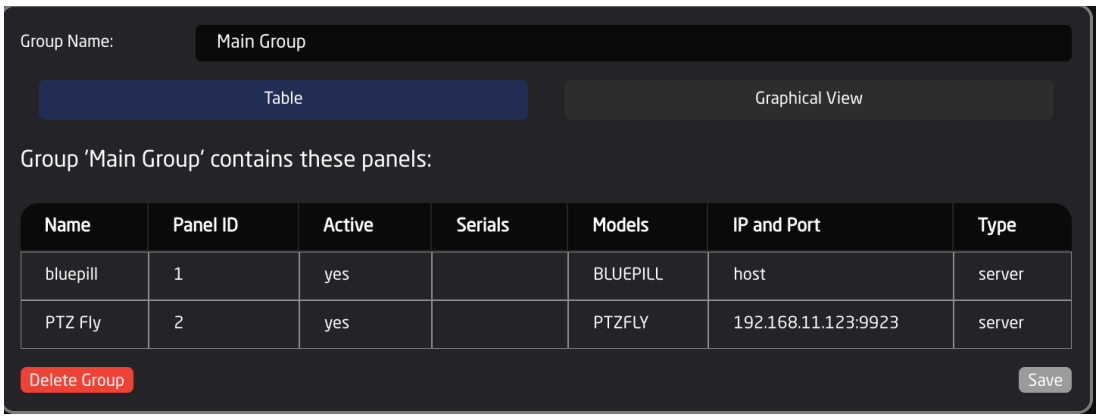
Panel Groups

Multiple panels can be arranged in groups. This allows for an easy visual breakdown of controller set ups. To access the information, set the Home screen to Show Advanced, then click on the edit icon next to the group name.

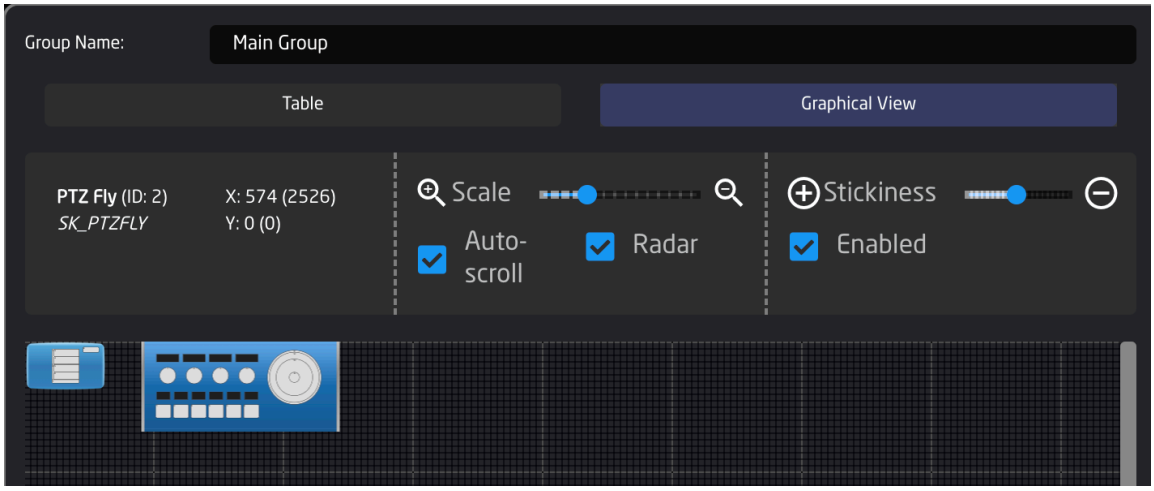


Inside the group details it is possible to edit the group name, view the group as a table, or in a adjustable graphical view.

In Table view the basic information of the set up is given including the panel name, id, status, serial number, model, IP, Port, and connection type.



In Graphical View it is possible to arrange the physical layout of the controllers as might be seen in the controller room. This view is used by both the Configuration (beta) page and the Simulator page for layout.

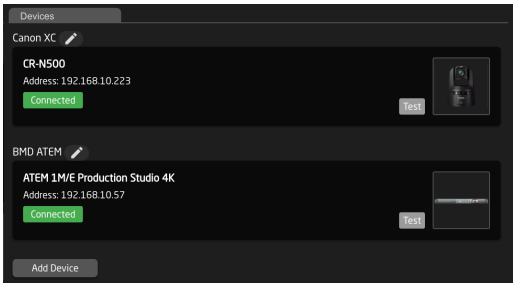


Confirm Connection

A panel that has established a connection with a blue pill will no longer display "Waiting for Blue Pill" or "Waiting for Raw Panel".

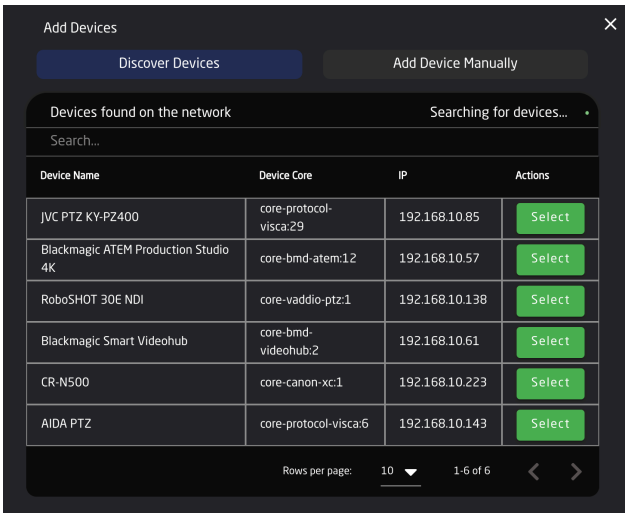
Device

The Devices section is where new devices are added to be connected with the SKAARHOJ panels. There are two ways a device can be added, Auto Discover and Manually.



Auto Discover

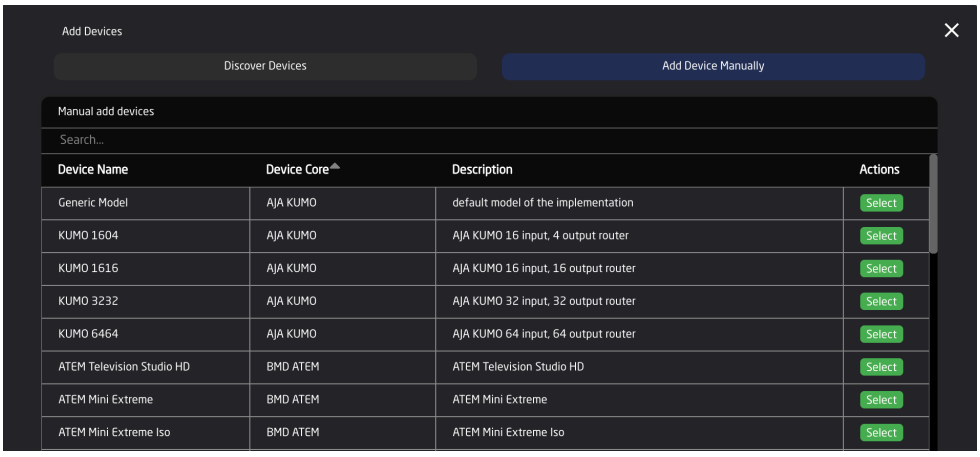
Searching on the network for devices will find many types by a combination of mDNS look-ups and other methods. However, not all devices can be discovered easily, but with those that can, it's a simple click of a button to add it to the Blue Pill device collection. Following an Auto Discover some device details may still be needed to establish connectivity. See Device Details section for set up.



Pressing SHIFT + the green Select button will allow for multiple panel selection.

Manually Added

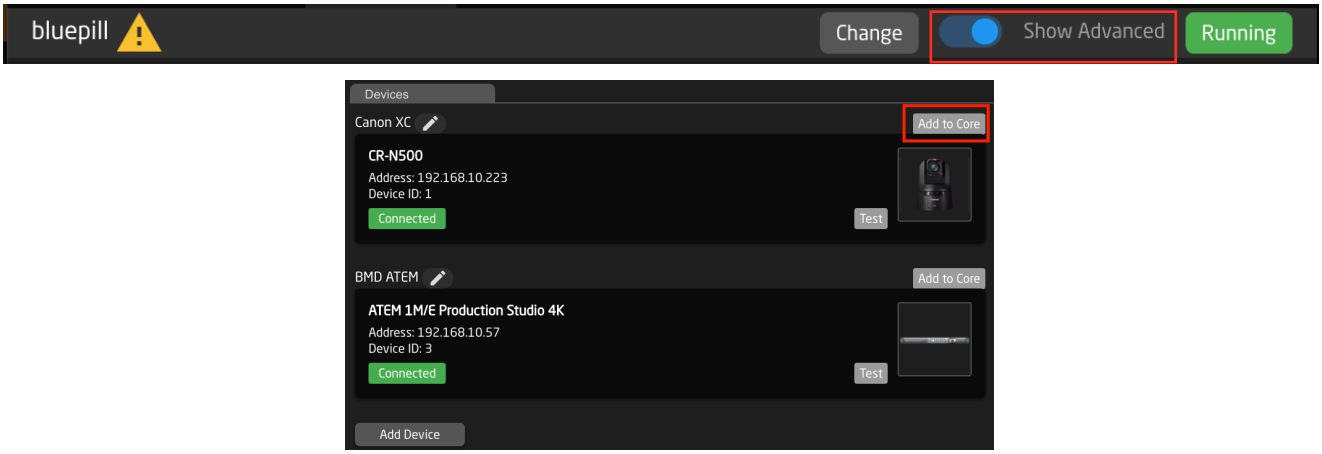
If a device must be added manually, it's easy to look it up in the list of supported models. Following a manual choice of device, the user will have to enter the IP address and other possible device detail information. See the Panel Details section for set up.



Pressing SHIFT + the green Add Device button will allow for multiple panel selection.

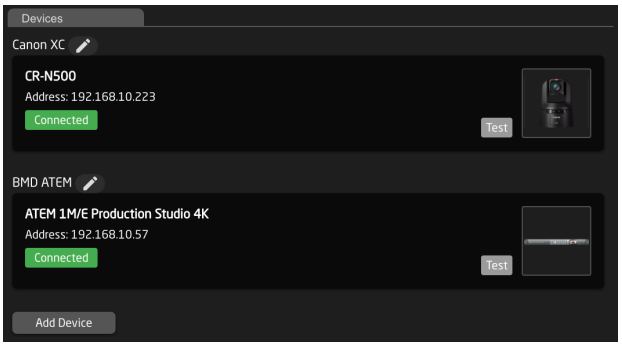
Adding Based on Core

When the Home screen is in Advanced mode, it become passible to add device pre-filtered based on the device core. Both methods of adding the device (discovered and manually) are still available.

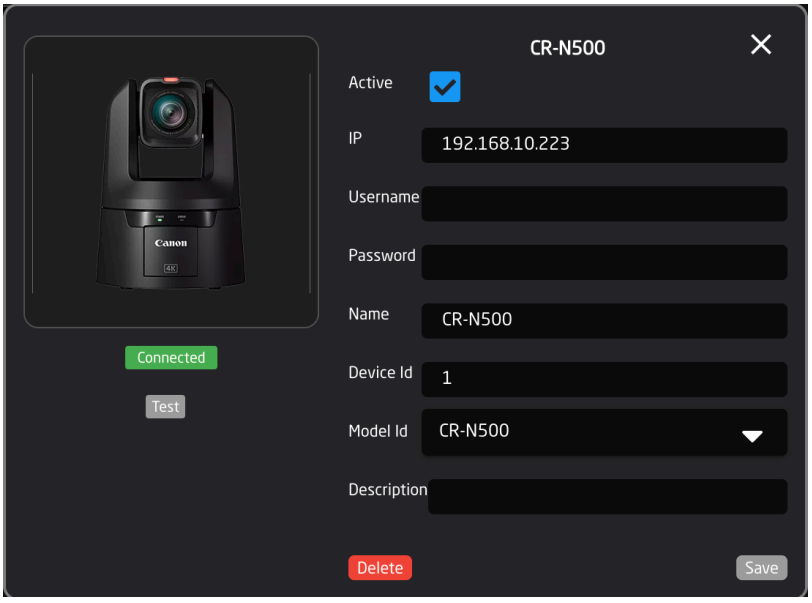


Device Details

Devices are presented in a nice overview with their connectivity status and IP address shown clearly.



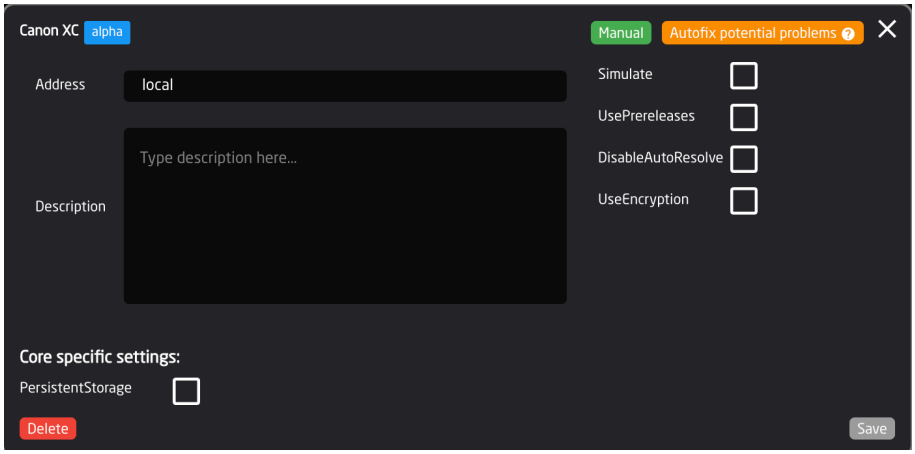
Clicking on a device's name in the Devices section will open up the details. Here the active status, IP address, Username and Password, Device Name, DeviceID, ModelID, and Description can be adjusted. (Details shown here are for a Canon CR-N500 PTZ camera, available detail options are device specific) . The device name and description are customizable. The device name will be the default name used by any camera selector for a display name. The status must be Active to control the device. The test button will trigger a test sequence to demonstrate connectivity, it is device specific and is not implemented across all devices.



Device Core Details

Devices will be shown broken down into different groups based on their associated device core. Device cores are auto loaded onto the blue pill when a new core is selected via a device.

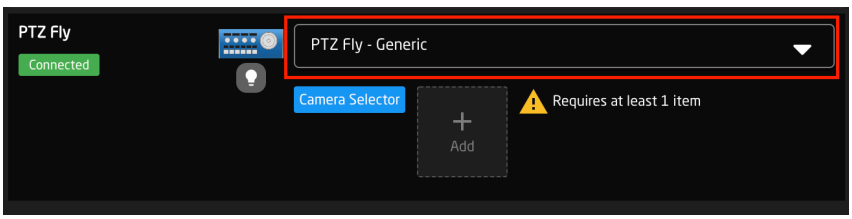
Clicking on the Device core name associated with the devices will open additional details.



Name	Description
Maturity Level	Development maturity level of the device core. See Packages section for more information on the different levels.
Manual	Opens device core manual.
Auto-fix Potential Problems	Re-syncs device core locally.
Address	The location of the device core. Local references the core on the individual Blue Pill. To connect to the core running off of a remote Blue Pill, use the IP address of the remote Blue Pill (must be accessible from local subnet).
Description	Custom description for personal reference.
Delete	Deletes the device core.
Simulate	Runs the device cores in simulation mode to see how it would work on the controller. The individual devices needs to be 'active' for it to simulate.
UsePrereleases	Uses the prerelease versions of the device core.
Disable AutoResolve	coming soon
Use Encryption	coming soon
Core specific settings	Settings specific to the individual device cores. Mouse over the name in the web interface for more details.

Mapping

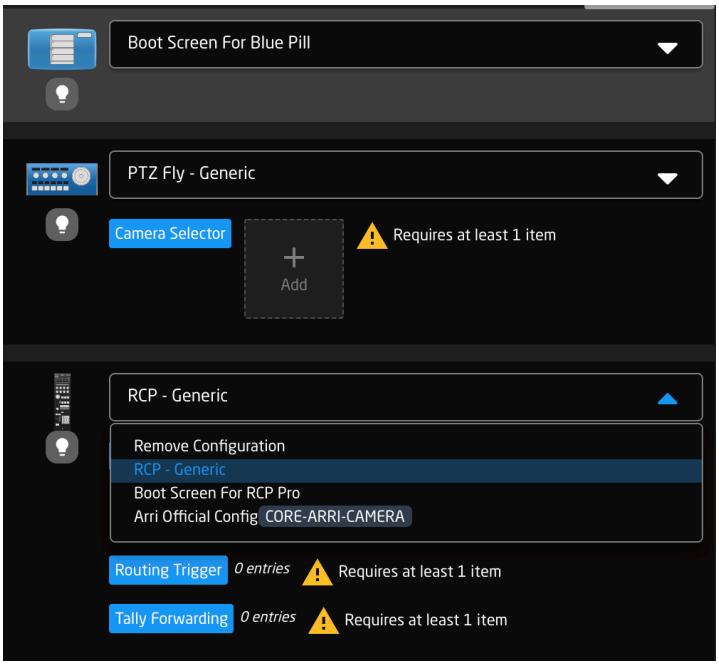
The Mapping section is built into the Panels section and is designed to provide a view into the current configuration and often it will refer to embedded standard configuration with associated *constant sets* for setting up cameras, router input/outputs or switcher inputs.



This is configuration the simple way using a pre-made default configurations.

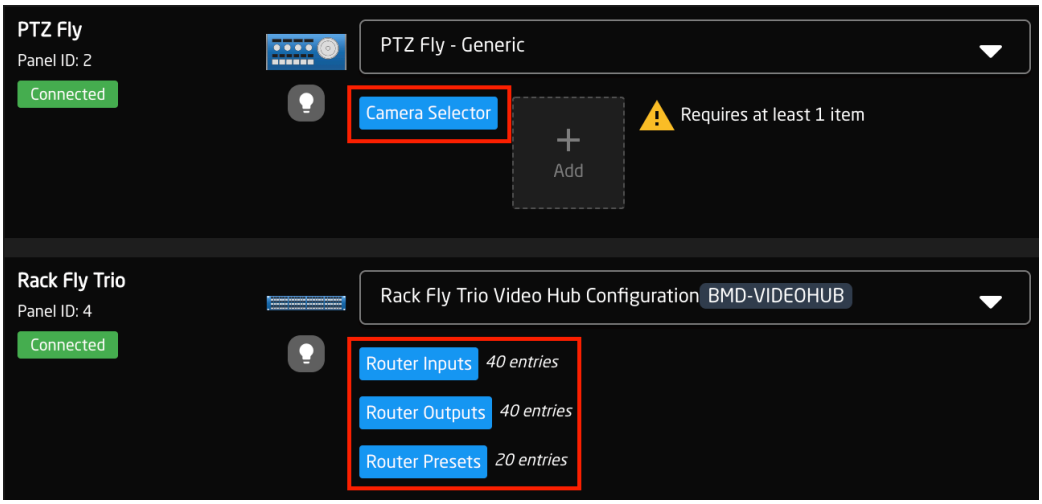
Configuration

The mapping page makes it easy to select a configuration for each controller. When the panel is added to the Blue Pill it automatically loads in the Mapping section with a drop down containing all the compatible configurations for that panel. In most instances the Generic configuration will be contain all the information needed to map the controller, though for some combinations of panels and devices there are specific configurations available.



Setup Tables (aka Constant Sets)

The final step is to fill in any setup tables, also known as *constant sets*. The constant sets are available based on the chosen configuration and should contain entries such as specific cameras for a PTZ controller, or the inputs for a video switcher etc. They are most common when working with PTZ cameras and routing panels, though may be available for additional device configurations.



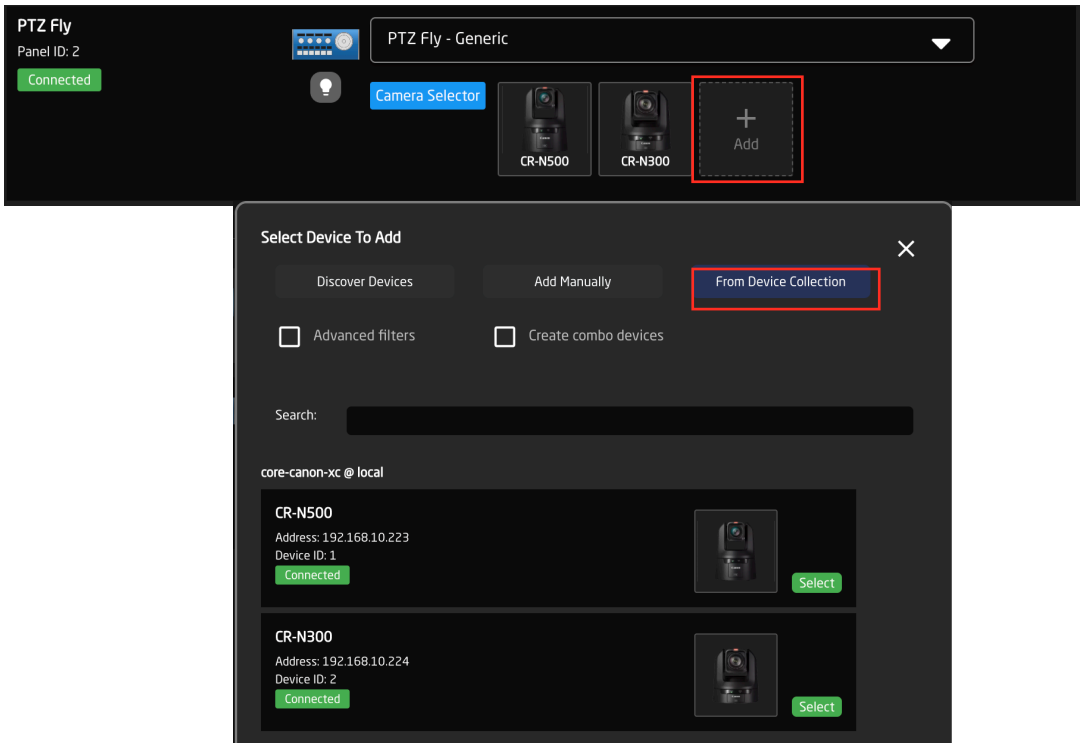
The setup tables will auto save and quickly appear on displays and enable the function.

An example of a setup table would be a camera selector and can be seen below. These can be different depending on the selected configuration. From here the order on the camera selector row of the panel will be set as well as the desired name on the displays.

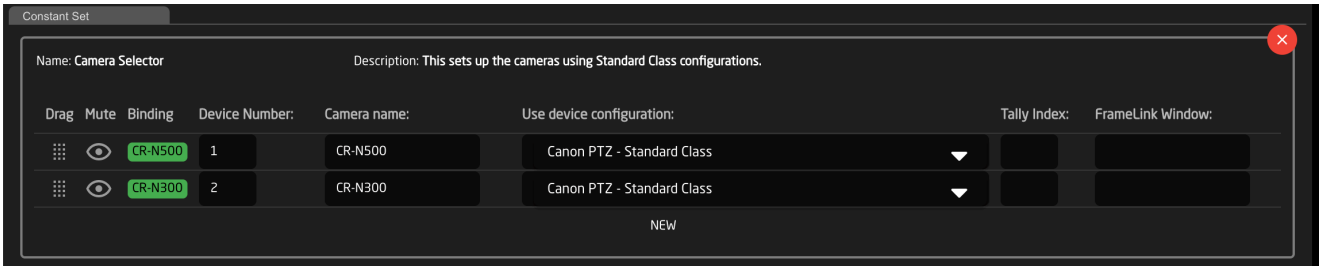
There are a couple of ways to work with the camera selector.

In the Panels Section, it is possible to click the ADD button next to the Camera Selector to select a device already in the device collection in the devices section or to add a new discovered or manually added device. Adding one with Discovered or Manually will require additional information added in the Devices section. Using a device from the current Device Collection will auto populate the Camera Selector, though it is still smart to click into the selector to double check the information is correct.

It is possible in the click and drag the camera icons in the Panels section to rearrange the camera selector row on your panel. The order that is seen in the home screen will be the order on the panel.

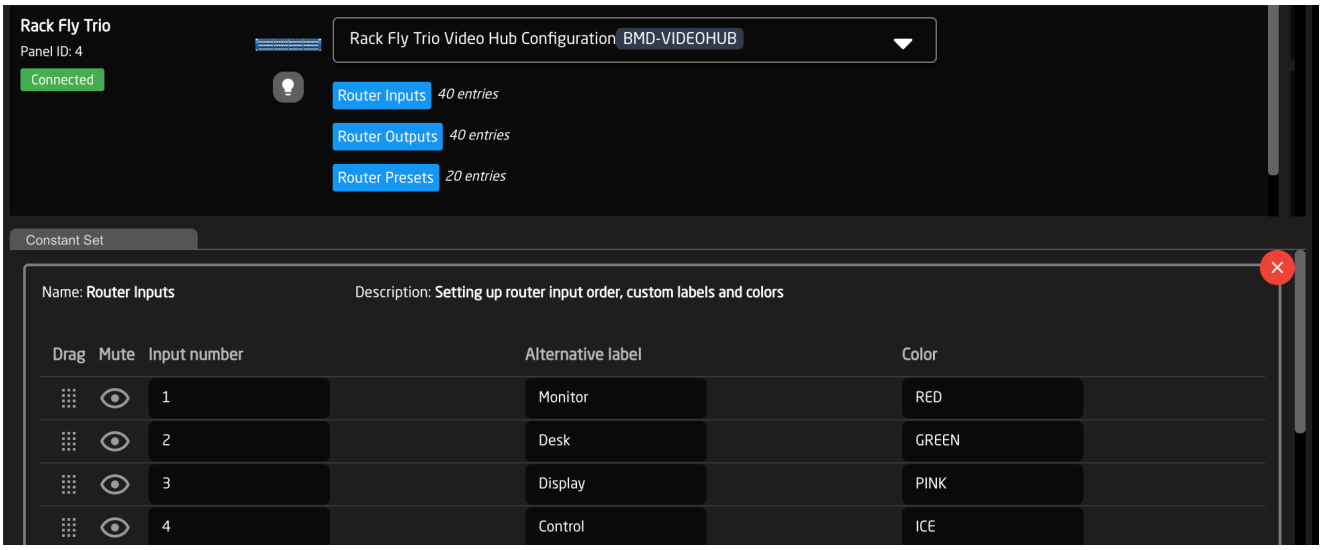


Clicking on the Camera Selector button will open the camera selector's Constant Set box. In there are a variety of options, some of which are not available in the Panels section. These options are device and configuration specific so may vary.



Column	Description
Drag	Allows for quick rearranging of camera order
Mute	Allows for removing access to a specific camera or to leave a blank spot on the panel
Binding	Allows for the selecting of a specific connected camera
Device Number	Ties the camera selector to the specific device. This is found in the Devices section. Each device will have a unique device number per device core. This box should auto-populate when a camera is selected in binding
Camera Name	Customizable name to appear on the displays. Character limit is determined by size of display and can vary.
Use Device Configuration	Selects the protocol based configuration associated with camera. Needed protocol can be seen in the Devices section, each device is grouped into their native protocols. **Double check the correct configuration is selected. Improper selection will effect camera control.**
Configuration for Iris/Master Black Channel	Selects the needed protocol for Iris/Master Black control. For cameras without a variable lens this will follow the same protocol as the device. For cameras with a variable lens, select the protocol for the attached lens. **Only available with some panel, mapping, and device combinations.**
Tally Index	Tally Index number set in advanced configuration tool. See Blue Pill/Reactor Manual for more information. This column does not need to be filled out for standard operation.
FrameLink Window	Sets the FrameLink Window value associated with the FrameLink device core for use with FrameLink compatible devices. See Blue Pill/Reactor Manual for more information. This column does not need to be filled out for standard operation.

An example of a router inputs selector can be seen below. These can be different depending on the selected configuration and device. From here the order on the inputs/outputs row of the panel will be set as well as the desired name on the displays and button color.

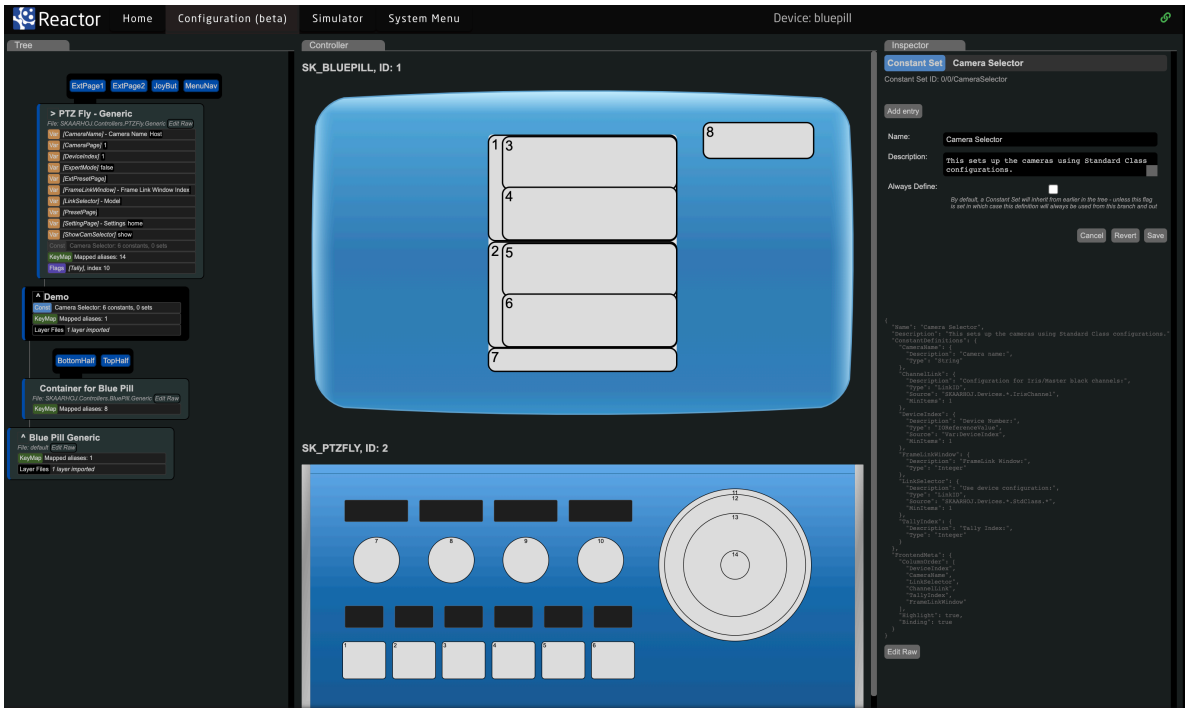


Column	Description
Drag	Allows for quick rearranging of input/output order on the panel.
Mute	Allows for removing access to a specific input/output or to leave a blank spot on the panel.
Output Number/ Input Number	Ties the selector to the specific input/output of the controlled device. This is determined by the individual router.
Alternative Label	Customizable name to appear on the displays. Character limit is determined by size of display and can vary.
Color	Sets the button feedback color. Color options are: OFF, WHITE, WARM, RED, ROSE, PINK, PURPLE, AMBER, YELLOW, DARKBLUE, BLUE, ICE, CYAN, SPRING, GREEN, MINT. The format for color selection is all large letters with no spaces between words.

Configuration- Experienced

For a more customized configuration either self constructed or to modify a default configuration used in the Mapping section it is necessary to move off of the Home page and into Configuration (beta).

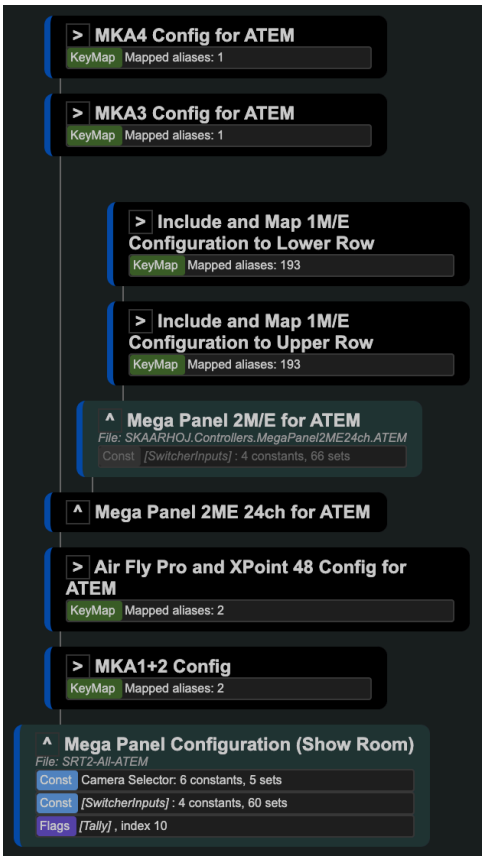
In the configuration menu is a view with a tree on the left side, the controller canvas in the middle and an inspector on the right side.



Pressing SHIFT when clicking into the Configuration page will allow the views to be reorganized depending on workflow. This is saved in the browser so will not follow between browsers (including incognito mode) or from computer to computer.

Please note, the Configuration page

The configuration of the SKAARHOJ Mega Panel would look something like this inside Reactors Configuration menu. There is a root layer at the bottom which is defined in its own configuration file, “SRT2-All-ATEM”. Inside there is a layer defining how the MKA1 and MKA2 accessory modules for the Mega Panel is configured. They are in fact set up to work like a big PTZ controller!



There is also a layer for Air Fly Pro and XPoint 48. This is only something we did for the SKAARHOJ showroom where we are basically making a third ME row on these two panels.

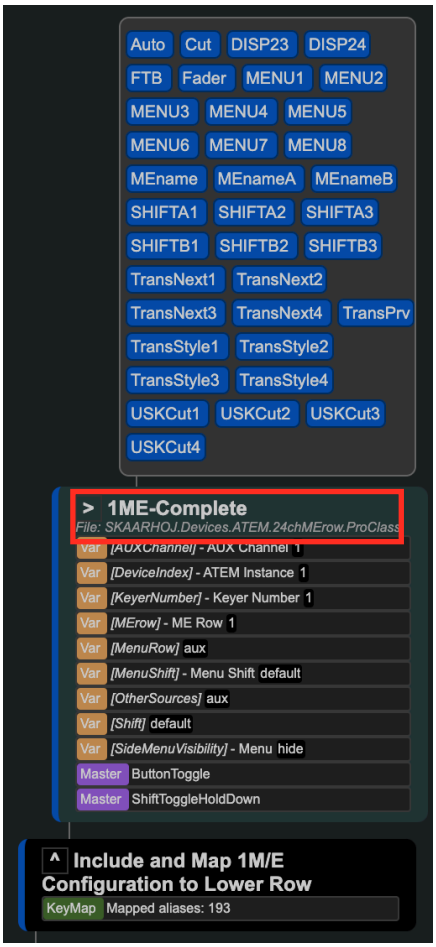
The Mega Panels ME rows (2x MK48 + MKT1 transition block) is organized as two similar layers inside a container layer called “Mega panel 2M/E for ATEM” (file “SKAARHOJ.Controllers.MegaPanel2ME24ch.ATEM”). They are in their own configuration files and a so called KeyMap is being used to map the configuration out on the right panel IDs.

At the top are configurations for the accessory modules MKA3 and MKA4.

The significance of the included layer files inside the structure is that it achieves a high degree of re-use of configurations throughout a system. The reused components will be driven by Constant Sets and local variables that make them behave differently depending on where they are included.

This is the unfolded layer of one of the 1ME configurations of the mega panel. Notice how the layer named "1ME-Complete" is drawn from the included configuration file "SKAARHOJ.Devices.ATEM.24chMErow.ProClass". The same file is of course included for the other ME row, and the difference is simply the KeyMap with the 193 mapped HWC Aliases to PanelID-HWCs.

On the root of the "1ME-Complete" configuration many behaviors are defined, such as Cut, Auto, Fader and obviously a lot of behaviors for navigation mixed up with specifics for setting upstream keyers and transition style - at least this is what we can guess from the alias names used. The designer has decided to let them all be on the root level of the layer, but they could have been organized inside of layers also. Usually, if a behavior is not changing based on a shift key or such, it can be safely placed on the root level and assumed to be the least complex way of organizing the behavior.



Here is the layers inside of "1ME-Complete".

The MKT1 Shift Layer seems to include alternative definitions to a number of buttons for the menu and transition style selector. It will be visible when the variable MenuShift is "1".

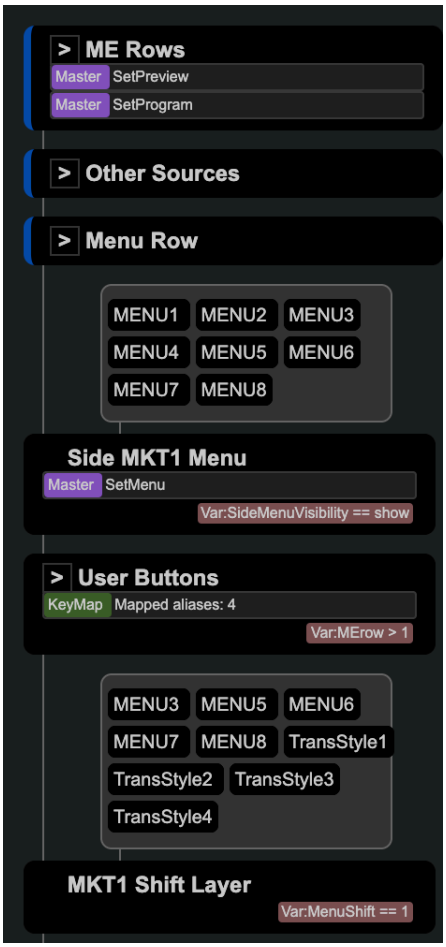
The User Buttons layer holds a reference to an included file on which the user can place his own behaviors and save it to the private folder of the configuration in order to avoid local changes to the main configuration file embedded from the system.

The layer Side MKT1 Menu would include navigation for a menu.

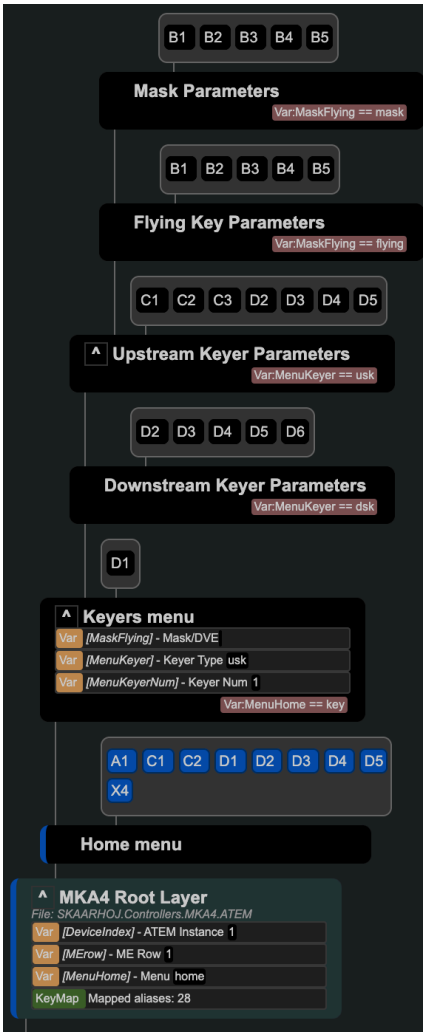
The Menu Row layer includes a lot of buttons for the top row on the MK48 panels

The ME Rows contain all the behaviors for the program row, preview row, displays and LED bars for color on the MK48 panels.

All of this is just examples of how the structure can be really deep but is well organized to make it fairly easy to work out large configurations.



On the mega panel there is a utility control panel, MKA4 with 42 buttons. Here is a configuration in progress with a nested menu structure where a number of variables are created and used to drive visibility of layers.



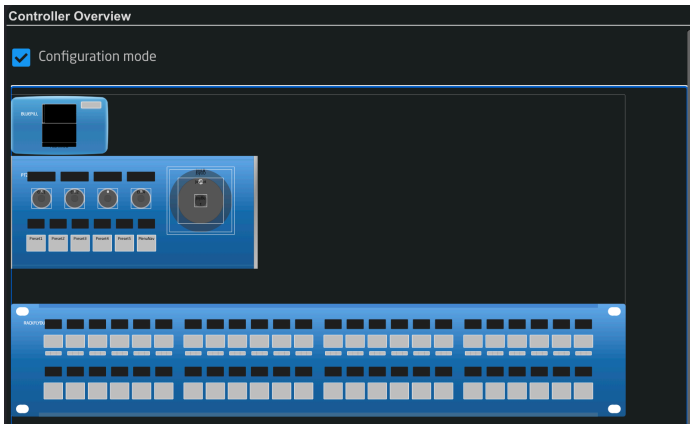
is currently in beta and is subject to change. We are constantly working to provide the most flexible controllers every available.

Tree

Here the configuration is organized in a tree structure that can accommodate many levels of nesting, and this essentially make the system virtually limitless, only governed by system resources

Controller

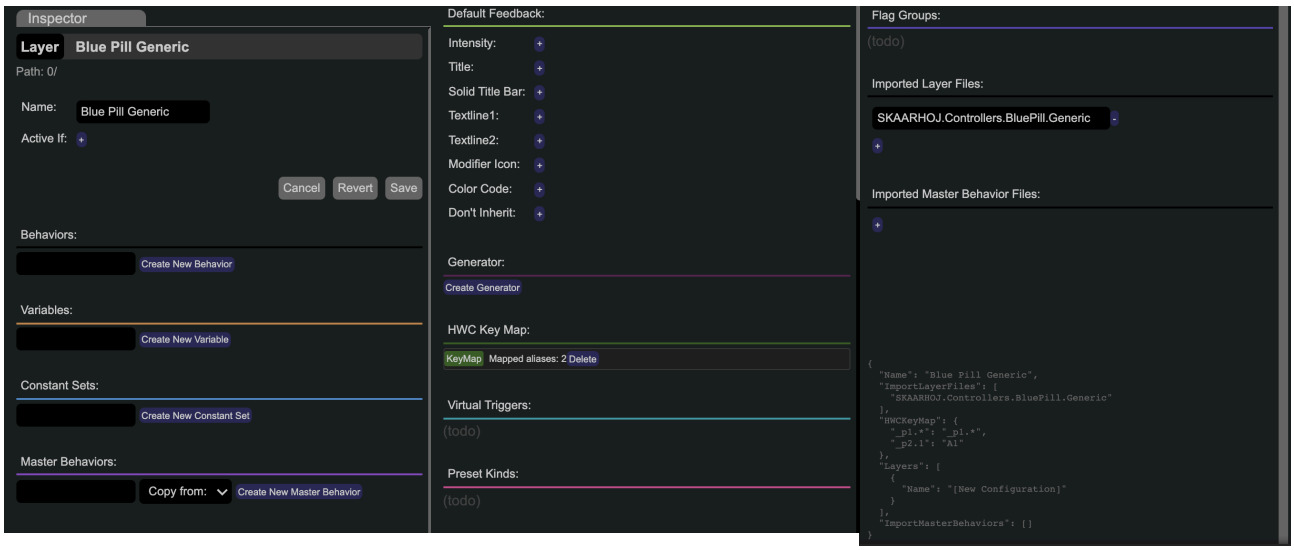
The controller section allows for a navigable visual interface to assist in moving around the tree. Right clicking on a hardware component opens the menu with five options. In order to create a Behavior, Page, Navigation, or Selector it is necessary to have a layer open (It is possible to start by selecting the root layer of the controller). Use “Highlight in all branches” to open up the tree and help navigate to the needed layer to start creating.



Unselecting Configuration mode will allow for the simulation of the the current configuration and any changes made.

Inspector

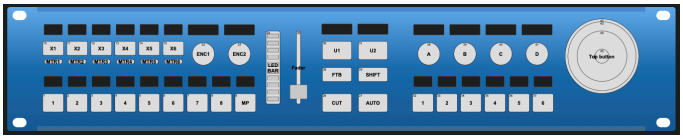
The inspector section allows for the editing of all aspects of the configuration. When a layer or individual button is opened the details are shown in the inspector. Features including but not limited to, Layer, Name, Behaviors, Variables, Constant Sets, and Feedback. The bottom section of the inspector tool also reveals the underlying code behind everything in raw json format.



Concepts of Reactor (BETA)

Hardware Components

In Reactor and UniSketch, *HardWare Components* - in short HWCs - refer to the buttons, knobs, faders and joysticks etc. on any given SKAARHOJ panel.

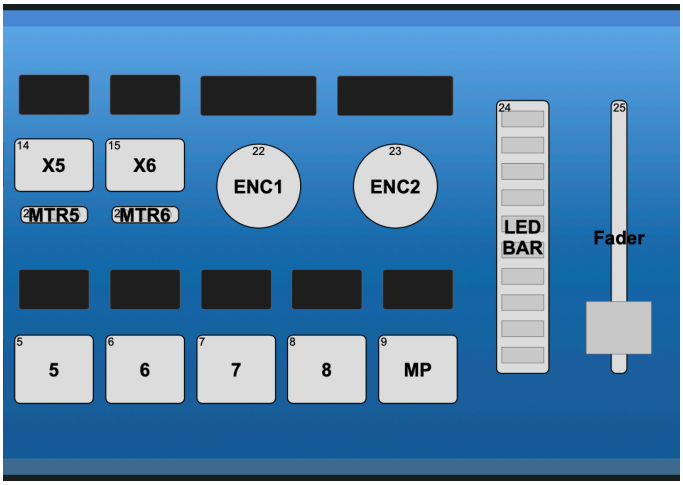


They generally produce four types of triggers:

- Binary triggers - button, press on an encoder, GPI inputs
- Pulsed triggers - encoders (knobs), roller wheels
- Absolute position triggers - faders of all sorts, binary dip switches too
- Intensity triggers - joysticks, shuttle wheels

At closer inspection of a controller, in the UI a small numbers is visible in the upper left corner - these are the hardware component IDs. We use these to refer to the components inside the configurations HWC Key Map.

Each panel on a canvas has an ID as well. Collectively, the panel ID and HWC ID is called something like PanelID-HWC and expressed as "_p1.23" in the HWC key maps. In this example, it would refer to panel ID 1 and HWC id 23 on that panel.



Most HWCs has RGB color feedback which consist of an intensity - Off, On, Dimmed - and a color tone. Both UniSketch and Reactor operators with 15 visually distinguishable colors, named such as RED, GREEN, BLUE, AMBER, PINK etc.

Displays on SKAARHOJ panels are usually graphical monochrome OLED displays with pixel resolutions from 64x32 and up. Grayscale and Color OLED displays are found on some models, and Reactor is perfect to make use of these displays.

Layers, Behaviors and Visibility

Reactor is build around an infinitely nested layer structure, starting with the root layer at the bottom. Look at it like a tree with branches and leaves.

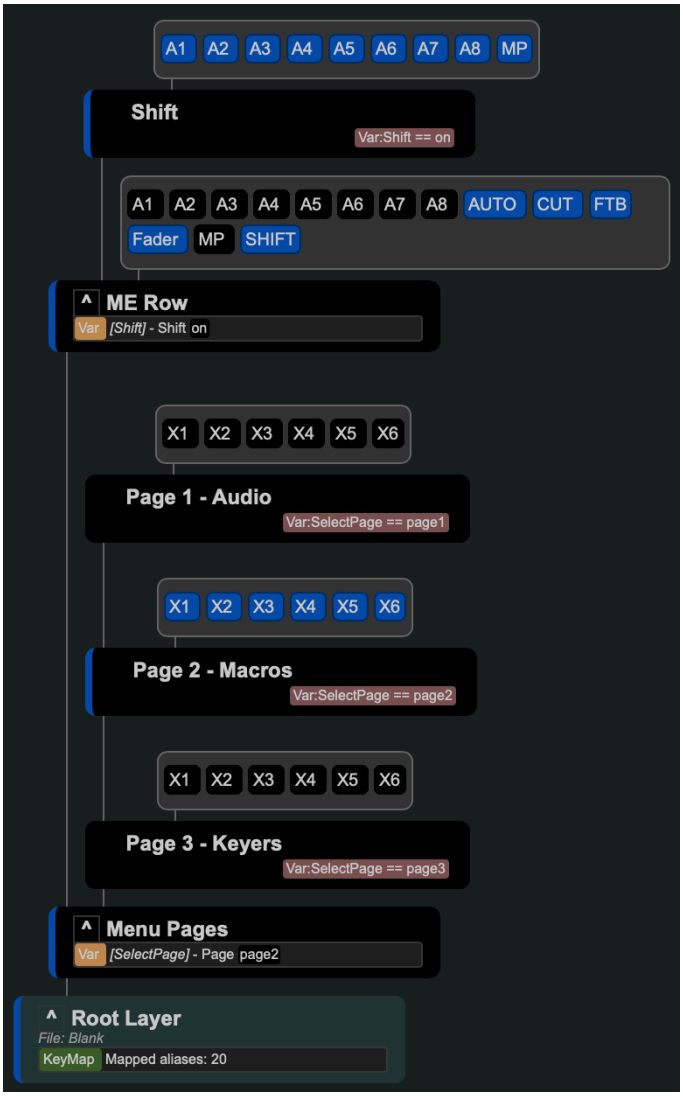
The leaves are behaviors for specific hardware components on the panel. If the same “leaf”, say the behavior X1, is found on two or more branches, the definition found on the top branch will be the active behavior - assuming the branch is also visible.

Branch visibility is determined by a condition called “Active If” and visible branches are marked with a blue line on its left side in the tree. The Active If condition can be driven by the value of a variable or device core parameter and supports nested logic expressions.

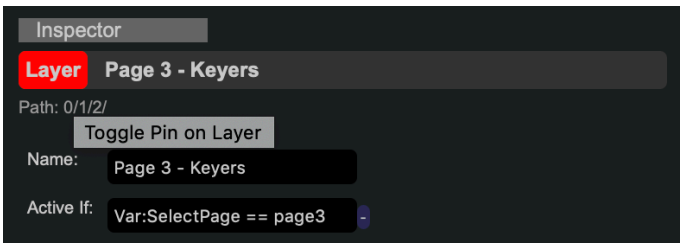
Active behaviors are colored blue in the tree as well - in this way it’s very easy to see which behaviors are currently active on the panel.

When trying to understand visibility, make sure to look on the tree from the top and down towards the root.

In the tree on the right it should be fairly easy to spot the blue behaviors (such as A1,A2,A3 etc) which are active and understand how behaviors with similar names on layers below are still black because they are occluded by the behaviors on the branch above.



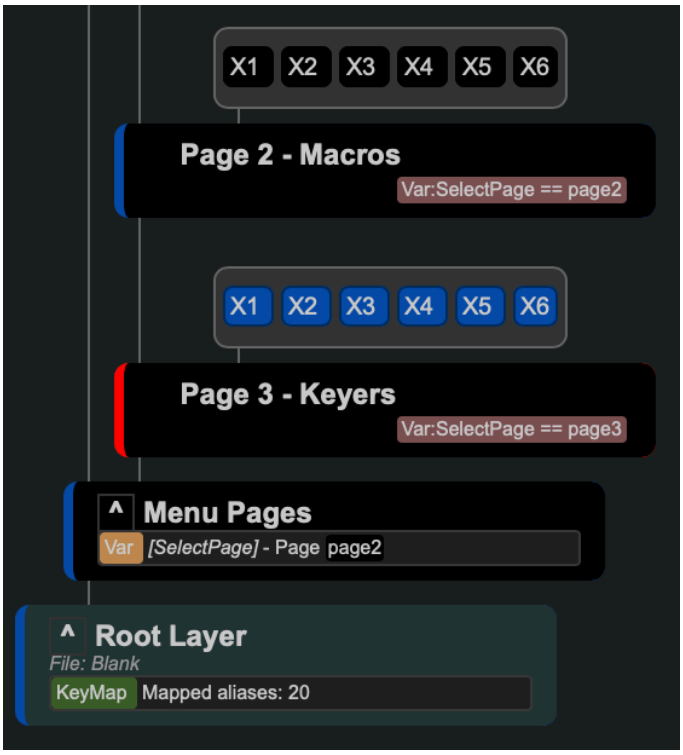
Layers can be pinned by clicking the Layer title in the Inspector. Pinning brings a layer to the top of the visibility stack and is designed to be a temporary shortcut for use during work in the Reactor UI.



A pinned layer will appear in the layer stack with a red indicator. The components on a pinned layer will usually be active (blue) as seen on the image on the right.

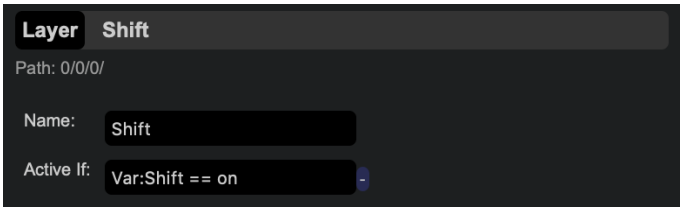
In this example, if the layer would not be pinned, it's likely that the behaviors X1-X6 on the layer just above ("Page 2 - Macros") would be active since that layer is visible. But pinning overrules this.

Multiple layers can be pinned at the same time. The last pinned layer is on top.



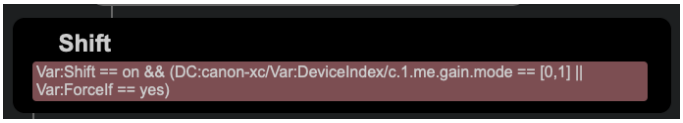
Layer visibility is driven by a condition, Active If and seen in the layer stack. Here the variable Shift needs to have the value "on" to make this layer visible.

The Active If condition can be edited in the layers Inspector



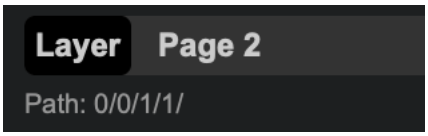
Active If is usually used with just a single variable, but can be complex and refer to device core parameters.

In this fictitious example, the variable Shift would have to be "on" and the gain mode of a selected camera would have to be either 0 or 1 or a variable from the system called "ForceIt" would have to be "yes" for the layer to show up.

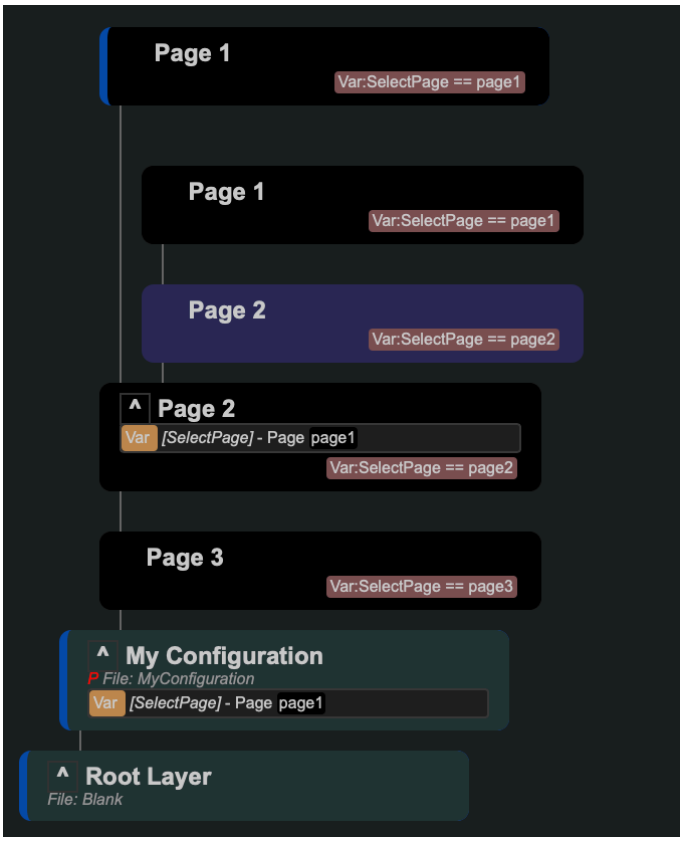


Var:Shift == on && (DC:canon-xc/Var:DeviceIndex/ c.1.me.gain.mode == [0,1] || Var:ForceIf == yes)

A layer in the tree is identified by its path which is a combination of the layer indices running back to the root. It looks like this:



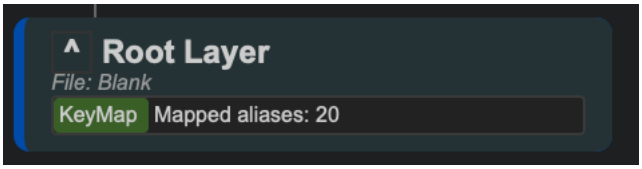
The select page, “Page 2” has this path, because it’s on the forth level in the tree. Numbering starts at zero and from the top among child layers on a layer.



KeyMap

The HWC Key Map is a feature potentially found in the layer tree at any position, but usually only on the root level

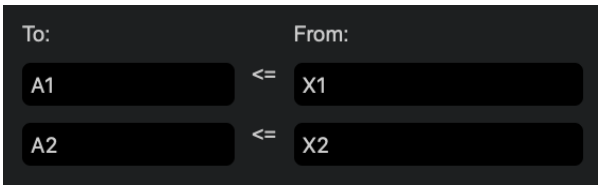
Overall it is not necessary to make changes to the KeyMap. Doing so can cause problems down the line. In most cases the KeyMap is autogenerated when making assignments on individual hardware components.



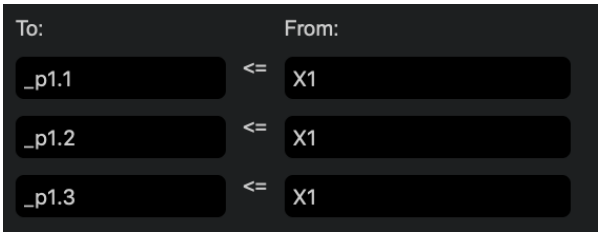
Inside is how the individual HWC Aliases used to describe the behaviors (eg. X1, X2 etc…) are mapped to PanelID-HWCs.

Full controller configurations as well as small snippets of specific functionality can easily be defined in a generic way when we are using HWC aliases in a human readable form (like A1, X1, Fader etc.) and then mapped onto the actual hardware components on any controller at the end.

It's also possible to map aliases to aliases:



A single behavior can be applied to multiple HWCs:



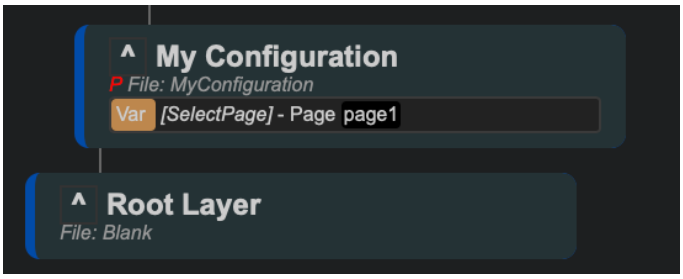
Full controller configurations can have their HWC aliases mapped to panel 1, and therefore it is possible to find panel to panel mappings in cases where there is a canvas with multiple controllers. In this case a configuration made for panel 1 is mapped entirely over to panel 5:

Import Layer Files

When creating new layers a box can be ticked to have them made “As Include File” and this will create a separate JSON file on the system with the configuration.

It’s seen clearly in the tree that a layer is the root of a configuration file since such layers are colored with army green background.

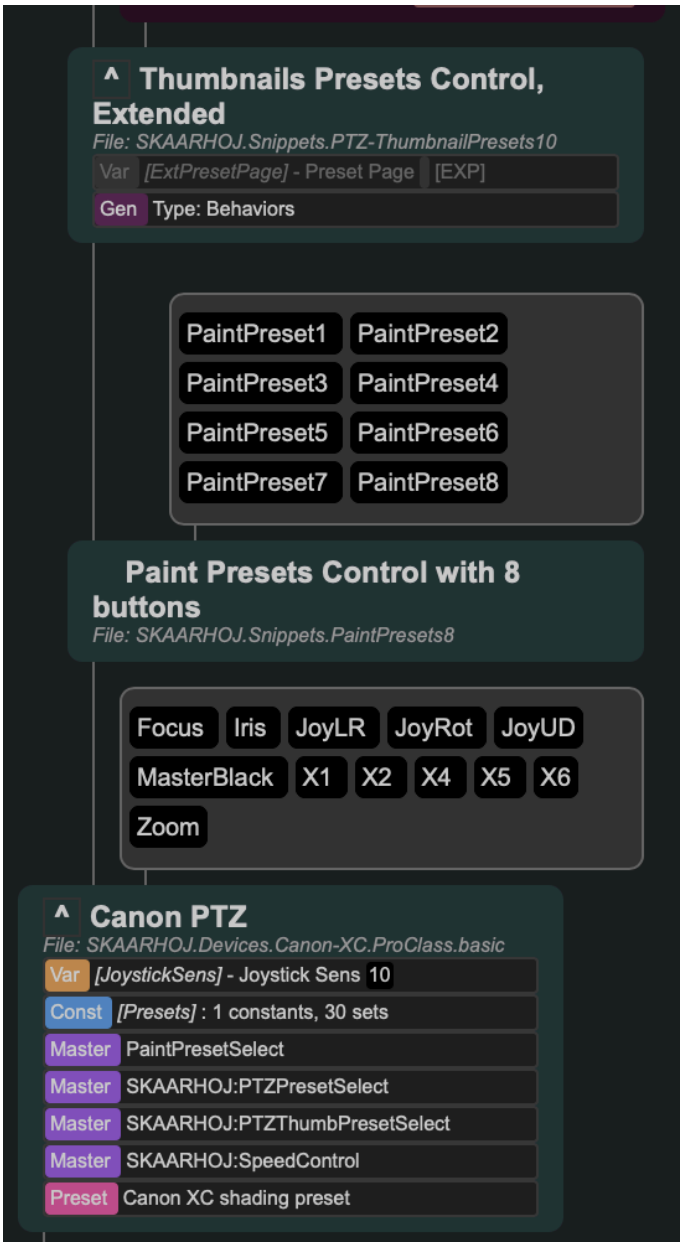
In this case, the little red “P” indicates that the file is a private include file only available to the main configuration we are working on (called “Blank” according to our Root Layer)



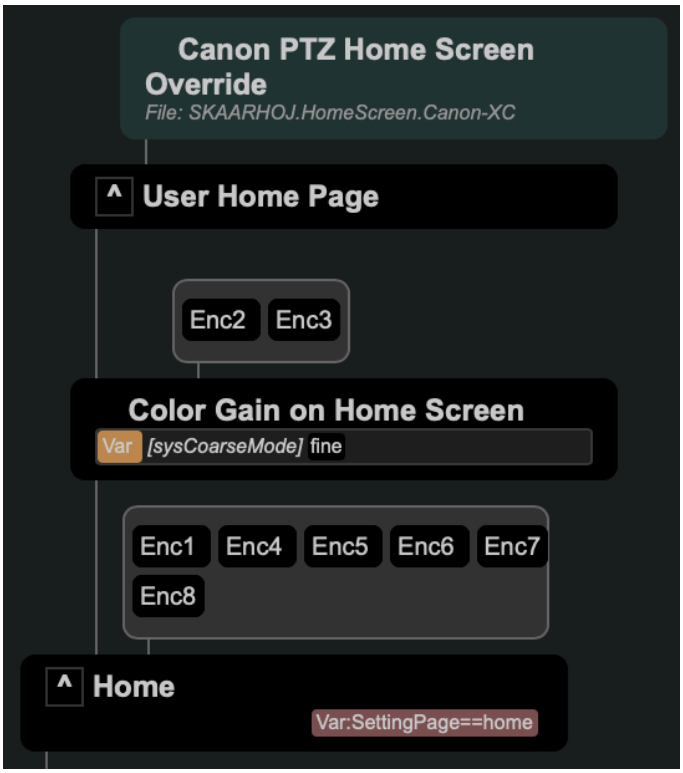
Included configurations is why starting from scratch with Reactor is not necessary.

In the case on the right is a file included called “SKAARHOJ.Controllers.PTZExtreme.Generic” which apparently is a universally useful configuration for a PTZ Extreme. If changes are made to this layer or any child layer is has, the changes will be saved to a private include file by the same name (and the little red “P” will appear) because Reactor won’t let an embedded configuration file be changed, but will allow for a saved copy.

Inside the PTZ Extreme configuration is visible how even that configuration is including other shared configurations, like one for Canon PTZ cameras, and inside that it will draw upon a generic set of snippet configurations for presets. The re-use is really high in this way.

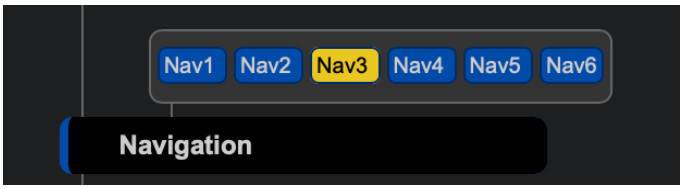


Many SKAARHOJ configurations are designed in a way that allow for the introduction of personalized functions on a home screen while *not* saving a private copy of the main configuration file in question. Searching the tree for an included file such as SKAARHOJ.HomeScreen.Canon-XC and adding behaviors to this, *that* file will get saved privately and used from there, but the general configuration for the whole PTZ controller and the Canon PTZ camera is still loaded from the embedded configuration. Thus insuring the benefits from any updates to that configuration that SKAARHOJ may publish in the future.



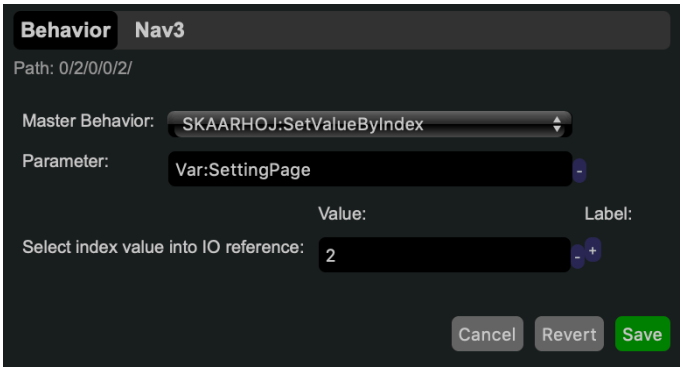
Behaviors: Master, Parameter, Constants

When clicking a behavior in the tree, it will appear in the inspector



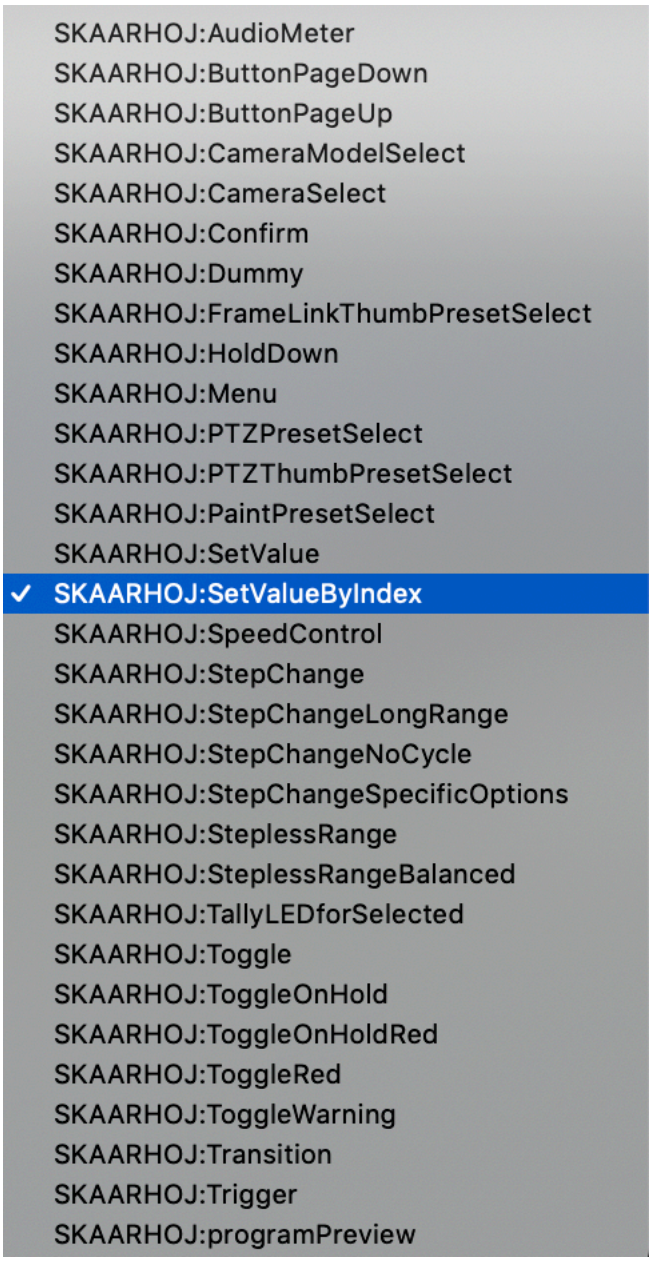
The two most important aspects of a behavior is instantly visible:

- The Master Behavior defines how the behavior works based on pointing to a master configuration provided by the system or from earlier in the tree.
- The Parameter is a reference to what is to be changed. In this case it's the variable SettingPage, but in most cases it will be references to a device core parameter, such as "DC:canon-xc/1/c.1.me.gain.mode" for gain mode on a Canon PTZ camera with Device ID 1.
- Depending on the selected master behavior, there may be one or more values to enter (technically called constants). In this case, "Select index value into IO reference" needs to be set to the index of the variable value to be set when this button is pressed.

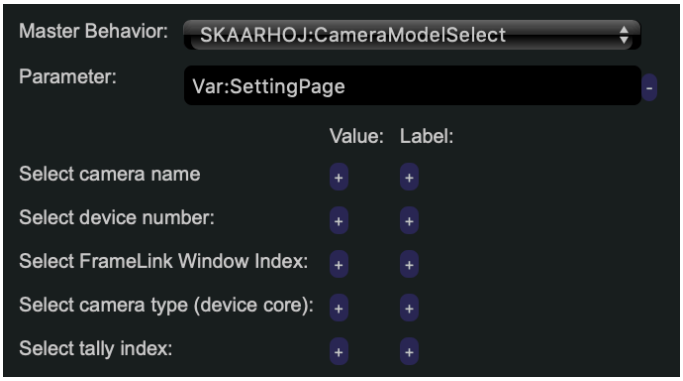


There is quite a nice list of master behaviors in Reactor by default. Lets point out most commonly used ones:

- SetValue will set a specific value on a parameter
- StepChange will cycle the value of a parameter on any kind of HWC, and it's ideal for small sets of values, like 2-10 options.
- StepChangeLongRange is better than StepChange if the value range is large, like from -50 to 50. The reason is that StepChangeLongRange has an additional trigger that toggles a fine/coarse mode so to more quickly scroll through the range
- Toggle is great for simple toggle situations with two values.
- HoldDown is an alternative to Toggle in case falling back to the default value when the button is released is wanted.
- ToggleRed is like Toggle (actually, it inherits from toggle) but applies a Red color when on.
- ToggleRedWarning is like ToggleRed but also blinks
- Transition is designed for analog components manipulating a large range like audio levels or switcher transitions.
- Trigger is used for trigger type parameters like "start record" or "one-push auto focus"
- SpeedControl is for mapping a joystick to pan/tilt/zoom parameters.
- Dummy is useful to just see something happen during panel configuration build up.



Some behaviors will have many more additional values to set, such as the CameraModelSelect behavior having up to five relevant values, including a label for the display, reference to the Device ID, which window on a FrameLink to pick for thumbnails, which camera type (that would link to a configuration file name) and which tally index (reference to a video switcher input number).



Behaviors: Feedback

A behavior consists of events that trigger changes on a device or in the system - and it contains feedback information that defines how a button light up and what content goes into the display.

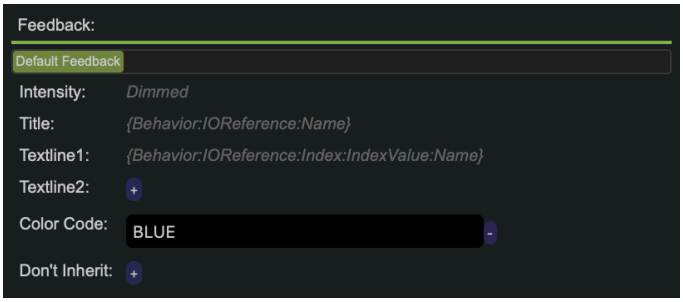
In Reactor the feedback has its own distinct property of a behavior. The example shown on the right shows how the color is set to BLUE while the gray values of the remaining features are inherited from the master behavior. In the specific case of Default Feedback, it actually also picks up any Default Feedback defined on layers on the path back to the root layer.

Intensity can have the value Off, Dimmed, On to determine how the button lights up. Default for most components is to be Dimmed when they has an active behavior

Title is the title bar in the display. In this example this value comes from {Behavior:IOReference:Name} which is whatever name the device core parameter or variable has.

Textline1 and Textline2 is the main text content in the display center. In the example here, Textline 1 is picked up from {Behavior:IOReference:Index:IndexValue:Name} which is the name of the option in question while Textline 2 is not defined.

Don't inherit is for blocking all inheritance and only apply custom values.

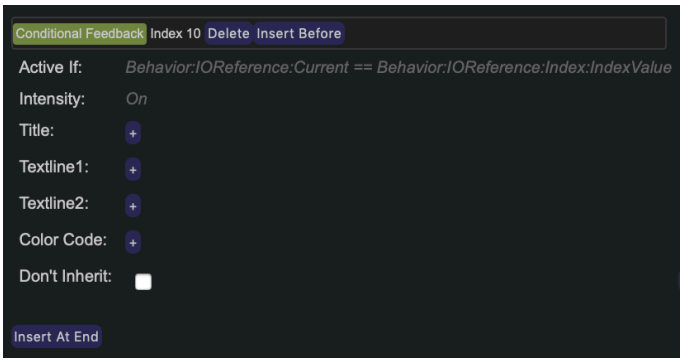


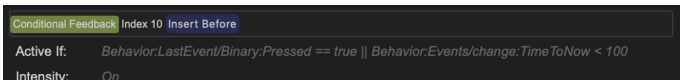
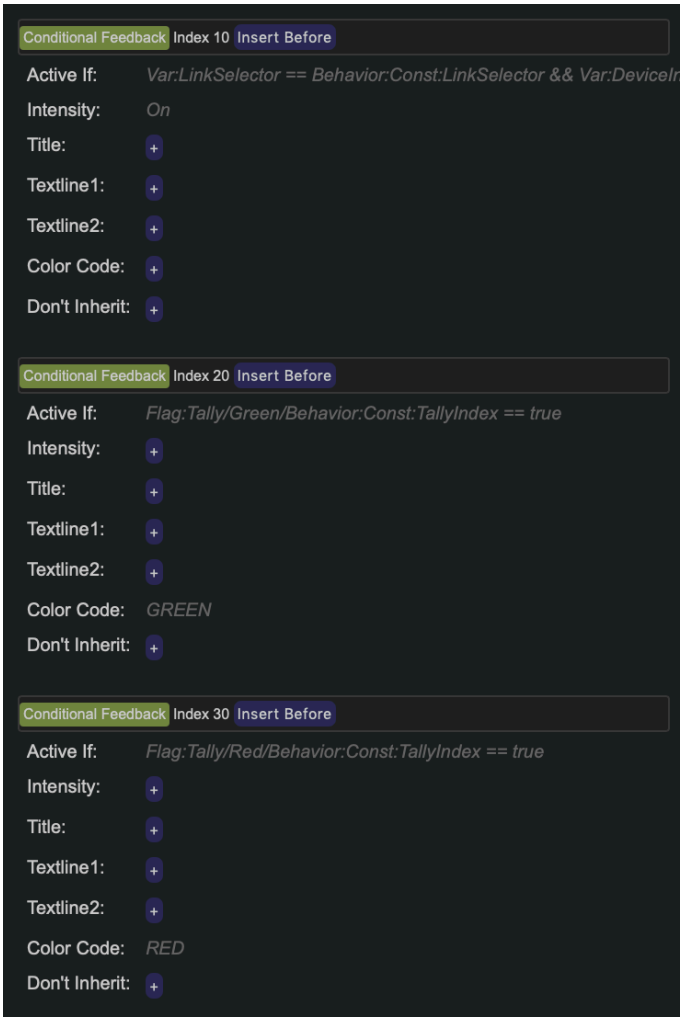
Most behaviors have conditional feedback too since we want their response to depend on system state.

Any number of conditional feedback items can be specified, but they must all have an Active If condition that determines if they are being applied or not.

In the example here, the conditional feedback will be active if the value of the current IO reference (system name for the parameter we have set) equals the value associated with pressing this button.

In case the condition is true, the only change is that the intensity changes to "On". Obviously, we could also change color, change display content etc. easily.



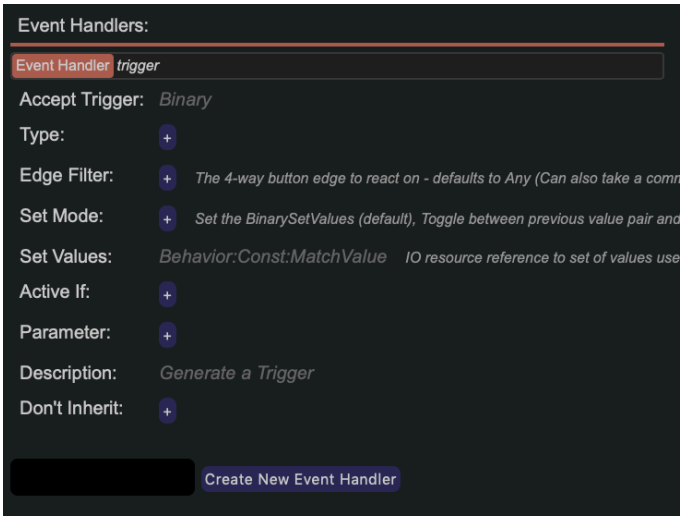
In another case like the StepChange master behavior, the conditional feedback is coded by this condition: “Light up, if the last event was a binary trigger which is still pressed or if the last event of change is younger than 100 ms”	 *Behavior:LastEvent/Binary:Pressed == true || Behavior:Events/change:TimeToNow < 100*
This example of conditional feedback is found on a camera selector. Notice in particular the last two conditional definitions at index 20 and index 30: They simply specify that the camera selector button shall be green or red depending on whether a given flag from the the Tally flag group is true or not. The flags in that flag group would be driven by a virtual trigger somewhere else connected to the video switcher system.	

Behaviors: Events

Event handlers are here to determine how to handle incoming events like a button press or turn of an encoder.

This example is SetValue which accepts a binary trigger and when received it will take the value from Behavior:Const:MatchValue (coming from the form field just under the Parameter field of the behavior).

Any amount of event handlers can be added. They need a name to represent them. In this example the name is “trigger”.



At closer inspection of the underlying code, it is possible to see that an event preprocessor is applied that will convert any non-binary triggers into binary triggers by some simple rules

- P2B converts pulsed triggers from encoders into ActDown binary events regardless of the direction turned.
- A2B will spawn a binary ActDown event every time a fader passes the midpoint regardless of direction.
- S2B will spawn a binary ActDown event every time a joystick passes some threshold value which is not clear from this configuration (may default to half way or full swing of the joystick).

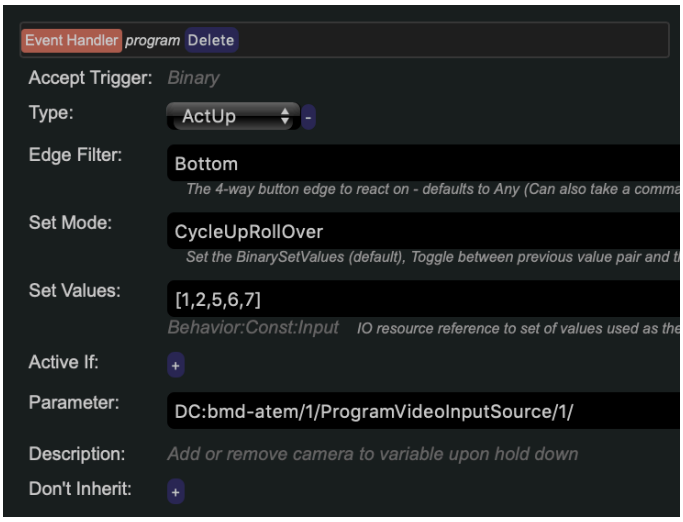
The fact that event preprocessing is not yet a part of the UI is a matter of priority and it may be implemented later, but until then it’s still possible for experts to use it, knowing the syntaxes. “Neo mode” has an editor with syntax highlight and schema information to help - even though the Chosen One wouldn’t need any of that (clue…).

```
"EventHandlers": {
    "trigger": {
      "Description": "Generate a Trigger",
      "AcceptTrigger": "Binary",
      "EventPreProc": {
        "P2B": {
          "InputPolarity": {
            "Default": {
              "OutputTrigger": "ActDown"
            }
          }
        },
        "A2B": {
          "InputMapping": {
            "Default": {
              "Threshold": 500,
              "OutputTriggerRising": "ActDown",
              "OutputTriggerFalling": "ActDown"
            }
          }
        },
        "S2B": {
          "InputPolarity": {
            "Negative": {
              "OutputTrigger": "ActDown"
            },
            "Positive": {
              "OutputTrigger": "ActDown"
            }
          }
       }
      },
      "BinarySetValues": {
        "Raw": "Behavior:Const:MatchValue"
      },
      "IOReference": {}
    }
  },
```

For Binary triggers, there are a number of options available:

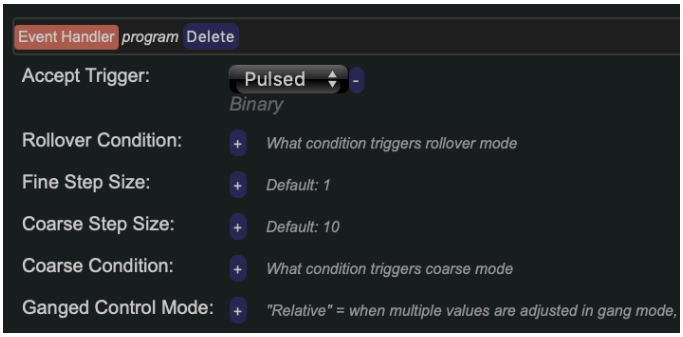
- Type: Act Up (release) or Act Down (press)
- Which edge of a four way button
- Modes, such as "Cycle Up and Roll Over" which would take the value through 1->2->5->6->7 and back to 1 in this case

An event handler is needed for every parameter needed to be changed. By default the parameter needing to be changed is the one set on the top level of the behavior (just under the master behavior selector). However, a local parameter can be inserted in a trigger if needed.



For Pulsed triggers, there is a different set of options available:

- Rollover condition determines whether an encoder shall start with the first value when it hits the end. Default is off.
- Fine and Coarse Step Size sets the increment/decrement numbers used when turning the encoder.
- Coarse condition is what makes the system choose fine or coarse steps. This is usually driven by a local variable SysCoarseFineMode defined on the behavior itself!
- Ganged Control Mode determines how the encoder manages if multiple devices/ parameters are being addressed. Either all values are adjusted relatively to their current value or the value of the first parameter is being applied to all others, which will get them in sync.

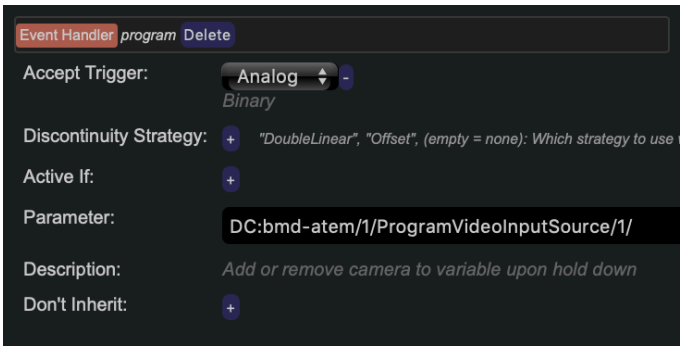


Analog triggers are straight forward but has some advanced modes to compensate in cases where they get out of sync with the real device value, like when a camera is changed on an RCP.

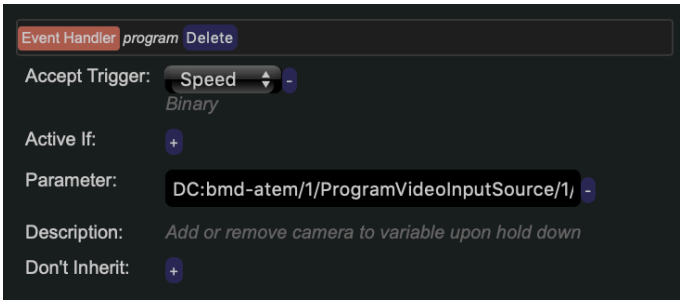
The Discontinuity Strategy offers two modes to deal with this:

- DoubleLinear: Let's say a device value is at 50% and so is the position of an associated fader. Then a change of device results in the device value being at 75%, but the fader is still stuck at the 50% mechanical position (we assume it's not a motorized fader). A double linear handling will then map the 50% travel movement of the fader towards the top to the 25% remaining of the value, while the 50% of the fader travel towards the bottom position would map to the 75% of the value. This secures a smooth continued adjustment of the value and is likely to be very intuitive in many cases. As the fader travels towards the end, this double mapping is continuously recalculated and will be completely evened out when it fully reaches the ends.
- Offset: In the same case as above the value will be adjusted with the original step size in both directions which means that in one direction it will never hit the end and in the other direction it will hit the end before it reaches the mechanical end position. It will eventually recalibrate its offset to get in full sync.

The Discontinuity Strategy has a special requirement to it: It has to have its own behavior for individual cameras, so parameters assigned with say Var:DeviceIndex will not work correctly unless Var:DeviceIndex is the same every time for the behavior. Luckily some features offered by Reactor makes it easy to achieve this. The generator called "Channel Pages" is able to create a configuration that fulfills this requirement.



Speed triggers (intensity triggers, like joysticks) have no special options so far.



Sometimes when digging into event handlers and their event preprocessor code a structure like the one here that is used to detect two types of cases is found:

- Send a trigger when a button is held for 1000 ms (1 second)
- Do *not* send a trigger if a button is held for more than 1000ms

The first event handler on the right ("set") uses an event preprocessor to achieve this. Notice that it converts binary to binary (B2B). Regardless of edge ("Default") it will block ActDown triggers and for ActUp it defines that the trigger shall only be fired if it happens within a time window of 1000ms.

The second event handler, "toggle", is what handles what will happen when held down for 1000ms. It's done using a repeater feature with delay. So basically, the first delay is set to 1000ms, then the repeater starts sending triggers back, but only a single one (max repetitions = 1) and to make sure the first ever trigger is not sent, it will block that one. E voila: A "press-and-hold" pattern. With proper modifications the same structure can be used to detect double and triple presses or send repeated triggers when a button is held down.

```
"EventHandlers": {
    "set": {
      "Description": "Clear and set a camera to
variable",
      "AcceptTrigger": "Binary",
      "EventPreProc": {
        "B2B": {
          "InputEdge": {
            "Default": {
              "PassThrough": false,
              "ActDown": {
                "OutputTrigger": "None"
              },
              "ActUp": {
                "TimeWindowToPrevTrigger": 1000,
                "OutputTrigger": "ActUp"
              }
            }
          }
        }
      },
      "BinaryType": "ActUp",
      "BinarySetValues": {
        "Raw": "Behavior:Const:MatchValue"
      }
   },
    "toggle": {
      "Description": "Add or remove camera to
variable upon hold down",
      "AcceptTrigger": "Binary",
      "EventPreProc": {
        "B2B": {
          "InputEdge": {
            "Default": {
              "PassThrough": false,
              "ActDown": {
                "Repeat": "Delayed",
                "RepeatDelay": 1000,
                "MaxRepetitions": 1,
                "BlockLeadingTriggers": 1,
                "OutputTrigger": "ActDown"
              },
              "ActUp": {
                "Repeat": "Stop"
              }
            }
          }
        }
      },
      "BinarySetMode": "ToggleAddRemove",
      "BinarySetValues": {
        "Raw": "Behavior:Const:MatchValue"
      },
      "IOReference": {}
    }
  },
```

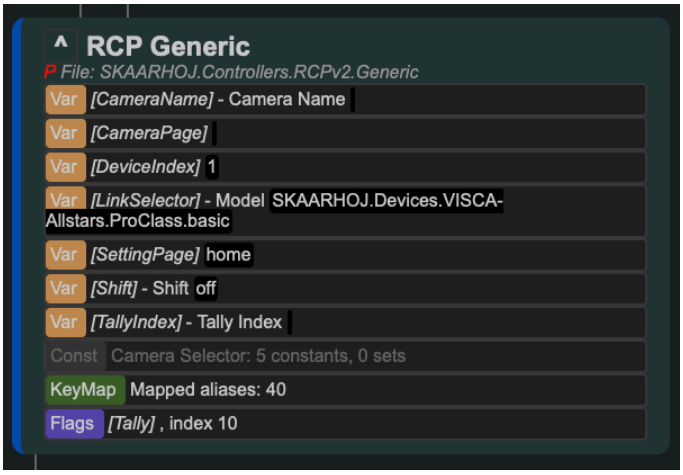
Variables

Variables are located anywhere in the tree and their scope is the branch they are in.

On the root layers of configurations can often be found many important variables applicable for that configuration.

Remember, the role of any variable is up to the designer of the configuration. Reactor itself imposes no constraints on this.

Typical conventions is the use of variable names such as “Shift” for control of Shift layers, “DeviceIndex” for device selection (like a camera selector), “SelectPage” or “SettingPage” for pages of adjustments. “LinkSelector” is often used to hold the file name of an external configuration being included by a generator.

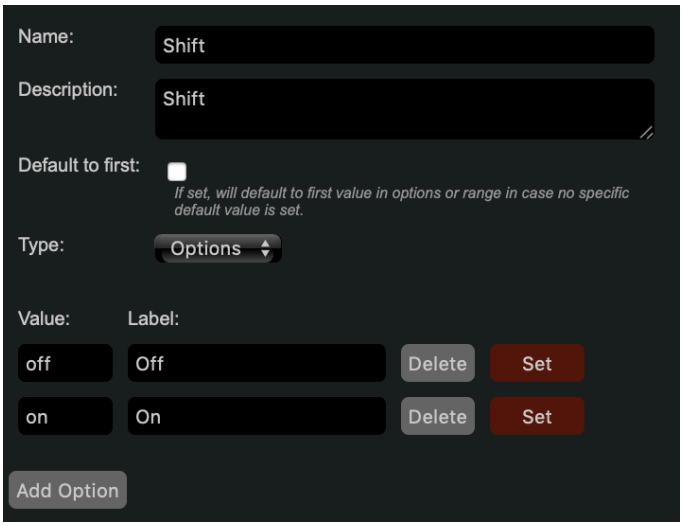


A variable will expose its path when viewed in the Inspector.

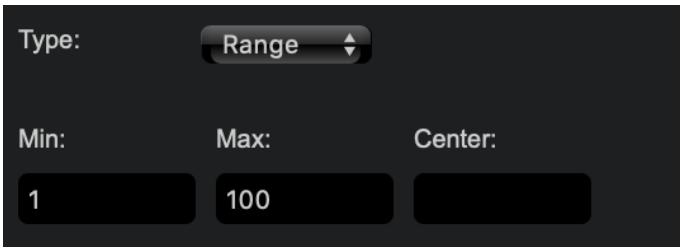
A typical Shift variable created using the “Create pages” function will look like this. It has two options, “off” and “on”.

Changing the labels of the options can be helpful if a display is associated with the button from which the Shift level is driven. The name of the variable is also shown in the title.

More options can be added by pressing “Add Option”.



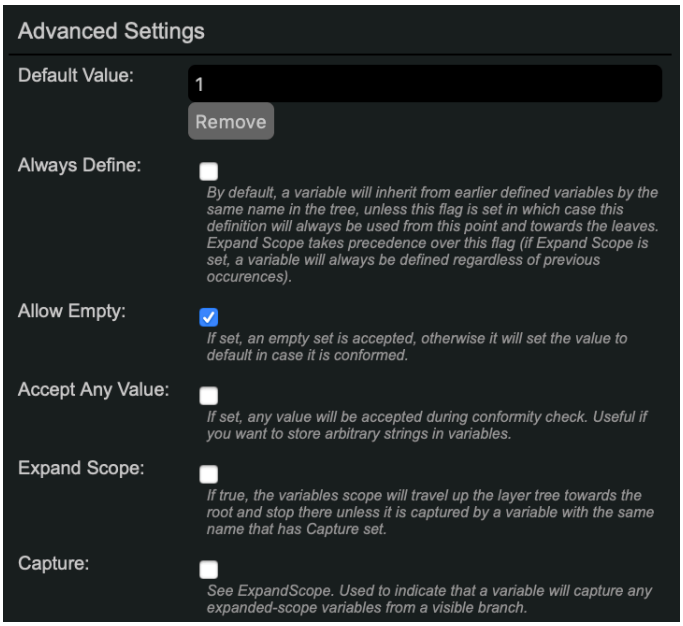
The other type of variable would be the Range type. Here an integer range is specified and possibly a center value that it would reset to with some behaviors.



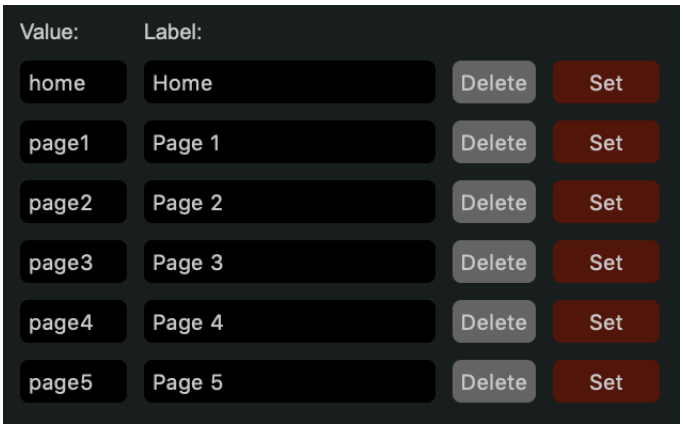
Variables have some advanced settings worth knowing about.

- **Always Define:** By default, a variable will inherit from earlier defined variables by the same name in the tree, unless this flag is set in which case this definition will always be used from this point and towards the leaves. It is often useful with imported configurations that any variable inside can be redefined on a higher level and driven from that scope, but usual system generated variables like "Shift" has Always Define set for "security reasons".
- **Allow Empty:** If set, an empty set is accepted, otherwise it will set the value to default in case it is conformed (force into its legal ranges). Allow Empty is used in camera selectors where multiple cameras can be selected to a variable - as well as a selection of no cameras should be possible too.
- **Accept Any Value:** If set, any value will be accepted during conformity check. Useful needing to store arbitrary strings in variables, such as a camera name on the RCP display.
- **Expand Scope:** If true, the variables scope will travel up the layer tree towards the root and stop there unless it is captured by a variable with the same name that has Capture set. This is the case where a variable affects the tree backwards from its branches. This feature is used in the sophisticated ways our PTZ configurations include model specific sub configurations and let them influence a navigational menu defined outside their scope.
- **Capture:** See ExpandScope. Used to indicate that a variable will capture any expanded-scope variables from a visible branch.

Furthermore its default value can be set specifically here as well.

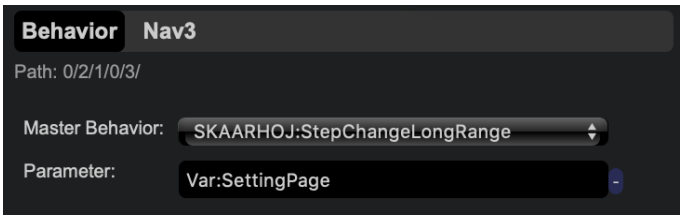


As a part of the frontend UI direct access is given to changing the value of variables via the red “Set” button. This is super useful as configurations are built up and custom conditional layers are desired.



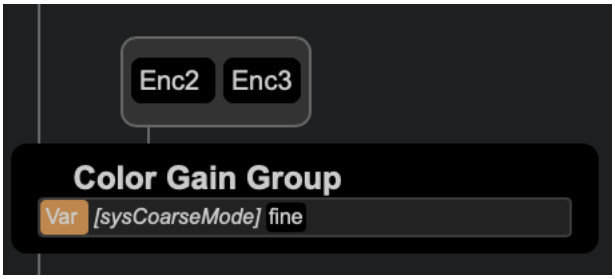
Variables are not only found on layers - they are in fact also a (lesser known) feature of behaviors. The best use case to pull out is how the master behavior StepChangeLongRange defines its own local variable “sysCoarseMode” to drive whether it adjusts values in fine or coarse steps.

There is currently (at deadline of this document) no UI to manage these variables, but this is how the code looks. Just a simple two-option variable.

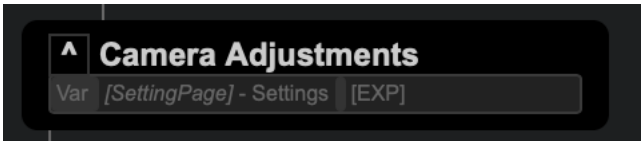



```
"Variables": {
    "sysCoarseMode": {
      "Description": "Toggles fine/coarse mode",
      "OptionList": [
        {
          "ValueID": "fine",
          "Name": "Fine"
        },
        {
          "ValueID": "coarse",
          "Name": "Coarse"
        }
      ],
      "Default": [
        "fine"
      ]
    }
  },
```

Variables on behaviors still are a part of the larger scheme of variable scope in layers and since the sysCoarseMode variable does not have its Always Define flag set, a variable can be created on the host layer of behaviors and just like that both behaviors will now refer to the same sysCoarseMode variable. This is incredibly useful for the grouping of related encoders to refer to the same fine/course mode state.

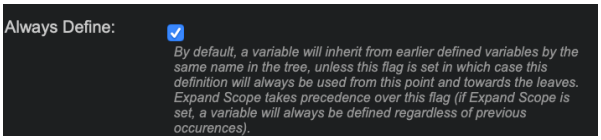


In cases where variables are defined but not active due to a previously defined variable, they will be grayed out.

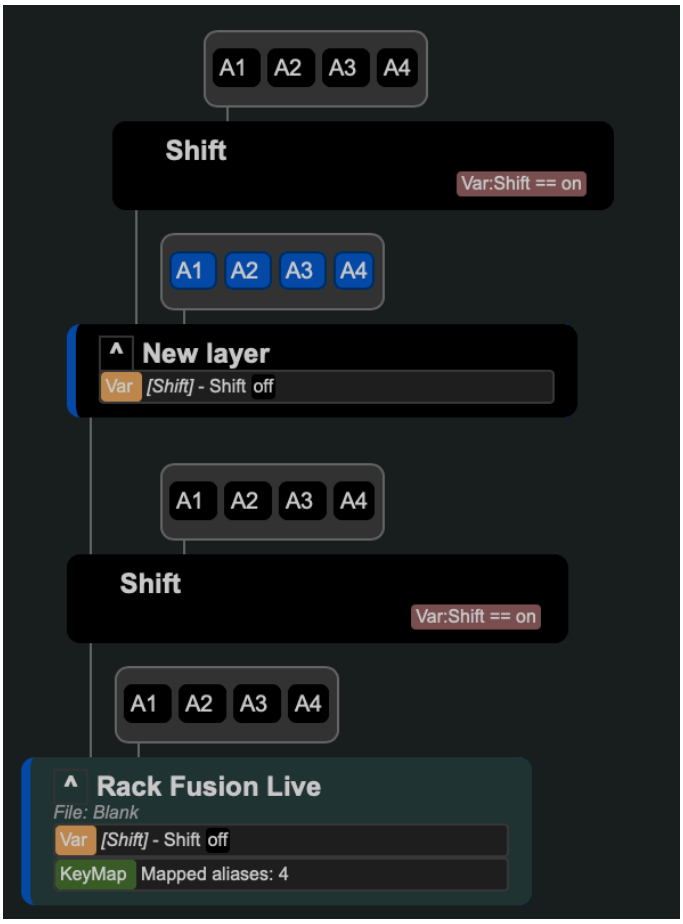


Here is a very simple example of how two Shift variables exist on the paths "0/" and "0/0/"

The one on "New layer" has the Always Define checkbox ticked. Therefore it will be active and used for anything inside the "New layer" layer.



If this checkbox is unticked, the variable would gray out and the Shift variable on path "0/ " would take over its role.



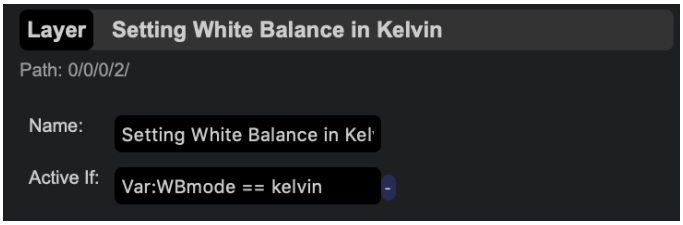
Parameters (IO References) and Conditions (Active If)

Parameters are everywhere in Reactor. Most notably when creating behaviors. Here they act as the target of a trigger from the panel as well as the source of the feedback. Exactly how this works is defined in the master behavior and any local overrides in the behavior being defined

In this case, the parameter is called "linkGain" and apparently in a RED Komodo or v-raptor camera.

Parameters can be variables as well

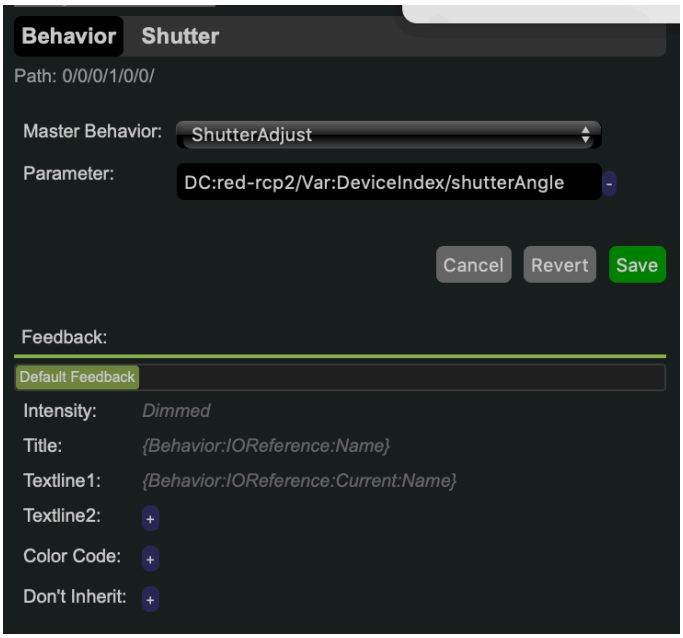
(There are more types, like Flags, Presets etc.)



Parameters are also found in the Feedback of a behavior, often as embedded values in the Title and Textline fields.

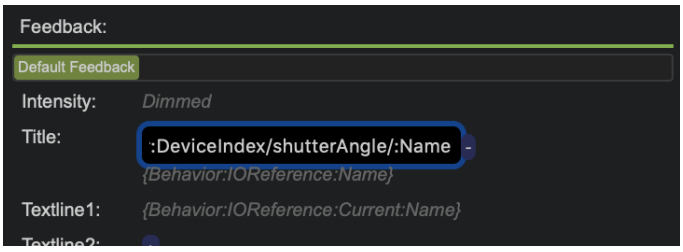
Setting the Parameter on the top level of a behavior can be very useful since the behavior can now refer to that parameter with “Behavoir: IOReference” everywhere else.

In this example that is done in the Title and Textline1 fields. Furthermore, see how the references used there are picking out the Name of the parameter for the title and the name of the current value for the Textline 1



Behavior:IOReference could be substituted with the full parameter itself.

Notice just one thing: Adding modifiers like “:Name” to the main body of a Device Core parameter should end with a slash.



Layer visibility can be driven by parameters. The Active If field of a layer is where the condition is inserted.

A condition is on the form [parameter] [operator] [parameter]. Operators can be == (equal to), != (not equal to), < (greater than), > (less than), <= (greater than or equal to), >= (less than or equal to).

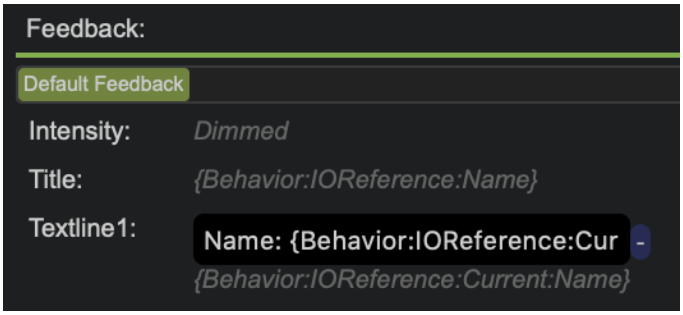
Balanced parenthesis can be used to group the logic of an Active If condition

A literal value like “1" is allowed, or a group like “[1]” or “[1,2]” can be made for more values. They will be evaluated as OR.

A Device Core parameter is prefixed with "DC:" and from that point it will be split by the slash. This allows for certain other parameters to be embedded inside, like variables (which never contain slashes). Each segment has a meaning: • First segment (red) is the device core reference • Second segment (blue) is the device index, often defined by the variable DeviceIndex (by convention) • Third segment (green) is the device core parameter name • Any other segments after this point will define *dimensions* of the parameter.	DC:**red-rcp2**/**Var:DeviceIndex**/**shutterMode** == **1**
In this case, two dimensions for the parameter name "audioGain" is defined in the parameter	DC:**red-rcp2**/**Var:DeviceIndex**/**audioGain**/2/1
Looking up audioGain in the red-rcp2 device core reference document reveals this description.	**Audio** **Audio Gain** *Set/Get Audio Gain (source 1 - internal; 2 -external) (Direction 1 - left; Direction 2 - right)* `audioGain`
In the same document is information on the parameters value range. It is a floating point value with normal feedback (so, it will tell us its value) and it has two dimensions: Two sources and two directions. The awesome thing is that Reactor gets all this information delivered from the device core over the TCP link and will automatically adapt to it: Knowing the value range in which to adjust the value, knowing how to display it with "dB" prefix in the display etc.	**Control:** Floating [-52.5:36.0]dB **Feedback:** Normal (Same) **Dimensions:** Source: 2 Direction: 2
Here is a very typical condition, evaluating a Shift variable.	Var:Shift == on
Actually, two variables can be compared, though, rarely done.	Var:Shift == Var:SomeOtherVariable
Conditional statements can involve logical operators like OR (||)	Var:DeviceIndex == 1 || Var:DeviceIndex == 2
However, it can be written shorter in this way. A group of literal values encapsulated in square brackets will also be evaluated as OR.	Var:DeviceIndex == [1,2]
To compare a value to a blank string make empty square brackets.	Var:Shift == []

Parameters used in the context of behaviors have some special features. We have already seen Behavior:IOReference as a convenient way to refer to the top level defined parameter. But a number of aspects of the included triggers can also be referred to.	Behavior:LastEvent/Binary:Pressed == true || Behavior:Events/change:TimeToNow < 100

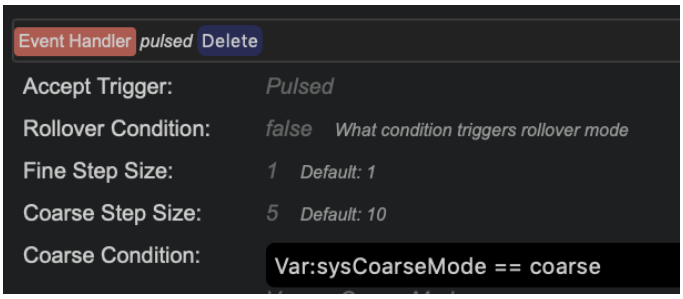
Some examples (copied out from internal documentation):
 	
`Behavior:LastEvent:Type == [Binary/Pulsed/Analog/Speed]`
 	
`Behavior:LastEvent:TimeToNow < 500 (milliseconds)`
 	
`Behavior:LastEvent/Binary:Pressed == [true/false]`
 	
`Behavior:LastEvent/Binary:Edge != NoEdge [NoEdge, Top, Left, Bottom, Right, Encoder]`
 	
`Behavior:Events/[the eventHandlerKey]:TimeToNow < 500 (milliseconds) Last accepted event.`
 	
`Behavior:IOreference[:modifiers added to IO reference]`
 	
`Behavior:Const:[constant]`
 	
`Behavior:Path`
 	
`Behavior:Name`
 	
`Behavior:Id`	

When parameters are used in text strings for feedback they can be mixed with plain text strings. It may be useful to extract the names of parameters and labels of values with modifiers too (here, the ":Name" modifier)	{Behavior:IOReference:Current:Name} 

Some examples of general modifiers for Parameters (IO References):

```
Name - Returns Name of the reference (like a parameter or variable name)

Mutable - Returns “true” if available for change

Assumed - Returns "true" if a parameter is assumed in the core

Exists - Returns “true” if exists

Default - Returns default values of IO reference

Default:Name - Returns names of default values of IO reference

FineSteps - For a DC this returns the recommended fine steps, else returns 1

CoarseSteps - For a DC this returns the recommended fine coarse, else returns 10

Current - Returns values (same as no modifier)

Current:Name - Returns names of values

Current:Normalized - Returns value normalized to value range 0-1000

Current:Index - Returns index of current value from option list, starting with 1

Current:BufferTimeToNow - Returns milliseconds from buffered value time out to current time.

Label    - for constants, returns the label a given constant may have

All - Returns all option values for a parameter

Index:[int/HWC behavior constant string ref] - Returns option value with index [int] or if its a
string, the constant of the HWC Behavior (works only for HWC behaviors then) (starting with zero)

Index:[int/HWC behavior constant string ref]:Name - Returns name of option value with index [int]
(starting with zero)

Index:[int/HWC behavior constant string ref]:Exists - Returns true if the index exists

Index:Last - Returns last option value

Confirm:[int] - If added to a setting, stepping or mapping values, it will change the value only
locally and waiting to be confirmed for [int] milliseconds after which it will fall back.
Confirmation happens if a trigger (valueless value setting) happens.

Wait:[int] - like Confirm, but the change is automatically accepted on the expiry of the time.
```

Conditions are driving many other things than just layer visibilities. Here is an example from the StepChangeLongRange master behavior where the condition to set the coarse mode is driven by the variable sysCoarseMode.

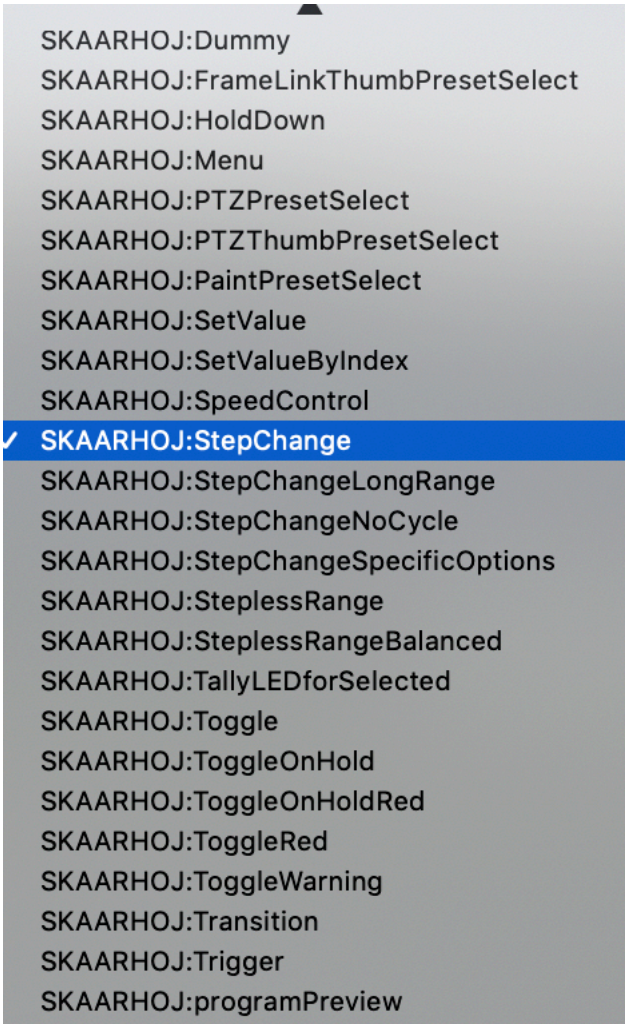


Master Behaviors

Master Behaviors are essential to facilitate quick and consistent configuration of a panel. In the far majority of cases, a user doesn't have to care to anything else then picking the master behavior and the parameter they want to adjust - everything else will adapt perfectly.

SKAARHOJ provides master behaviors embedded in the system.

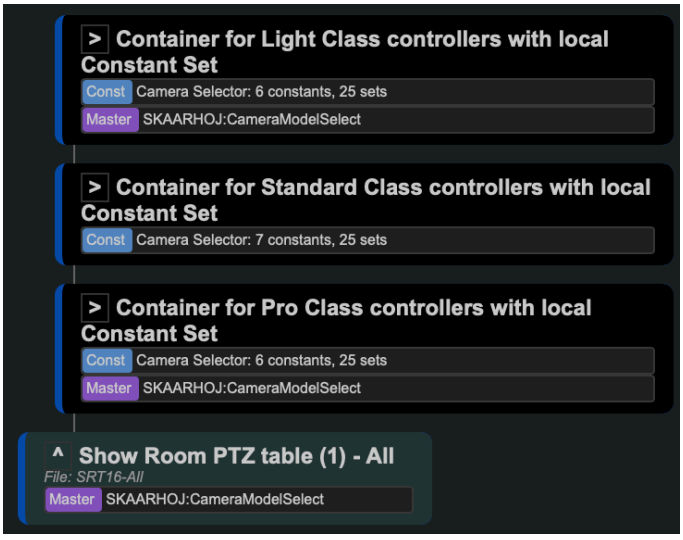
(It's actually possible to embed other libraries of behaviors, but this is a pending feature waiting to be unlocked by a future online repository.)



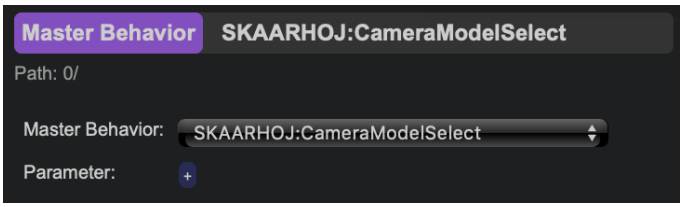
Master behaviors can be overridden anywhere in the tree. Here is the SKAARHOJ:CameraModelSelect being redefined on the very root level of the configuration.

The layers named "Container for Pro Class…" and "Container for Light Class"… also seems to redefine this master behavior.

The awesome thing about redefining a master behavior with its own name is that any included configuration in the branches will have to use this redefined master behavior. This provides a way to customize what imported configurations do. We use it heavily for preset selectors for example.

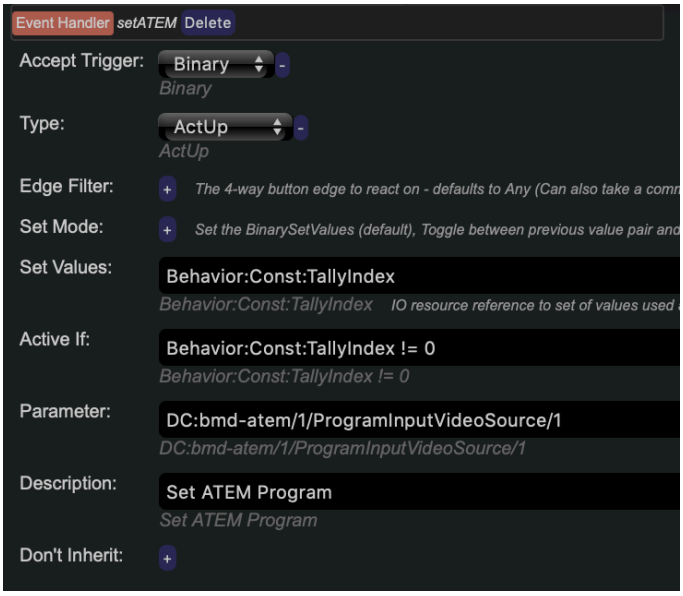


A closer look at the root level redefinition reveals that we are in fact basing this master behavior on itself but from a previous level (the one embedded in the system)



Scrolling to the event handler section we will see a lot of additional event handlers are added which add handling for local switching on ATEM switchers and video routers.

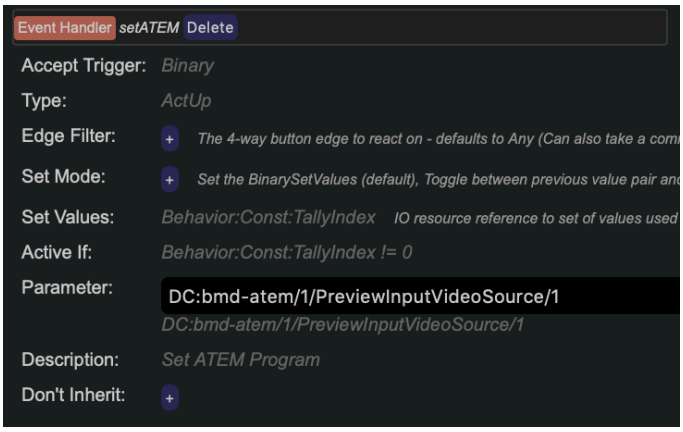
(This config is from our showroom and the purpose is to switch sources on a few ATEM switchers and a router to make the selected camera appear on the screen.)



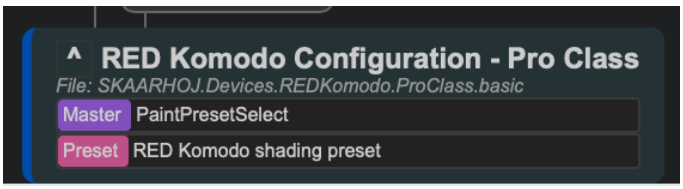
The redefinition that happens on one of the child layers mentioned simply change just a single parameter that turns out to be local just to them. In this case, changing the specific parameter that the trigger affects.

On the code level this is actually super simple:

```
{
  "ParentID": "SKAARHOJ:CameraModelSelect",
 "EventHandlers": {
    "setATEM": {
     "IOReference": {
        "Raw": "DC:bmd-atem/1/
PreviewInputVideoSource/1"
      }
    }
  }
}
```

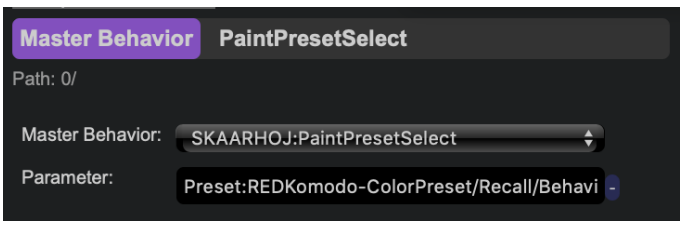


Instead of redefining a master behavior with its own name, a custom name can be used. Doing so leaves the original master behavior untouched.



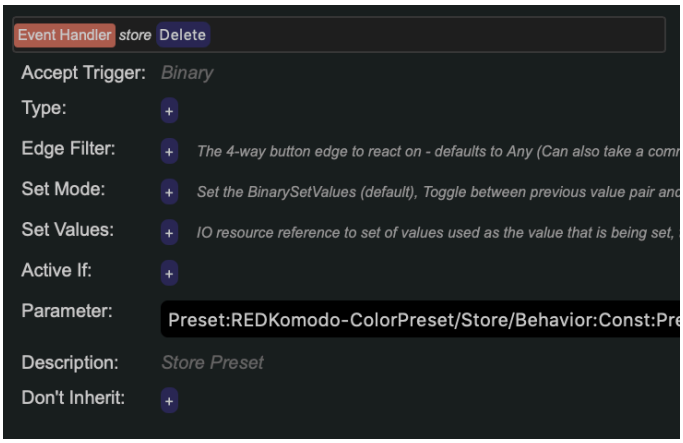
Here “PaintPresetSelect” becomes a new master behavior based on “SKAARHOJ:PaintPresetSelect”.

Sometimes the use of master behaviors is very convenient if the parameter is the same (often involving a constant or variable in the parameter string).

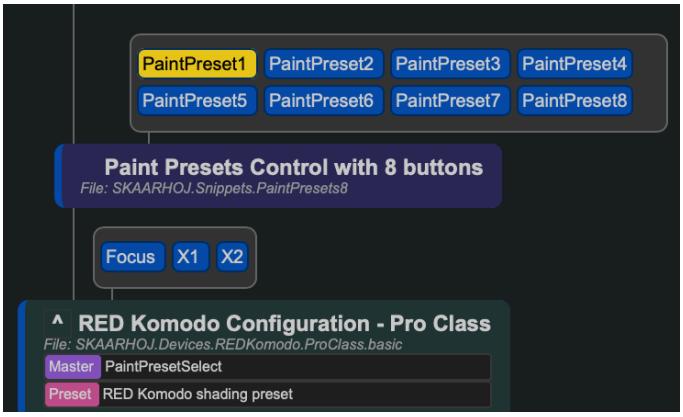


This particular master behavior is driving presets for a RED Komodo camera. Since presets often involve two parameters - a recall and a store parameter when held down for a second - we need to define the parameter two places.

Usually, the recall parameter is defined on the top level and the store parameter is defined - and overriding the top level parameter - down in the event handler.

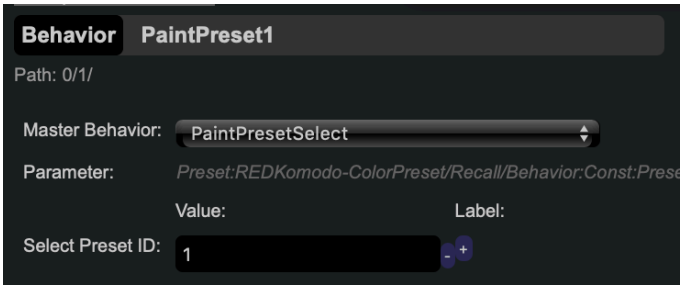


The PaintPresetSelect master is used in an included layer file, SKAARHOJ.Snippets.PaintPreset8, and if we check out one of the behaviors in this included file, we see how it's based on the master behavior defined on the root layer.



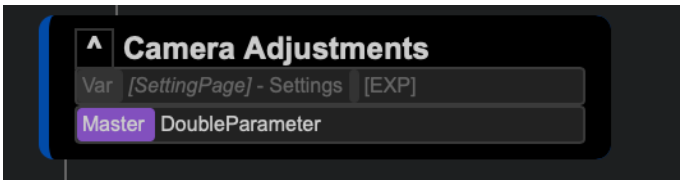
The value that makes the eight preset behaviors different is only the “Select Preset ID” field.

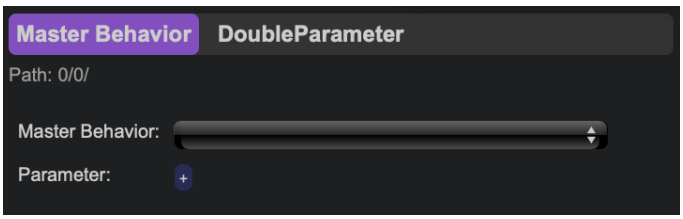
This beautifully demonstrates how clever use of master behaviors and their inheritance in the tree can reduce the amount of redundant information in the configuration.



Master behaviors doesn’t have to base themselves on anything.

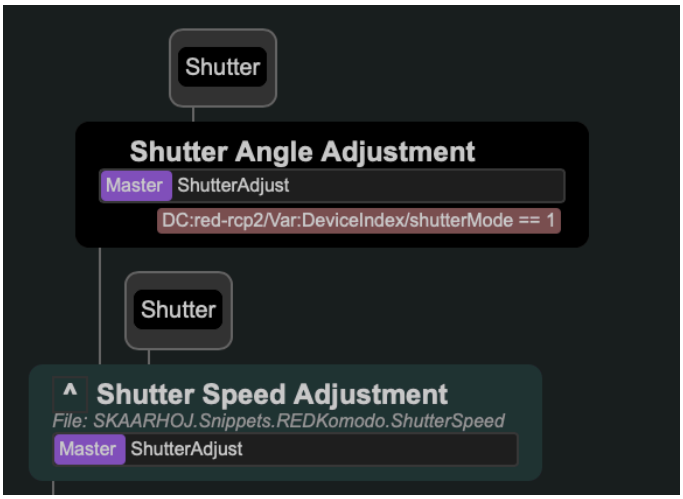
In this example, a master behavior is built from scratch.





Master behaviors defined with custom names can of course also inherit and extent themselves as much as desired.

In this case, a custom master behavior named ShutterAdjust is redefined on a child layer.



Constant Sets

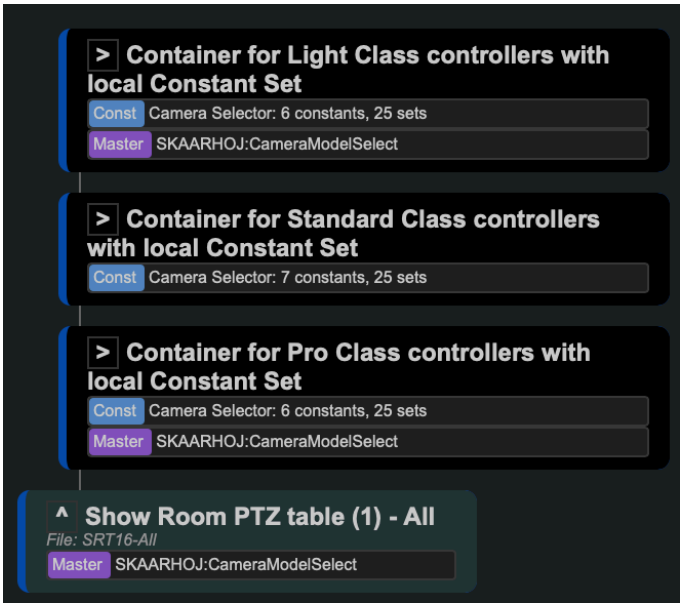
Constant Sets are sets of values which are often repeated a number of times to define cameras, switcher inputs, router outputs and even configuration settings inside Reactor.

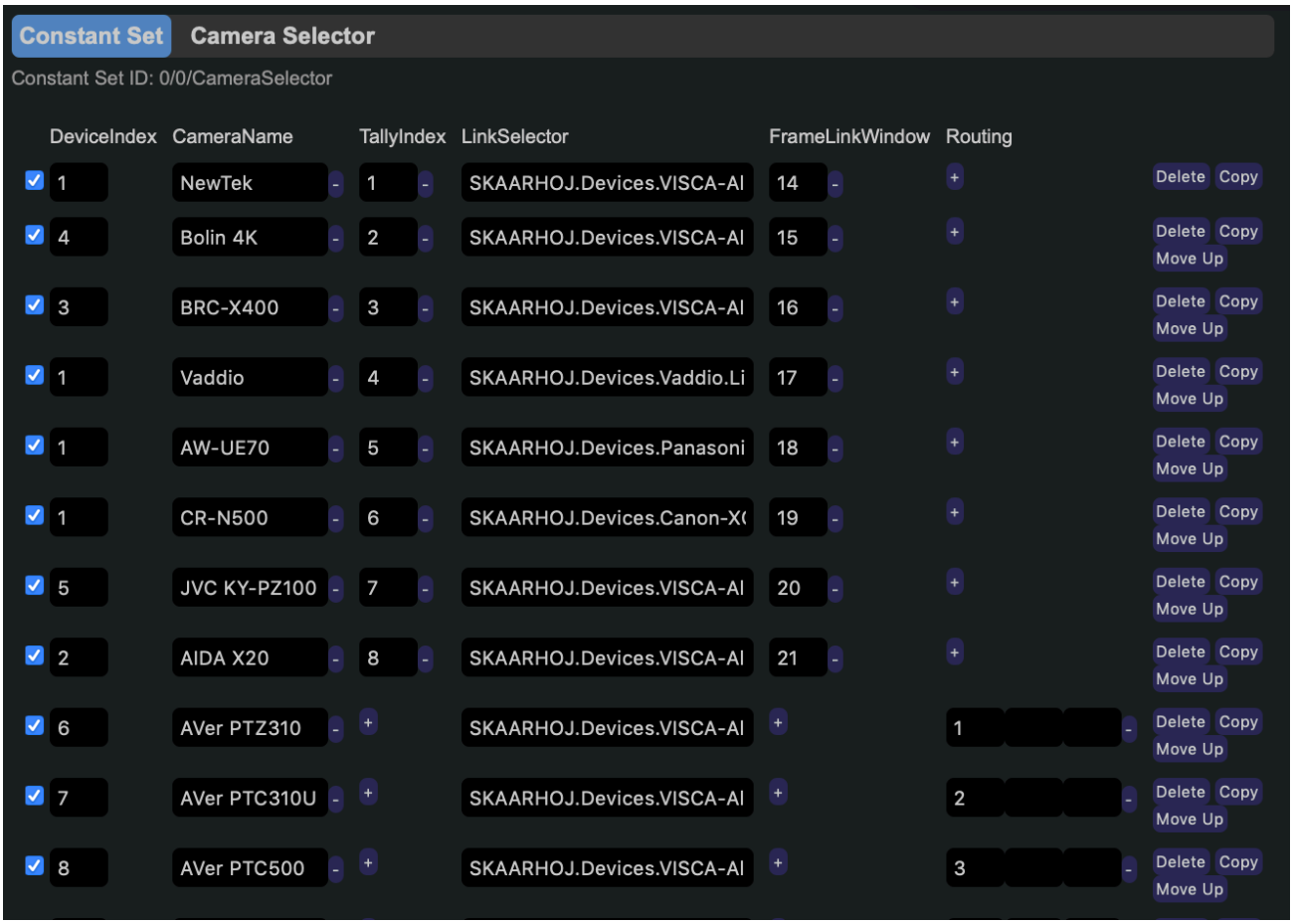
Constants Sets are the way a complex configuration can be made accessible and customizable on selected areas for standard users of a panel or system. They are available through the Home screen of Reactor in many cases so users doesn't have to enter the configuration tree to set up their controller.

In the Home screen of Reactor, the workflow of setting up a controller is:

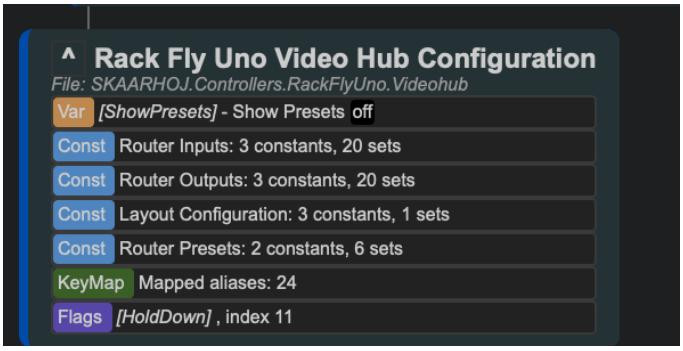
1. Connect panels
2. Connect devices
3. Pick a configuration that fits the combination of panels and devices
4. Edit any highlighted **Constants Sets** to customize camera selectors, source select buttons etc.

In this example (right side), a Constant Set is defined three times with 25 cameras. The difference is the linked configuration which are different between the classes of controllers (Light, Standard and Pro class). This is our show room configuration for PTZ cameras, so its exceptionally ambitious and large. See below.

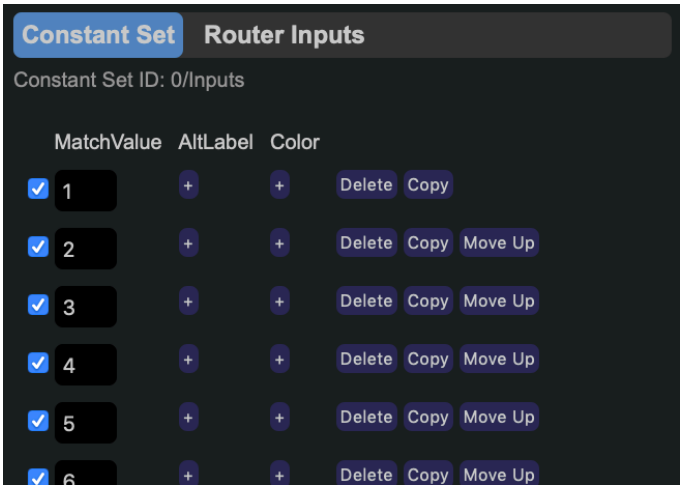




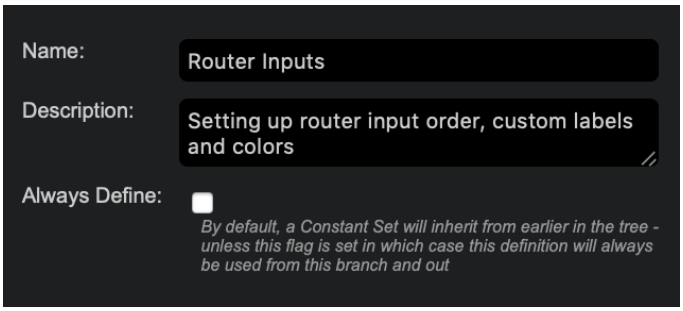
In a configuration for a Video hub router panel, there are four Constant Sets defined: For router inputs, router outputs, router presets and layout configuration.



The Constant Set for Router Inputs would look like this. Rows can be added and removed by the user, alternative labels can be added if desired (default would be the label from the Video Hub itself), and colors can be defined to color code the buttons.



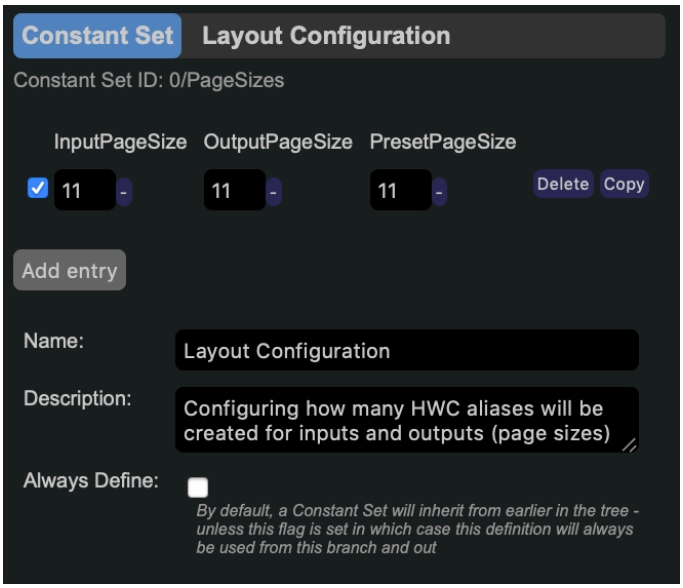
The Constant Set has a title and a description field. Like Variables, the Always Define option determines if a Constant Set will allow itself to be redefined from a position closer to the root of the layer tree. This is usually a good thing to allow so that users can define overriding Constant Sets before including a shared configuration.



The Constant Set contains not only its data but also a definition of the data fields. This drives the UI.

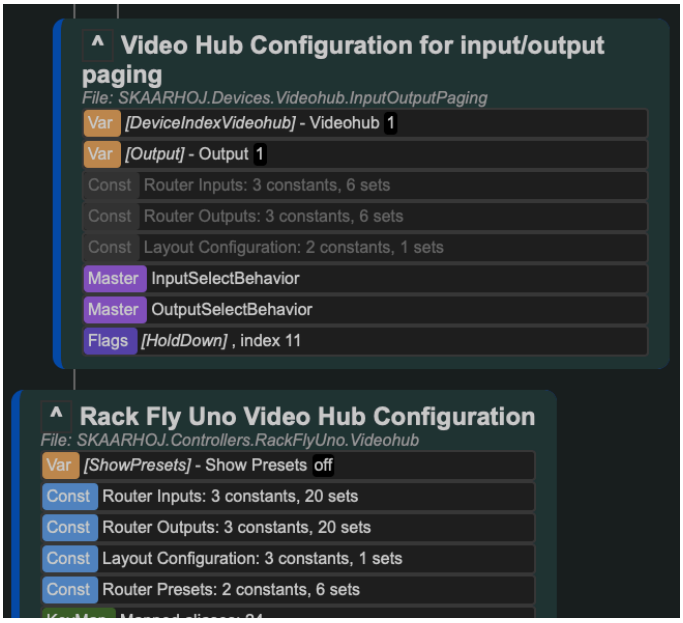
```
"ConstantDefinitions": {
   "AltLabel": {
     "Description": "Alternative label",
     "Type": "String"
   },
   "Color": {
     "Description": "Color",
     "Type": "Color"
   },
   "MatchValue": {
     "Description": "Input number",
     "Type": "Integer",
     "MinItems": 1
   }
 },
```

A Constant Sets can also be used to store settings related to how Reactors generators create the inner configurations which are shared between Rack Fly Uno, Duo and Trio panels in the router example. Each panel will have different settings for the page sizes.

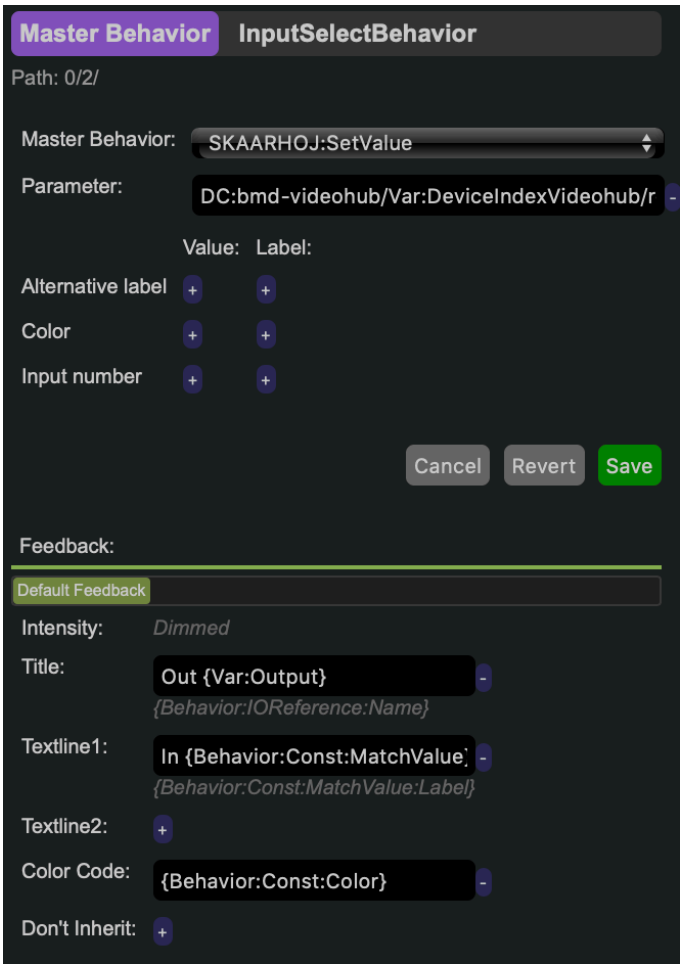


Notice the three Constant Sets grayed out on the layer "Video Hub Configuration for input/output paging". They are inactive because the constant sets on the root layer takes precedence. This is because the Always Define flag is not set inside of them.

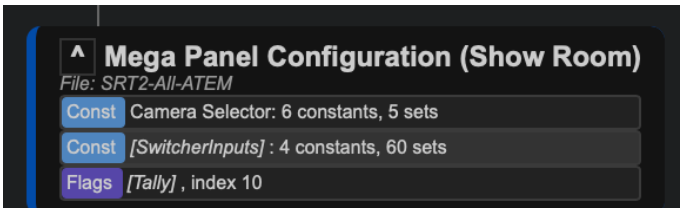
The example perfectly demonstrates why shared configurations should often leave the Always Define flag unset for their Constant Sets and Variables.



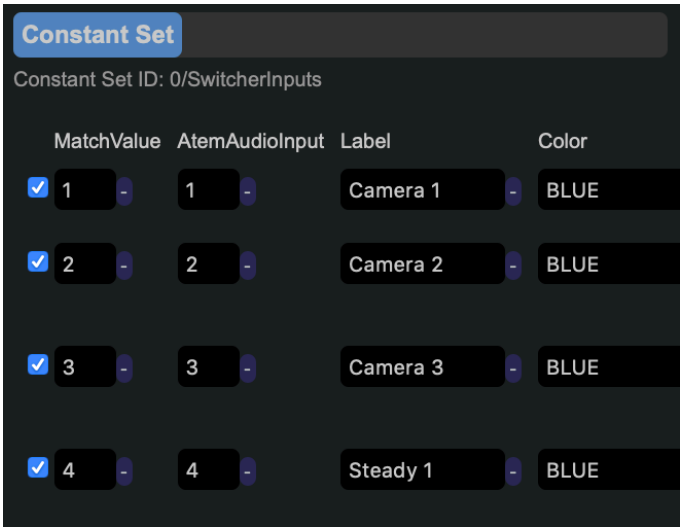
In the Videohub example, the Constant Set entries are being used in a behavior that selects the input sources on a video hub. The master behavior InputSelectBehavior is present to serve this purpose, and looking inside of it we can see how the SetValue master behavior is its basis and how the Default Feedback fields are locally redefined with references to "Behavior:Const:MatchValue" (the router input number) and "Behavior:Const:Color" (the color).



On a SKAARHOJ Mega Panel configuration a large Constant Set is found with switcher inputs defined. In this case 60 rows of inputs are defined!



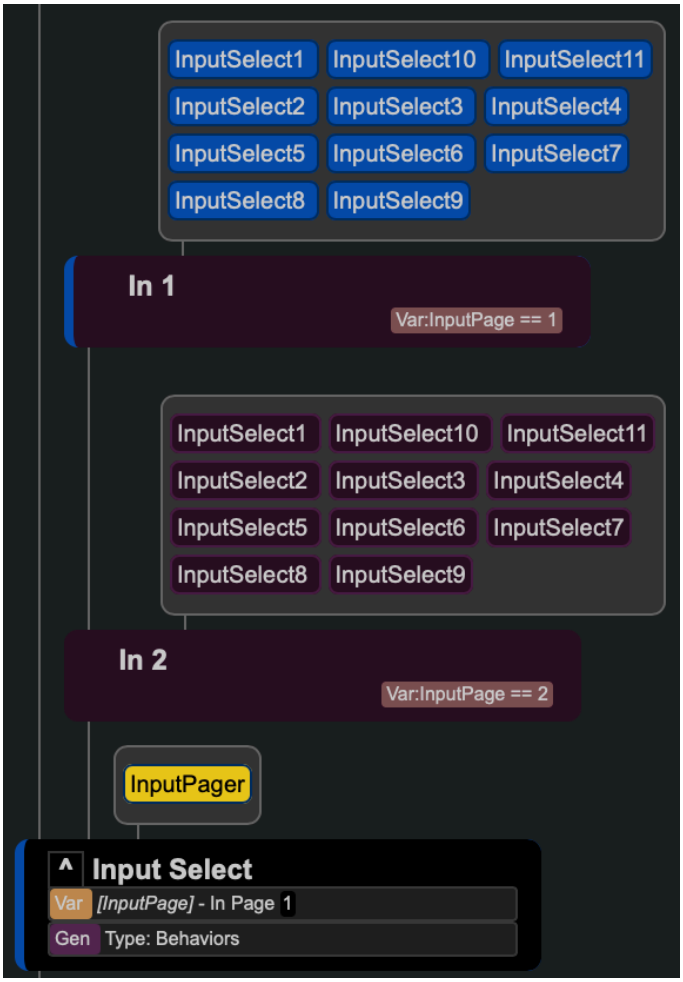
The Constant Set for the switcher inputs is fairly simple: A value for the input, an associated audio input, a label for the displays and the color code.



Generators

Generators are features that generate layers and behaviors at runtime. Automatically generated features are colored purple in the tree making them easy to identify. They cannot be edited since they don't actually exist in the configuration files.

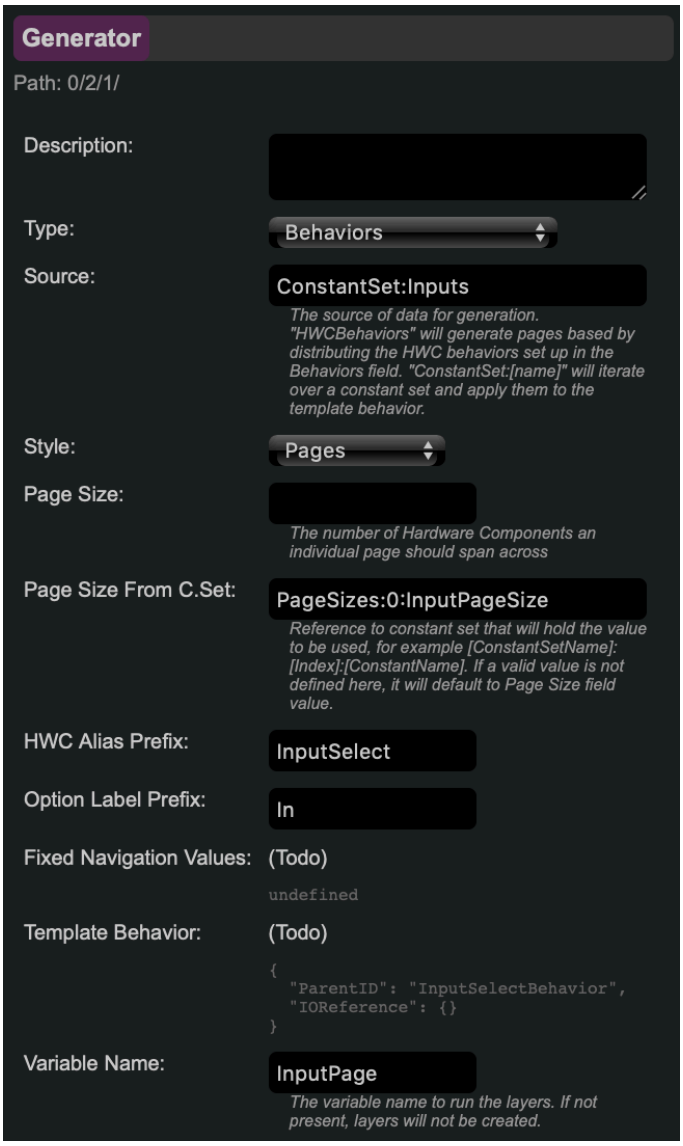
Usually a generator will create content based on the data rows from a Constant Set. The example on the right is from the Video Hub router example used in the description of Constant Sets previously. In this case, two pages of input select buttons are generated by the generator on the layer "Input Select".



The generator on this layer is of the type Behaviors, it uses the Constant Set named “Inputs” and is asked to generated Page-style layers.

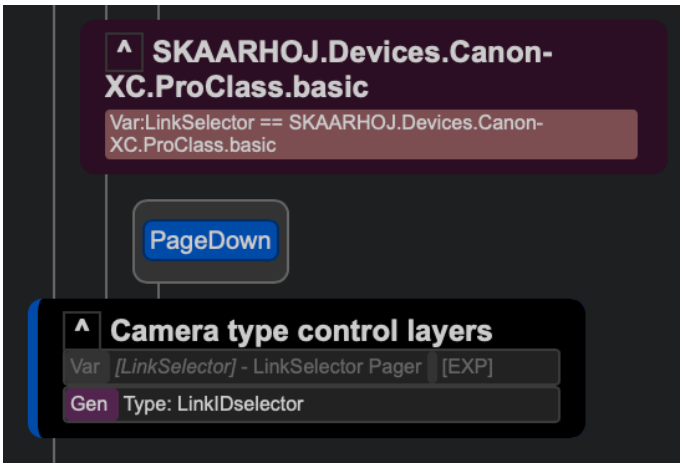
Furthermore we see how the layout constant set is used to pick the page size (PageSizes:0:InputPageSize).

The HWC Aliases are numbers appended to the string “InputSelect” and the variable name to drive the layer visibility is InputPage

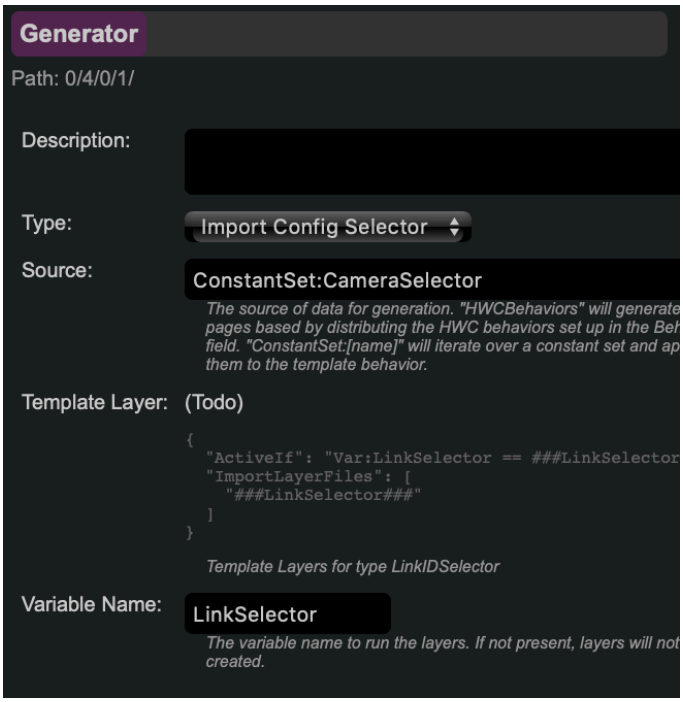


Another type of generator is Import Config Selector. This generator is designed to dynamically include relevant configurations for cameras. This is how Reactor facilitates highly specialized configuration for the exact models of cameras a user has - something very difficult to do in UniSketch on a large scale. The generator also makes it possible to use very generic configurations for a given controller such as a PTZ Extreme and let it adapt to the actually used cameras only by the entries in the constant set.

The the example on the right, a configuration file is included dynamically and the visibility condition automatically set up to match the configuration from the Constant Set.



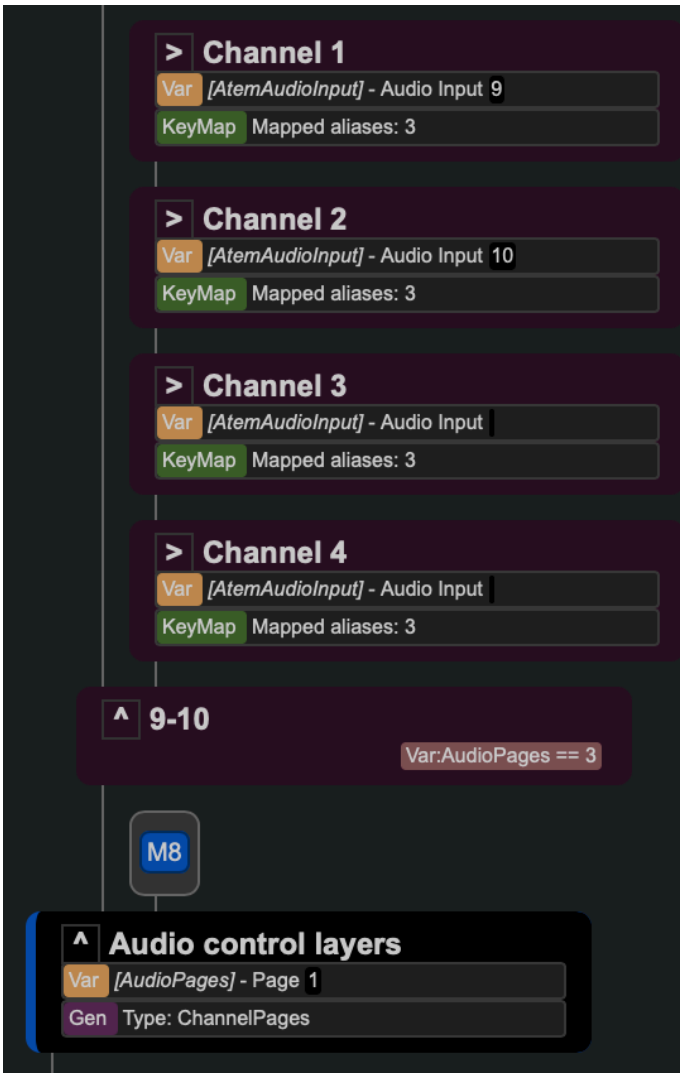
The Import Config Selector generator has various fields with reference to the source Constant Set and the visibility variable etc.

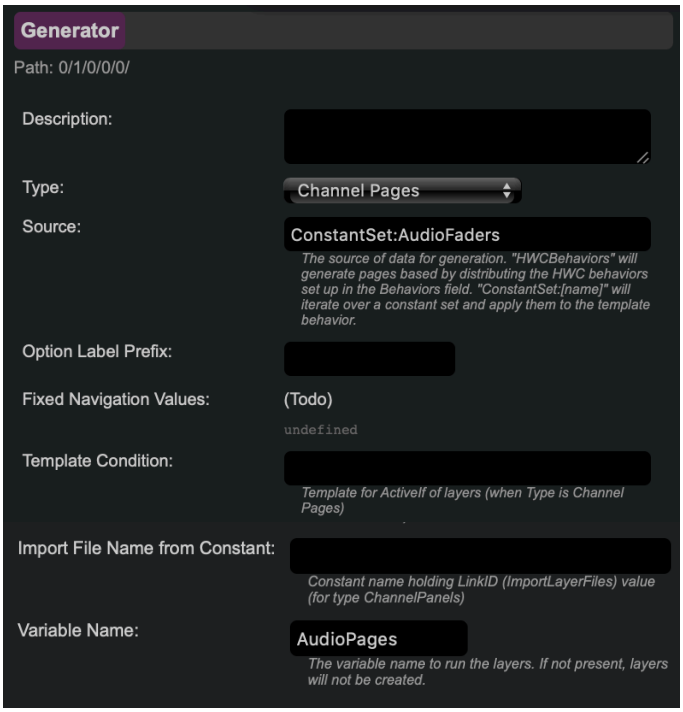


Final type of generator is the Channel Pages type. This is used to manage the case where multiple different cameras on such as a Color Fly or Wave Board controller shall have some parameters mapped onto the motorized sliders and knobs of the controllers.

The Channel Pages solves this by setting specific values for given variables in the tree just before including a tiny include file with just a few definitions for the channel.

Channel Pages are also very useful in a special case with only a single channel on the RCP when an individual behaviors is needed to be managed on the RCP joystick so that discontinuity strategies can be applied successfully.



Example of how Channel Pages type has a certain set of fields different from the other types. There is more under the hood and the UI for generators will be greatly expanded soon.	

Virtual Triggers

Virtual Triggers is conceptually the same as in UniSketch and they are used to set up listeners to parameter values and produce triggers if they change in certain ways.

Virtual Triggers will have a UI in a future update. They exist fully on the server side though and just like generators and many other features, they can be driven by Constant Sets.

There are two main types of virtual triggers with respect to monitoring a parameter: Binary and analog. A Binary type is monitoring a condition using IO references (BinaryActiveIf) and forwarding the result of that into a Behavior. The behavior has everything it needs to trigger something else based on that.

Virtual triggers are also located in the tree and layer visibility can be used as a way to turn virtual triggers on and off.

```
{
            "Name": "Set Program Tally",
            "Description": "Writes program tally to TSL",
            "ConstantSetReference": "Inputs",
            "Mode": "Binary",
            "MuteOnInit": true,
            "BinaryActiveIf": "DC:bmd-atem/Var:DeviceIndex/
ProgramInputVideoSource/Var:MErow == Behavior:Const:Input",
            "Behavior": {
                "ParentID": "SKAARHOJ:SetValue",
                "IOReference": {
                    "Raw": "DC:bmd-atem/Var:DeviceIndex/
ProgramInputVideoSource/2"
                },
                "EventHandlers": {
                    "trigger": {
                        "BinarySetValues": {
                            "Raw": "Behavior:Const:Input"
                        }
                    }
                }
            }
        },
```

A virtual trigger can also monitor a device parameter value and convert that to an analog / absolute value forwarded into a behavior. This means that Reactor can basically map one analog parameter to follow another analog parameter.

```
        {
            "Name": "Analog mapping",
            "Mode": "Analog",
            "MuteOnInit": true,
            "AnalogIOref": {
                "Raw": "DC:bmd-atem/Var:DeviceIndex/TransitionStyle/
Var:MErow/:Current:Normalized"
            },
            "Behavior": {
                "ParentID": "SKAARHOJ:SteplessRange",
                "IOReference": {
                    "Raw": "DC:bmd-atem/Var:DeviceIndex/
TransitionPosition/Var:MErow"
                }
            }
        }
```

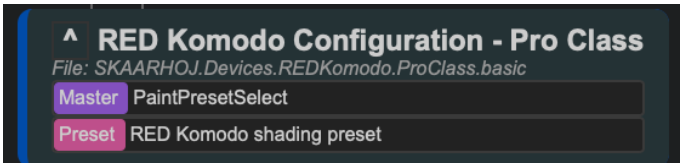
Virtual Triggers can also be based on schedules following a cron-tab syntax

```
            {
                Name:     "Do something on time",
                Mode:     "Cron",
                Schedule: "*/5 * * * * *",
                Behavior: &Behavior{
                    ParentID:    "SKAARHOJ:HoldDown",
                    IOReference: IOref{Raw: "Flag:Tally/Red/
Behavior:Const:Input"},
                },
            },
```

Preset Kinds

Preset Kinds are features of the tree, stored anywhere. In the most basic sense and how it's currently present in configurations, is as a feature that can store and recall settings of a device on a number of select parameters.

In this example a preset, "RED Komodo shading preset" is defined on a layer



The configuration of the example preset looks like this. There is still no UI for it.

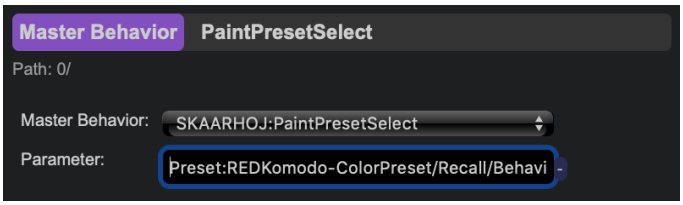
The configuration mentions a list of parameters as a device core reference and an action from that device core.

When the preset kind is being applied as store or recall it will save or recall the value of these parameters.

It works very much like presets known from PTZ cameras.

```
 "PresetKinds": {
    "REDKomodo-ColorPreset": {
      "Name": "RED Komodo shading preset",
      "Description": "Preset storing various shading
related value",
      "Parameters": [
        {
          "DcKey": "red-rcp2",
          "ActionKey": "color"
        },
        {
          "DcKey": "red-rcp2",
          "ActionKey": "temperaturePresets"
        },
        {
          "DcKey": "red-rcp2",
          "ActionKey": "exposureAdjust"
        },
        {
          "DcKey": "red-rcp2",
          "ActionKey": "iso"
        }
      ],
      "Instant": true,
      "LockOnDeviceIndex": true
    }
  }
```

Using a preset kind in Reactor is no different from recalling it on a real external device core. It's just a different format of the Parameter string (IO Reference).




```
Preset:REDKomodo-ColorPreset/Recall/
Behavior:Const:PresetId/Var:DeviceIndex
```

Presets in their full extend are going to be:

- Generally speaking, presets store and recall parameter values and triggers in Reactor
- The values to be stored can be predefined in a preset profile or they can be completely arbitrary (recording what changes)
- A preset profile can define values across device cores, device IDs and parameter dimensions or they can be narrowed down
- With narrow preset profiles, there are inherent opportunities to use presets to copy/paste sets of values between devices and dimensions
- Reactor supports an infinite amount of preset profiles (re-)defined anywhere in the layer tree
- Storage and recall of a preset can work either instantaneously or played back over time
- When recorded and played back over time, values are organized in multiple segments. Each segment is essentially a time line and at the end of a timeline, playback will continue to the next segment either automatically or by user invocation (waiting for user input).
- Playback order of segments can be shuffled and waiting time between segments can be randomized. Playback for a timeline can be looped
- Recording and playback allows cancellation which will restore the state before recording or playback.
- Support for ganged recording and playback of multiple preset numbers, device ids, and dimensions (fairly exotic, honestly)
- Prepared for parameter animation (must be implemented in device collection)

Existing preset commands are show to the right.

NextSegment (trigger)

During recording, this will end the segment and start a new one assuming there has been values added.

During playback, this will skip to next segment

AddUserWait (trigger)

Like NextSegment, but when used during recording it will insert a User Wait at the end and cap the segment length to the last added value.

Play (trigger)

Starts playback

PlayToggle (trigger)

Starts or stops playback

PlayToggleNext (trigger)

Starts or stops playback, except if waiting in which case it will

PlaySkip (trigger)

Starts playback, and skips to next segment if already playing (which may result in stopping altogether)

PlayPause

Starts playback and toggles pause if already playing

PlayPauseNext

Starts playback and toggles pause if already playing, unless at a user wait in which case it skips to next.

Record (trigger)

Starts recording

RecordToggle (trigger)

Starts or stops recording

RecordNewSegment (trigger)

Starts recording, and creates new segment on second press

RecordAddUserWait (trigger)

Starts recording, and creates new segment with user wait on second press

Stop (trigger)

Stops recording or playing

Delete (trigger)

Deletes preset if it is not recording or playing

Cancel (trigger)

If recording, it will stop recording, recall the values from before recording and reinstate the previous content for the preset

If playing, it will stop playing and recall the values from before playing.

Recall (trigger)

Instantly recalls the final values of the first segment of a preset.

RecallStateFromBeforeRecording (trigger)

Recalls the initially stored state of a given preset

DurationRandomExtension (Integer value, ms)

Sets the Random extension value for a given recorded preset.

Loop (bool)

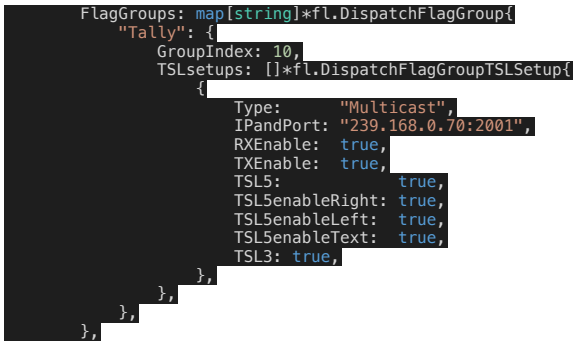
Enable looping for a preset

Shuffle (bool)

Enable shuffle for a preset

Flags

Flags are just binary values, organized in four categories (Red, Green, Blue, White) and grouped by an index.
 	
They can serve the same role as Flags does in UniSketch but with unlimited, unspecified capacity.
	<pre>"FlagGroups": {
	  "HoldDown": {
	  "GroupIndex": 11
	  }
	}</pre>

A flag group can be hooked up with TSL network messages. Currently read and write of messages to a multicast TSL configuration is supported. Direct TCP and UDP for TSL may be supported later.	

Reactor Image Processing

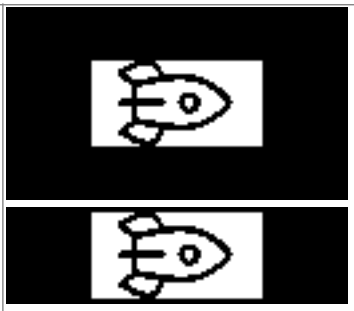
Reactor has a variety of powerful image processing capabilities that can facilitate the rendering of dynamic graphics for monochrome, grayscale and color displays on SKAARHOJ controllers.

The basics

The DataSource decides what kind of graphics you will render. These are the options:

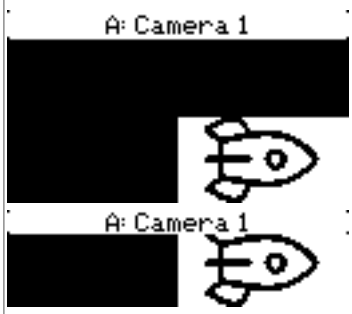
- Icon - a reference to an embedded icon file in Reactor (see list in appendix)
- Inline - a base64 inline image string
- IOref - image taken from an IO reference, such as a thumbnail from a device core.
- QRcode - generates a QR code
- Generator - generates a gray wedge or sample frames
- Composition - produces a layered composition of various types of elements, including those in this list.
- Widget - generates a complex, often dynamic graphic used in specific cases on a SKAARHOJ panel, such
as a VU meter or a strength indicator for a T-bar display.
The output from any of the data sources is rendered onto the destination tile on the panel, but being subject to various filters, alignments and other conveniences. This table provides examples (the full list of settings and values is found inside Reactor in the JSON editor. The examples are rendered onto tiles of the size 128x72 and 128x36 pixels with color capability (these tiles are found on the Blue Pill).

```
"FeedbackDefault": {
    "DisplayGraphics": {
} }

"ShrinkMode": "Ignore",
"DataSource": "Icon",
"IconFile": "icons/64x32bw/Fun-SpaceCraft.png"
```



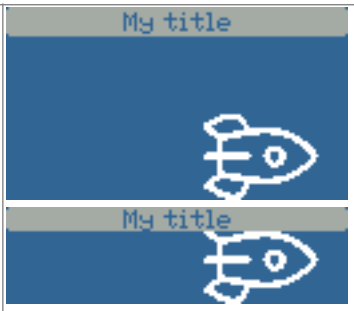
Inclusion of an embedded icon.

```
"Constants": {
    "TextSample": {
"Values": [

"1"

        ],
        "Labels": [
"Camera 1"

] }

},
"FeedbackDefault": {
"DisplayGraphics": {
"ShrinkMode": "Ignore",
"Title": "A: {Behavior:Const:TextSample:Label}",
"SolidHeaderBar": "On",
"DataSource": "Icon",
"IconFile": "icons/64x32bw/Fun-SpaceCraft.png",
"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right"

} }
```



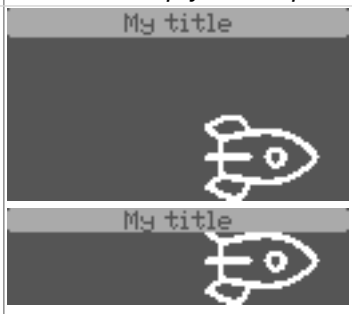
Adding a title with solid bar behind. Title includes a value from a constant (IO reference).
Notice how the title changes the available area for the image underneath. The icon is aligned to the bottom and right (other options include Center by default and Top/ Left of course)

```
"Constants": {
    "BgColor": {
"Values": [

"#336699"

] }

},
"FeedbackDefault": {
"DisplayGraphics": {
"ShrinkMode": "Ignore",
"Title": "My title",
"SolidHeaderBar": "On",
"PixelColorCode": "LIGHTGRAY", "BackgroundColorCode":
"{Behavior:Const:BgColor}", "DataSource": "Icon",

"IconFile": "icons/64x32bw/Fun-SpaceCraft.png",
"ImageFilters": "Invert",
"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageBlendmode":
"Screen"

} }
```



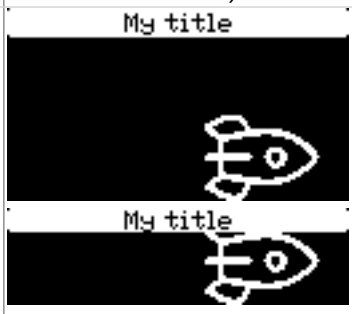
In this example the Pixel and Background colors have been set up. You can use one of the color labels from Reactor or an HTML color definition and even fetch it from an IO reference as we do in this example, picking it up from the constant "BgColor".
An invert filter is applied to the image and then the blend mode is set to Screen so the whites are applied to the background but not the blacks. Other blendmodes include Multiply and Alpha

```
...

}

"ImageBlendmode": "Screen",
"ForceImageMode": "Gray" }
```



The color mode is usually defined by the tile the graphic is generated for, but you can force it to Gray if you need to. (RGB was the default for this tile)

```
...

}

"ImageBlendmode": "Screen",
"ForceImageMode": "Mono" }
```



The color mode is usually defined by the tile the graphic is generated for, but you can force it to Mono if you need to.

```
...

} }

"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageFitting":
"Fit", "ImageBlendmode": "Screen"
```

You can force the image to fit into the available area using the "Fit" function.

```
...

} }

"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageFitting":
"Fill", "ImageBlendmode": "Screen"
```

If you want to fill the entire image area with the image, use the "Fill" function.

```
...

} }

"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageFitting":
"Stretch", "ImageBlendmode": "Screen"
```

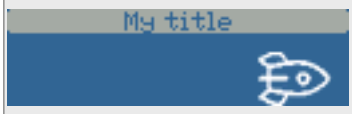
The "Stretch" function will force the image into the entire area and is likely to scale it out of proportions.

```
...

} }

"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageFitting":
"0x20", "ImageBlendmode": "Screen"
```

The ImageFitting field can also take a value on the form (W)x(H) to scale to a specific size. If one of the dimensions are set to zero (like W in this case) it will be scaled proportionally to the other dimension (here: 20 pixels height)

```
...

} }

"ImageVerticalAlign": "Bottom",
"ImageHorizontalAlign": "Right", "ImageFitting":
"0x20", "ImageBlendmode": "Screen", "ShrinkMode": ""
```

ShrinkMode "Ignore":



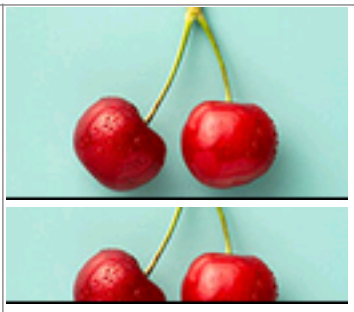
ShrinkMode not set (default):

Adjacent tiles on SKAARHOJ controller displays may have shrink-bits set for their bottom or right edges. This helps to leave a single pixel strip empty of content so tiles are visually distinct from each other. In some cases you may want to disable this border to fille the entire area of a tile. ShrinkMode "Ignore" disables this for image rendering.

Data Sources

The table below demonstrates the various data sources that exists for images.

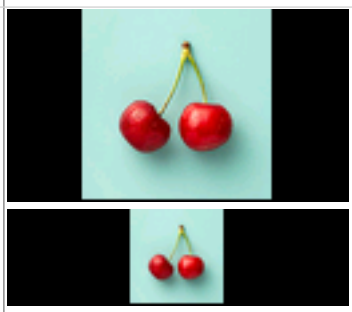
```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Icon",
        "IconFile": "icons/pictures/cherries.jpg",
        "ImageFitting": "Fill"
} }
```



Icon - Fill

Example of icon type, image fitting set to fill. Notice the black line in the bottom - this is the 1 pixel shrink-border to the neighbouring tile.

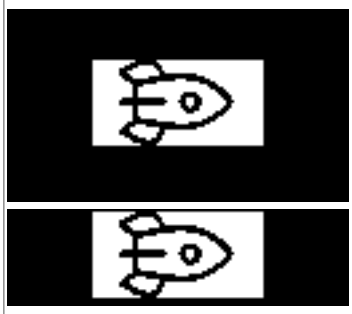
```
"FeedbackDefault": {
    "DisplayGraphics": {
"DataSource": "Icon",
"IconFile": "icons/pictures/cherries.jpg",
"ImageFitting": "Fit"

} }
```



Icon - Fit

Example of icon type, image fitting set to fit.

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Inline",
        "InlineImage":
"png::iVBORw0KGgoAAAANSUhEUgAAAEAAAAAgCAIAAAAt/
+nTAAAAGXRFWHRTb2Z0d2FyZQBBZG9iZSBJbWFnZVJlYWR5ccllPAA
AA2hpVFh0WE1MOm
NvbS5hZG9iZS54bXAAAAAAADw/eHBhY2tldCBiZWdpbj0i77u/
IiBpZD0iVzVNME1wQ2VoaUh6cmVTek5UY3prYzlkIj8+IDx4OnhtcG
1ldGEgeG1sbnM6e
D0iYWRvYmU6bnM6bWV0YS8iIHg6eG1wdGs9IkFkb2JlIFhNUCBDb3J
lIDUuMy1jMDExID
Y2LjE0NTY2MSwgMjAxMi8wMi8wNi0xNDo1NjoyNyAgICAgICAgICAgIj4g
PHJkZjpSREYgeG1
sbnM6cmRmPSJodHRwOi8vd3d3LnczLm9yZy8xOTk5LzAyLzIyLXJkZ
i1zeW50YXgtbnMj
Ij4gPHJkZjpEZXNjcmlwdGlvbiByZGY6YWJvdXQ9IiIgeG1sbnM6eG
1wTU09Imh0dHA6L
y9ucy5hZG9iZS5jb20veGFwLzEuMC9tbS8iIHhtbG5zOnN0UmVmPSJ
odHRwOi8vbnMuYW
RvYmUuY29tL3hhcC8xLjAvc1R5cGUvUmVzb3VyY2VSZWYjIiB4bWxu
czp4bXA9Imh0dHA
6Ly9ucy5hZG9iZS5jb20veGFwLzEuMC8iIHhtcE1NOk9yaWdpbmFsR
G9jdW1lbnRJRD0i
eG1wLmRpZDo4RUJGNzcxQjEyMjA2ODExODA4M0IwMjBFNkI1OUYwNS
IgeG1wTU06RG9jd
W1lbnRJRD0ieG1wLmRpZDpFMkMzNUI4NTk4QjcxMUU3ODE0MUFDQTJ
FM0ZCNzlDRiIgeG
1wTU06SW5zdGFuY2VJRD0ieG1wLmlpZDpFMkMzNUI4NDk4QjcxMUU3
ODE0MUFDQTJFM0Z
CNzlDRiIgeG1wOkNyZWF0b3JUb29sPSJBZG9iZSBQaG90b3Nob3AgQ
1M2IChNYWNpbnRv
c2gpIj4gPHhtcE1NOkRlcml2ZWRGcm9tIHN0UmVmOmluc3RhbmNlSU
Q9InhtcC5paWQ6O
TRCRjc3MUIxMjIwNjgxMTgwODNCMDIwRTZCNTlGMDUiIHN0UmVmOmR
vY3VtZW50SUQ9In
htcC5kaWQ6OEVCRjc3MUIxMjIwNjgxMTgwODNCMDIwRTZCNTlGMDUi
Lz4gPC9yZGY6RGV
zY3JpcHRpb24+IDwvcmRmOlJERj4gPC94OnhtcG1ldGE+IDw/
eHBhY2tldCBlbmQ9InIiPz4EUSuEAAABOklEQVR42sxY2w6FIAwTsv
//
Zc6JJsZMHV03wD34YHCspe5Caa1tHiulPL73+smy6gr9LXoD2GgTgv
U72ceC/
3P+OUgw9PP9qhNAJdR2sxcsEZLgPwAoDwRDotL6YV0DelM/
bXEkEK90lFfnhpMIDEfesGHE6wkHw5f4bDmRFMZ8Vk4YiX+hkplXrn
X7gClGXBhkTohIQ
bxWQzxr1/
nRgx0heA6SuH233UAaJ29XUtMrgPqE8KCgzpaQ4jWSr5BvxeXioISs
OLtmTlINJ5kSyj0
TJDIE4YI0inDPleRJALox0Q2F0GoeUSUItxLfld4bnFfTJjJiwWMSy
50HJHLKHKrciczR
Stw3AwkeOhNLkMKhwSUXMhvMR++F7qNG7mQ86V5IXVqpv2IV99Gh/
gv0uwuZ0Zatul7/
CTAABJXYUo2pweYAAAAASUVORK5CYII="
} }
```

Inline

The 64x32 pixel space craft icon from before. The base64 string is generated by a simple command on the command line (Mac/ Linux):

```
base64 icons/
64x32bw/
Fun-SpaceCraft.png
```

Due to the large data size embedded in configuration text files, this is not recommended on a large scale basis.

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Scan Me!",
        "BackgroundColorCode": "ROSE",
        "DataSource": "QRcode",
        "QRcodeContent": "http://www.skaarhoj.com",
        "QRcodePixelSize": 2,
        "ImageBlendmode": "Multiply"
    }
}
```

QRcode

Generates a QRcode from a text string and with a given pixel size. By default it's rendered black on white, but in this case we set a rose-colored background and use Multiply blend mode to impose it onto the background.
Also notice the header with default white color and no solid header bar. It's generally a bad idea to use image fitting (fit, fill, stretch etc) with pixel-exact graphics, so adjust the QRcodePixelSize instead to fit your display.

(TODO: IO reference) - thumbnail from PTZ

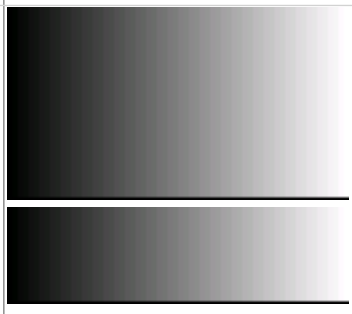
(TODO: IO reference) - thumbnail from Device Core

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Generator",
        "GeneratorConfig": "Clip:Cam1:1"
    }
}
```

Generator - Clip

Sample clips at a given frame rate

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Generator",
        "GeneratorConfig": "Wedge"
    }
}
```



Generator - Wedge

Gray wedge

Data Sources - Widgets

Widgets generates a complex, often dynamic graphic use in specific cases on a SKAARHOJ panel, such as a VU meter or a strength indicator for a T-bar display.

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Widget",
        "Widget": {
            "Type": "Strength",
            "Subtype": "Tbar",
            "Title": "Output",
            "Value": "75%",
            "Data1": {
                "Raw": "750"
            }
} }

}
```

Widget - Strength

Generates a strength meter in grayscale (Work-in-progress). Designed to be rotated 90 CCW

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "DataSource": "Widget",
        "Widget": {
            "Type": "VUMeter",
            "Subtype": "Fixed176x32",
            "Title": "Output",
            "RangeMapping":
"0,77,154,231,308,385,462,538,615,692,769,846,923,10
00",
            "Data1": {
                "Raw": "90"
            },
"Data2": {

                "Raw": "768"
            },
"Data3": {

                "Raw": "450"
            },
"Data4": {

                "Raw": "900"
            }
} }

}
```

Widget - VU Meter

A VU meter for audio. Has a fixed size of 176x32 pixels because of the background image (Work-in- progress). Designed to be rotated 90 CCW
Data1 and Data2 is normalized values from 0-1000 for the L+R bars. Data3 and Data4 are the peak points.
The RangeMapping can be used to set which values from 0-1000 will comply with the 13 steps on the scale. The example is the default completely linear scale. Crafting the RangeMapping values carefully can make the VU meter perfectly match the values from any host system for accurate measuring.
(The Widget presented above is rendered at exactly 176x32 and not on any of the Blue Pill tiles.)

Data Sources - Compositions

Compositions is a particularly complex data source since it represents a layered structure that can include other data sources as well as graphical primitives and text. The table below demonstrates examples:

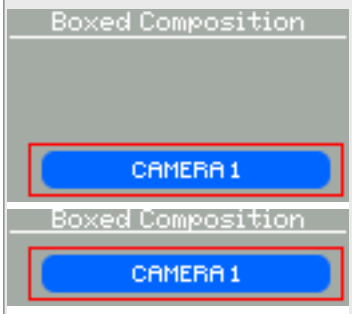
```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "My Title Line",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "Composition": {
            "Type": "Graphics",
            "Graphics": [
                {
                    "Type": "Text",
                    "Text": "Camera 1",
                    "TextSize": 20,
                    "TextFont": "NotoSans-Bold"
}, {

} ]

        },
        "ImageBlendmode": "Alpha",
        "ShrinkMode": "Ignore"
} }

"Type": "Rectangle",
"ColorCode": "#0066ff",
"RoundedCorner": 10,
"X": 5,
"Y": 3,
"W": -10,
"H": -6
```

Basic Composition

Renders true type text onto a blue rectangle with rounded corners. The text is centered in the canvas both horizontally and vertically. The Rectangle is defined from upper left corner with X and Y starting position being 5,3. Since the W and H of the rectangle is negative, it will pick up the canvas width/height and subtract these numbers.

Notice that ImageBlendMode is set to Alpha since a composition is transparent by default, otherwise it would render black background all over the canvas area (like here:):

```
"Variables": {
    "ShowComposition": {
        "Name": "Show Composition",
        "Description": "If equal to on, the composition
will render",
        "OptionList": [
            {
                "ValueID": "on",
                "Name": "Show"
            },
            {
                "ValueID": "off",
                "Name": "Hide"
            }
        ],
        "Default": [
"on"

] }

},
"IOReference": {},
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Boxed Composition",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "Composition": {
            "ActiveIf": "Var:ShowComposition == on",
            "Box": {
                "Width": -10,
                "Height": 20,
                "VerticalAlign": "Bottom",
                "HorizontalAlign": "Right",
                "OffsetX": -2,
                "OffsetY": -2
},
"ShowBox": true, "Type": "Graphics", "Graphics": [

                {
                    "Type": "Text",
"ColorCode": "WHITE", "Text": "Camera 1", "TextFont":
"Small"

}, {

} ]

        },
        "ImageBlendmode": "Alpha",
        "ShrinkMode": "Ignore"
} }

"Type": "Rectangle",
"ColorCode": "#0066ff",
"RoundedCorner": 5,
"X": 5,
"Y": 3,
"W": -10,
"H": -6
```



Composition canvas

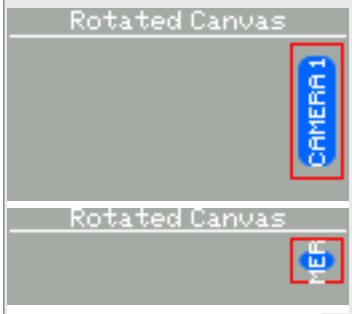
A composition has a canvas which by default is equal to the target it is placed on (in this case that would be the gray area under the white line of the tile). But you can change it's width, height, offset and alignment like here where the width is tile-width minus 10, height is fixed, it's aligned to the bottom right and offset -2,-2 from the edges.

The ShowBox property is true, which renders the red border around the canvas of the composition as a temporary help to the designer.

The text font is changed to "Small" which is a truetype pixel font that renders sharp edges at sizes 8 (default), 16, 24 etc.

Compositions have an ActiveIf field which decides if it's rendered or not. An empty field equals true. In this case we are using a variable to determine whether to render it or not.

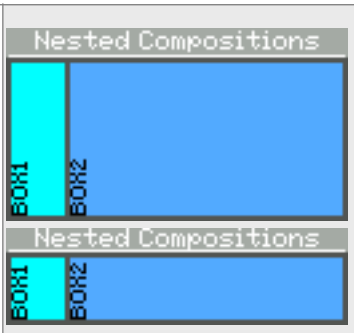
```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Rotated Canvas",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "Composition": {
"Box": {
"CanvasOrientation": "CCW", "Width": -10,
"Height": 20, "VerticalAlign": "Bottom",
"HorizontalAlign": "Right", "OffsetX": -2,
"OffsetY": -2

            },
            "ShowBox": true,
            "Type": "Graphics",
            "Graphics": [
                {
                    "Type": "Text",
                    "ColorCode": "WHITE",
                    "Text": "Camera 1",
                    "TextFont": "Small"
}, {

} ]

        },
        "ImageBlendmode": "Alpha",
        "ShrinkMode": "Ignore"
} }

"Type": "Rectangle",
"ColorCode": "#0066ff",
"RoundedCorner": 5,
"X": 5,
"Y": 3,
"W": -10,
"H": -6
```



Rotation of the canvas

The only change to the previous configuration (except the removal of the ActiveIf field) is that CanvasOrientation is set to CCW (Counter Clock Wise). This changes the reference for alignment, offset as well as width and height. For example the width is now (original height)-10. Compare it to the previous example to see the similarity.

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Nested Compositions",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "Composition": {
"Box": {

"CanvasOrientation": "CCW" },

"BackgroundColorCode": "DARKGRAY", "Type": "Layers",
"Layers": [

{
"Box": {

                        "Width": -4,
                        "Height": 20,
                        "VerticalAlign": "Top",
                        "OffsetY": 2
                    },
                    "BackgroundColorCode": "CYAN",
                    "Type": "Graphics",
                    "Graphics": [
                        {
                            "Type": "Text",
                            "Text": "Box1",
                            "TextHorizontalAlign":
"Left",
                            "TextVerticalAlign": "Top",
                            "TextFont": "Small"
} ]

}, {

} ]

} ]

},

        "ShrinkMode": "Ignore"
    }
}

"Box": {
    "Width": -4,
    "Height": -26,
    "VerticalAlign": "Bottom",
    "OffsetY": -2
},
"BackgroundColorCode": "ICE",
"Type": "Graphics",
"Graphics": [
    {
        "Type": "Text",
        "Text": "Box2",
        "TextHorizontalAlign": "Left",
        "TextVerticalAlign": "Top",
        "TextFont": "Small"
```



Nested compositions

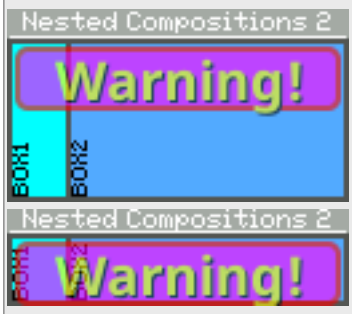
In this example the composition is of type "Layers" which holds another two compositions. The main composition is rotated CCW and has a dark gray background color (so we don't need Alpha blend mode here).

The first nested composition is 20 pixels high and width-4, aligned to the top and offset 2 pixels down. Since the main canvas is rotated CCW, all of this makes sense if you "turn your head" counter clock wise too to see it from that reference. The background of this box is cyan and we render a basic text "Box1" in the top left corner

The second nested composition is bottom aligned, has a height equal to the main composition minus 26 pixels (which accommodates 2 pixels borders three places and the 20 pixel height of the first nested composition). In this composition we also render a basic text "Box2" in the upper left corner.

Nested compositions and boxes helps us to divide the tiles into flexible sections where we can place any fixed or dynamic content from Reactor.

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Nested Compositions 2",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "IOReference": {},
        "Composition": {
            "Box": {
                "CanvasOrientation": "CCW"
            },
            "BackgroundColorCode": "DARKGRAY",
            "Type": "Layers",
            "Layers": [
{
"Box": {

                        "CanvasOrientation": "CW",
                        "Height": 30,
                        "VerticalAlign": "Top"
                    },
                    "Type": "Graphics",
                    "Graphics": [
                        {
                            "Type": "Text",
                            "ColorCode": "YELLOW",
                            "Text": "Warning!",
                            "TextSize": 20,
                            "TextFont": "NotoSans-Bold"
}, {

}, {

} ],

                    "Transparency": 40
                },
.... (Insert the two boxes from prev example here) ...
] },

        "ShrinkMode": "Ignore"
    }
}

"Type": "Text",
"ColorCode": "BLACK",
"OffsetX": 1,
"OffsetY": 1,
"Text": "Warning!",
"TextSize": 20,
"TextFont": "NotoSans-Bold"
"Type": "Rectangle",
"ColorCode": "PINK",
"LineWidth": 2,
"StrokeColorCode": "RED",
"RoundedCorner": 4,
"X": 4,
"Y": 4,
"W": -8,
"H": -8
```



Overlapping composition layers

In the basic example the two nested compositions didn't overlap, but if they did, the top one would occlude the bottom one.

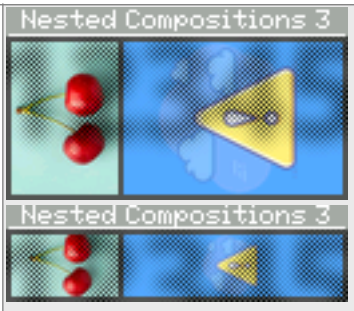
In this example a new composition is added that spans across the two boxes from before. The box of that composition is rotated CW which brings it "back" to the original orientation. The height is set to 30 and it's top-aligned. Implicitly the width matches the parent composition.

The composition itself is of type "Graphics" and has two text items and a rectangle at the bottom.
The text items are simply rendering the same text, but with a black shadow under, offset 1,1 pixel.

The rectangle is with pink fill, red stroke of 2 pixels and with rounded corners. Width and height is picked up from the canvas and 8 pixels are subtracted.

Finally, notice the Transparency for the composition is set to 40% (0-100, 100=completely transparent)

```
"FeedbackDefault": {
    "DisplayGraphics": {
        "Title": "Nested Compositions 3",
        "BackgroundColorCode": "LIGHTGRAY",
        "DataSource": "Composition",
        "Composition": {
"Box": {

                "CanvasOrientation": "CCW"
            },
            "BackgroundColorCode": "DARKGRAY",
            "Type": "Layers",
            "Layers": [
{

"Mask": { "Box": {

        "CanvasOrientation": "CW"
    },
    "BackgroundColorCode": "#000000",
    "Type": "Graphics",
    "Graphics": [
        {
            "Type": "Text",
            "ColorCode": "WHITE",
            "Text": "12345",
            "TextVerticalAlign": "Top",
            "TextSize": 50,
            "TextFont": "NotoSans-Bold"
        }
],

"ImageFilters": "GaussianBlur=3" },

"ShowMask": false,
"DisableMask": false,
"Type": "Image",
"Image": {
    "DataSource": "Icon",
                    "IconFile": "icons/64x32bw/Pattern-
CheckerFine-256x256.png",
                },
                "Blendmode": "Multiply"
}, {

} },

{
"Box": {

                    "Width": -4,
                    "Height": -46,
                    "VerticalAlign": "Bottom",
                    "OffsetY": -2
                },
                "BackgroundColorCode": "ICE",
                "Type": "Image",
                "Image": {
                    "DataSource": "Icon",
                    "IconFile": "icons/Scenarium/2997979.png",
                    "Fitting": "Fit"
} }

] },

        "ShrinkMode": "Ignore"
    }
}

"Box": {
    "Width": -4,
    "Height": 40,
    "VerticalAlign": "Top",
    "OffsetY": 2
},
"BackgroundColorCode": "CYAN",
"Type": "Image",
"Image": {
    "DataSource": "Icon",
    "IconFile": "icons/pictures/cherries.jpg",
    "Fitting": "Fill"
```



Masks and image filters

In this example the underlying two boxes are still in place, but the height is adjusted a bit and they are filled with images instead of text labels. In the first case, the cherry picture is instructed to fill the entire composition (centered by default) while the second example is fitting the icon in. The icon is an embedded PNG file with transparency so it blends in with the ice colored background.

The top layer is a checker box image (alternating black and white pixels) which is multiplied onto the background (so white colors become transparent) through a mask. The mask image is the real tricky part: It's generated by a Text type graphics on a canvas rotated back to normal, and then blurred with an image filter.

To understand the components better, this is what you get with DisableMask set to true (just the checkerboard image) and remove the blend mode setting:

If ShowMask is set to true instead you see this:

Combined with no Multiply mode:

```
"FeedbackDefault": { "DisplayGraphics": {

"DataSource": "Composition", "Composition": {

"Box": {}, "Type": "Layers", "Layers": [

{
"Box": {

"Width": 35,
"Height": 35, "VerticalAlign": "Top", "HorizontalAlign": "Left"

},
"Type": "Layers", "Layers": [

{
"Box": {},

"Type": "Graphics", "Graphics": [

{
"Type": "Rectangle", "LineWidth": 2, "StrokeColorCode": "ICE", "RoundedCorner": 6,
"X": 1,
"Y": 1,
"W": -2,
"H": -2

} ]

}, {

"Box": {}, "Mask": {

"Box": {}, "BackgroundColorCode": "#000000", "Type": "Graphics",
"Graphics": [

{
"Type": "Rectangle", "ColorCode": "WHITE", "RoundedCorner": 6,

    "X": 1,

    "Y": 1,

"W": -2, "H": -2

} ]

},
"Type": "Image", "Image": {

"DataSource": "Icon",
"IconFile": "icons/pictures/cherries.jpg", "Fitting": "Fill"



}

}

] },

{
"Box": {

"Width": -38,
"Height": 12, "VerticalAlign": "Top", "HorizontalAlign": "Right"

},
"Type": "Layers", "Layers": [

{
"Box": {},

"Type": "MonoText", "MonoText": {

"Text": "My title text", "FontFace": 1, "OffsetY": 2, "ColorCode": "WHITE"

} },

{
"Box": {},

"Type": "MonoRect", "MonoRect": {

"RoundedCorner": 1,

"ColorCode": "WHITE" }

} ]

}, {

} ]

} ]

},

"ImageBlendmode": "Alpha" }

}

"Box": { "Width": -38,

"Height": -12, "VerticalAlign": "Bottom", "HorizontalAlign": "Right"

},
"Type": "Graphics", "Graphics": [

{
"Type": "Text",

"ColorCode": "WHITE", "Text": "カメラ1", "TextFont": "Unifont"
```

This example demonstrates a "typical" complex composition with nested layers.

The first layer defines a box of 35x35 pixels (top/left aligned) and inside this layer (composition) we will place the image and a border around it as nested layers. The ice blue rectangle with 6 pixel rounded corners and a line width of 2 has X,Y,W,H set so the double line width is accommodated. On the next layer the image itself is placed with a fill scaling but masked by another rounded corner rectangle, this time filled with white instead of stroked with ice blue.

Another box for the title is defined and inside of that a MonoText and MonoRect object is placed. These are legacy rendering of text and rounded corner rectangles for monochrome displays and are useful if you desire a style coming from UniSketch's tradition.

The lower box is aligned to bottom right and relative to the parent composition size and contains Japanese text ("Camera 1") rendered with the embedded Unifont (all character sets).

If you set ShowBoxOnTop to true for the three main layers/ compositions you will see the bounding boxes shown with red lines, useful for debugging:

Appendix

Icons

(Todo: List of embedded icon paths, their dimensions and color state to be)

Reactor / UniSketch color codes

- DEFAULT • OFF
- WHITE
- WARM
- RED
- ROSE
- PINK
- PURPLE
- AMBER
- YELLOW
- DARKBLUE • BLUE
- ICE
- CYAN
- SPRING
- GREEN
- MINT
- LIGHTGRAY • DARKGRAY • BLACK

HTML colors on the form #ff6600 or #f60 are often allowed too.

Image Filters

A comma separated, ordered list of image filters:

- Grayscale
- FlipHorizontal
- FlipVertical
- Invert
- Sharpen=[0:10] , example "Sharpen=5"
- GaussianBlur=[0:10]
- Threshold=[0:100, 50 is default]
- Saturation=[-100:500]
- Contrast=[-100:100, default 0]
- Brightness=[-100:100, default 0]
- Gamma=[0.0:2.0, default 1]
- Colorize=[Hue 0:360];[Saturation 0:100];[Percentage 0:100] • Hue=[-180:180]

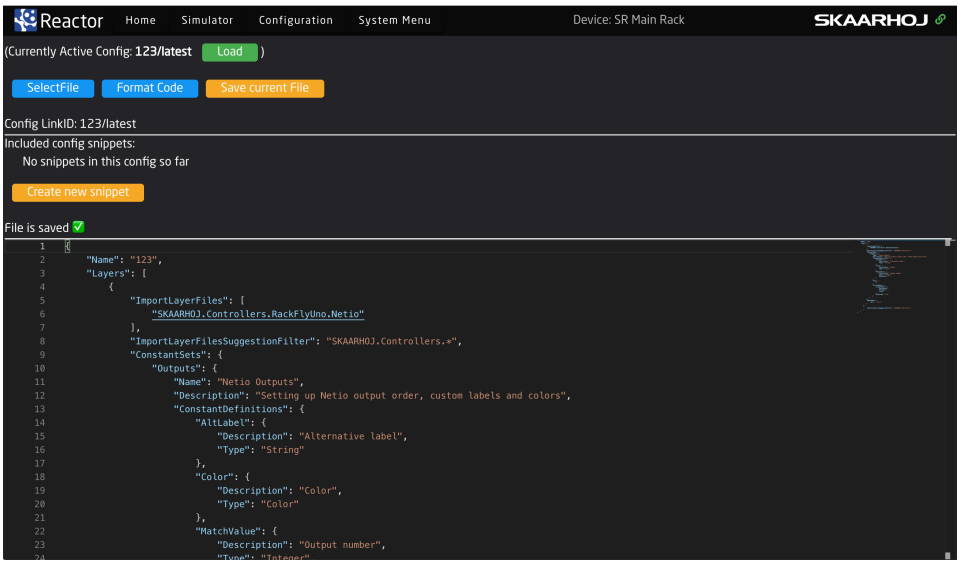
Embedded Fonts

These are font names you can use with Text type graphic elements (in compositions)

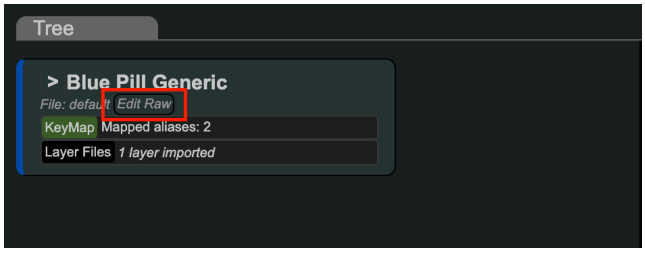
NotoSans-Bold
NotoSans-BoldItalic
NotoSans-Italic
NotoSans-Regular
m5x7
m3x6
Small
Dogica
Dogica-Bold
437Win
Pixellari
SuperStar
Unifont

Neo Mode

For the really adventurous (those, who would have taken the red pill to know the real truth), there is a code editor that provides access directly to configuration on the JSON code level.



To enter Neo Mode, taking the red pill is not needed. On the configuration, click on Edit Raw in the Tree to open the desired layer's snippet.

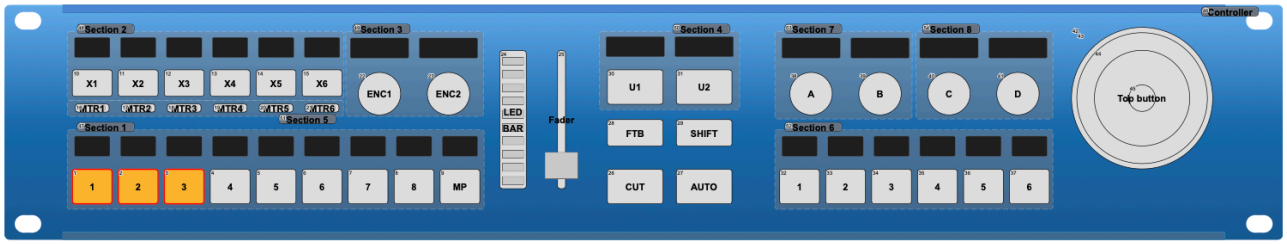


Compared to UniSketch

UniSketch vs Reactor

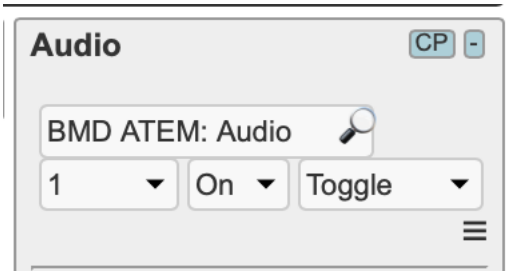
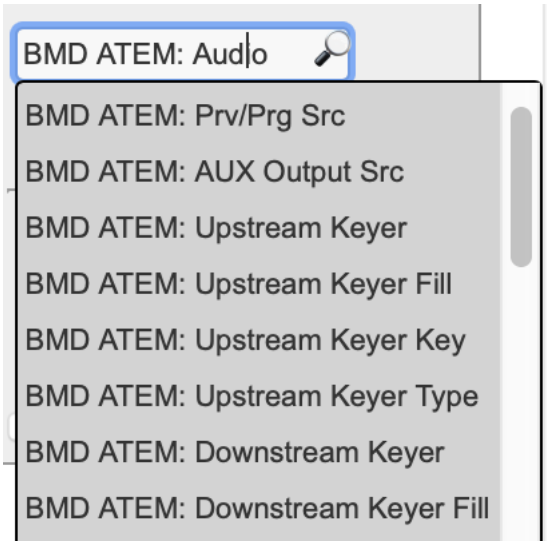
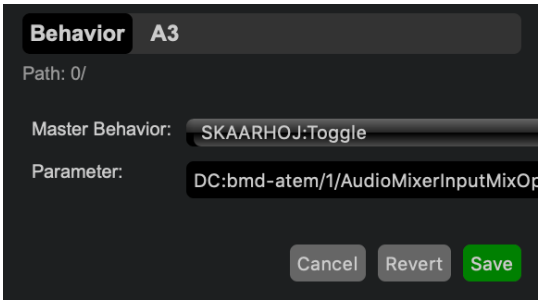
While Blue Pill and UniSketch are complementary technologies, it's also true that there is an overlap between the platforms. Everything related to device control and configuration is essentially provided on both platforms with pros and cons associated. UniSketch is extremely road tested, but also facing resource limitations which Reactor has essentially none. Reactor on the other hand is new technology with the need to mature.

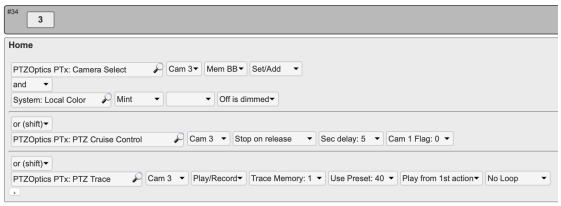
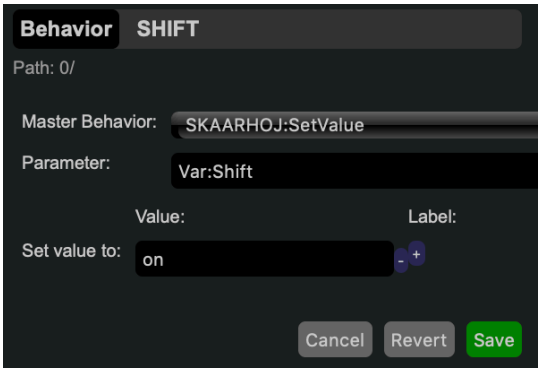
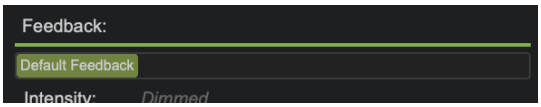
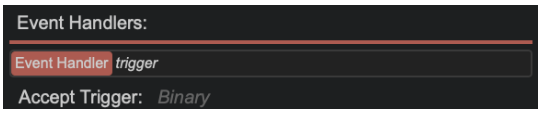
In this section, we will explore the differences with some examples from a Rack Fusion Live controller. It's assumed familiarity with UniSketch and would like to see how the heritage of UniSketch is applied in Reactor.

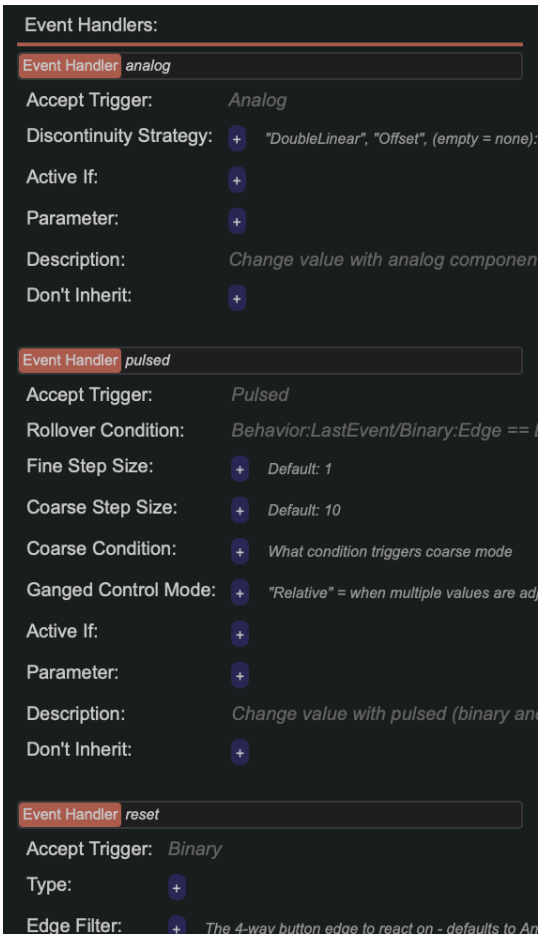


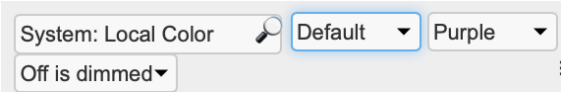
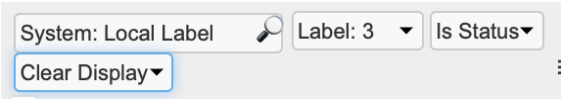
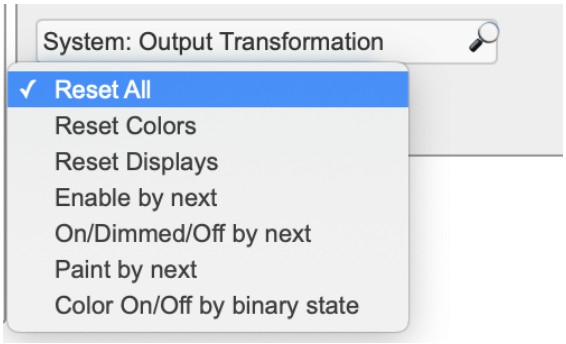
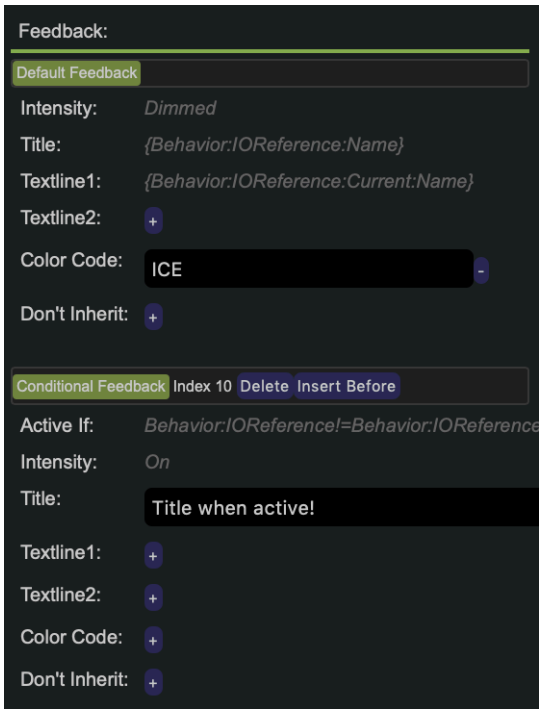
Actions, Behaviors, Events, Feedback...

First, let's look at some terms used in both systems.

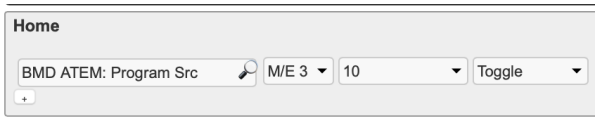
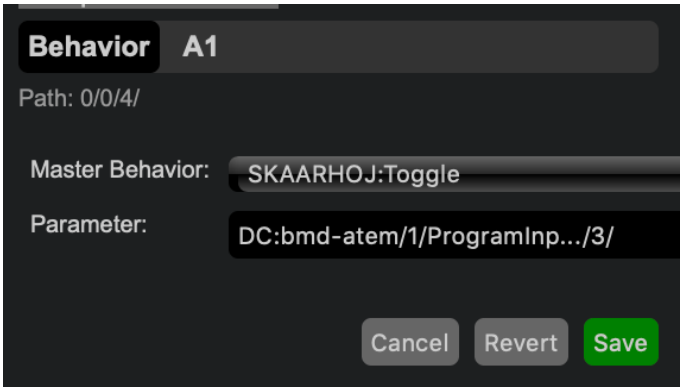
	UniSketch	Reactor
Actions	In UniSketch, *actions* is the most known term for how we create functionality on the controller: We assign an action to a button for example. This is picked from a list of fixed actions provided by the device core and bound to each action is often found a modifier for how it should behave (like Hold Down, Toggle, Cycle etc). Furthermore, whether an action will work on a button, encoder, fader or joystick is not obvious. The far majority of actions will work only on the most obvious types: Like selecting a switcher source with a button, but not with a joystick. Or change PTZ tilt speed with a joystick, but not a button.   	In Reactor, *actions* is not a used term and behaviors have distinct properties such as a single parameter, single master behavior, title etc.  The most close counterpart for *actions* might be the event handlers of a behavior. Where in UniSketch many actions may be assigned to a behavior, in Reactor many event handlers could be assigned to a behavior. Where UniSketch also (mis-)uses *actions* to place colors and graphics on components, this is not necessary in Reactor. Feedback exists independently of the events assigned.

	UniSketch	Reactor
Behaviors	It may be a little known fact, but actually the total number of *actions* for a hardware component in UniSketch is called a *behavior:*  This UniSketch behavior consists of four actions, spread across three shift levels (normal, shift=1, shift=2)	In Reactor *behavior* means exactly the same! It's how a given hardware component behaves when the behavior is active (which is defined by the layer visibility and the occlusion it may be subject to). Most notably a Reactor behavior separates how feedback and events are implemented *completely* from the parameter they affect.  Behaviors consist of both events (pressing a button or moving a fader) and feedback (color and display content back to the controller).  

	UniSketch	Reactor
Events	Events is not a concept in UniSketch, but essentially embedded in the hardcoded nature of *actions*. Whether an action responds to a given event type (like a button press or joystick speed value) will depend on the underlying code.	The most close counterpart for *actions* might be the event handlers of a behavior. Where in UniSketch many actions may be assigned to a behavior, in Reactor many event handlers could be assigned to a behavior:  An Event handler can accept only a single type of trigger of the four standard triggers: Binary (ie. buttons), Pulsed (ie. encoders), Analog (ie. faders) and Speed (ie. joysticks), but here is the beauty: Specifying an event preprocessor can convert *any* trigger type to *any other* trigger type and with a huge number of options for how that conversion is done. This is used everywhere in Reactor to make sure that all mainstream master behaviors will respond in some way regardless of what hardware component they are applied to.

	UniSketch	Reactor
Feedback	Feedback in UniSketch has to be defined as actions with no "function", just the ability to affect the color of the components or display contents. Here is the Local Color action that offers ways to set colors for dimmed and on states. It has a lot of fixed logic in it since it tries to include rules for the on and dimmed state along with color. In Reactor that is separated out into an Intensity property and the color property itself.  Setting content for a display is quite convoluted too, having to refer to a label number defined elsewhere. In Reactor that is just a text input field in the inspector.  Due to UniSketch's very limited configuration engine, a collection of output transformations were introduced to do what Reactor does far more straight forward with its feedback engine. 	Reactor allows for custom feedback to any behavior. There is default feedback (active by default) and there is conditional feedback (active when a given condition is true, like if a source is routed to a given output). Furthermore feedback is inherited from previous layers and master behaviors which means that most of the time defining feedback is not necessary because it's perfectly set up from previously in the tree. Below example shows default and conditional feedback where the gray values are the inherited values and the color "ICE" is the only overwritten value defined on the behavior itself: 

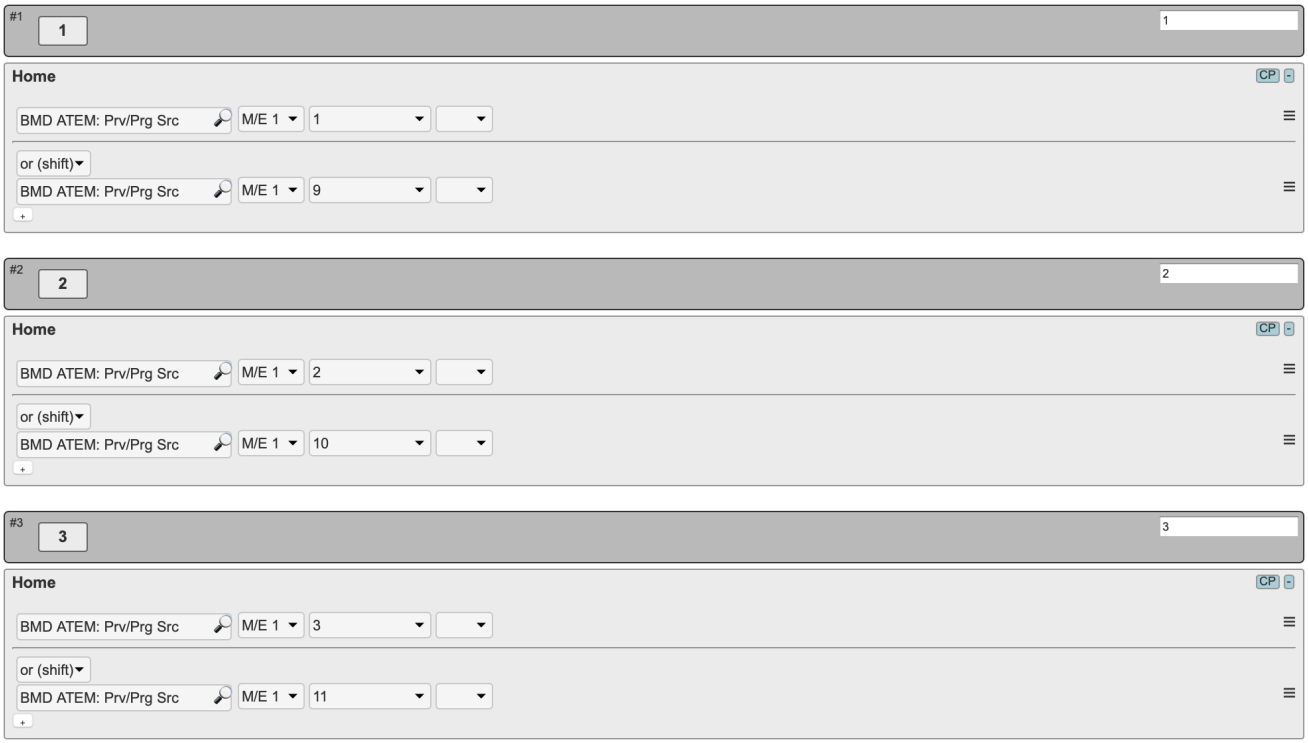
Another way to evaluate the differences in the light of what we have learned above would be to look at a well known UniSketch action, "ATEM Program Src":

UniSketch	Reactor
The action used in UniSketch has some fields: • First field is the device core + parameter • Next field is the ME Row, here it's 3 • Next field is the value we set when triggering this action: Setting the input source 10 on Program • Final field is the mode that we will apply: As we press repeatedly it will set the value 10 - or the previously detected value. Toggling. 	In Reactor the device core ("bmd-atem"), the parameter ("ProgramInputVideoSource") and the ME row (dimension of the parameter = "3") is all captured in the *Parameter* string. This is logical as they are all needed to identify what parameter we are specifically trying to manipulate. So that was like the first three fields of the UniSketch action in a single line. Of course this line is only partially human readable :-) so eventually there will be a browsing utility inside of Reactor that helps to pick this parameter with a UI (pending). The final modifier from UniSketch - the Toggle - is simply implemented through the choice of Master Behavior, in this case "SKAARHOJ:Toggle" 

Shift

UniSketch

UniSketch has a very simple and still powerful way of building navigation. There is a two dimensional matrix to organize actions in what is known as **Shift levels** and **States**. Here is how switcher input select actions for an ATEM switcher is organized:



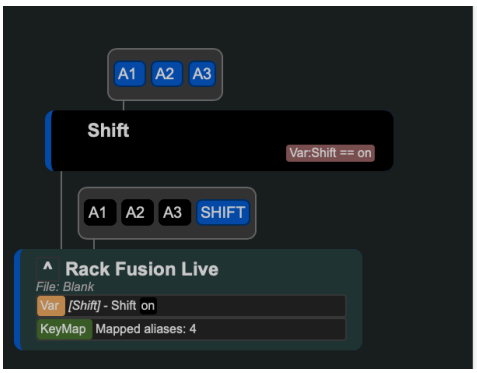
The horizontal line indicates which actions are active when Shift is off (the actions above the line) and which are active when Shift is on (below the line). Multiple actions can be executed in both situations and more Shift levels can be added.

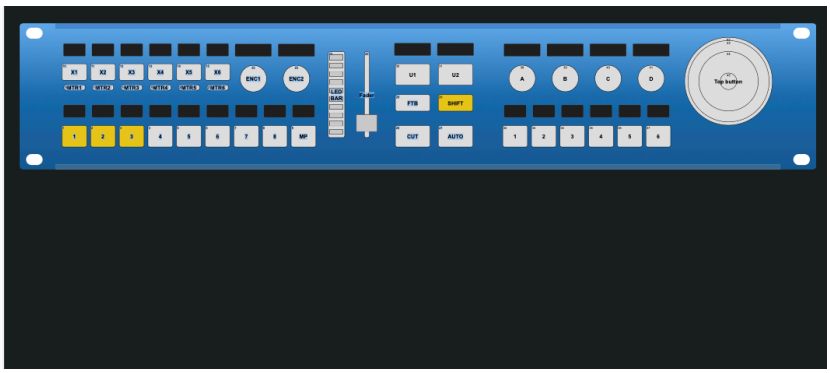
To create navigation for the Shift register, a specific action would be selected for that:

Shift is a global feature on the UniSketch panel, but using registers A-D it's possible to have essentially 5 different Shift registers in use on a panel. But this is where it gets much less intuitive.

Reactor

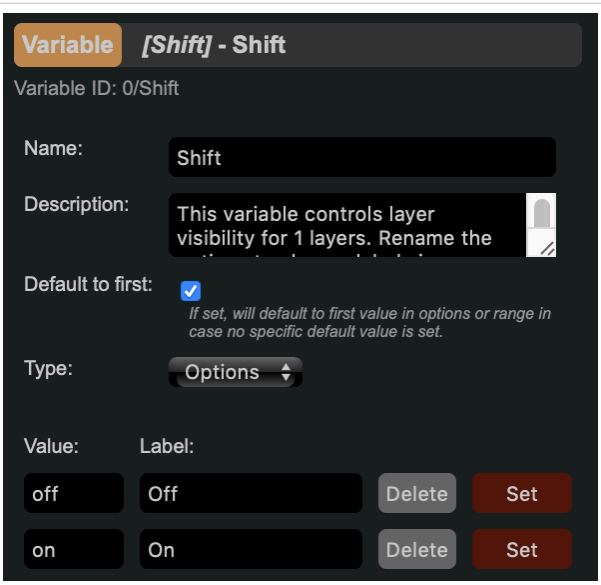
In Reactor this is how the same thing would look:

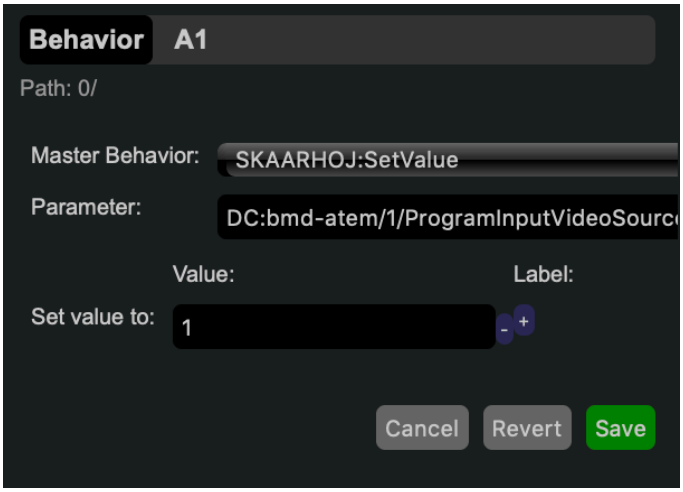




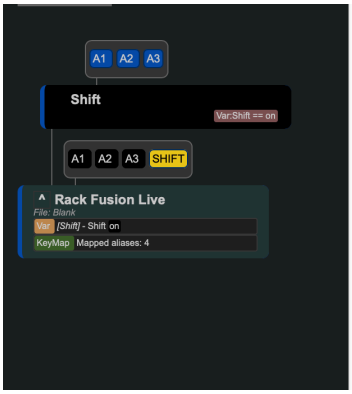
There is no fixed structure it is necessary to follow. Therefore it is possible to create three behaviors (A1, A2, A3) on the root layer and simply create another layer on top with the same behaviors, but with its visibility determined by the *variable* "Shift" having the value "on". The layer "Shift", the variable and its options and the condition to activate the Shift layer is created only when needed and as many times and places as needed. Reactor will help create these navigational structures quickly and with consistency according to conventions.

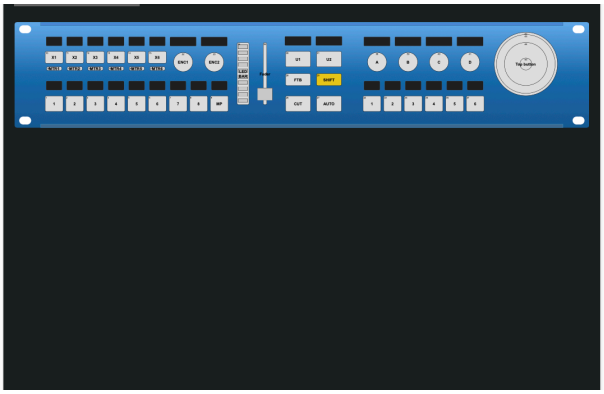
The variable would be created with two options, an "off" and "on" value. The behaviors A1, A2, A3 on the layers would simply point to a master behavior (SetValue), a parameter (Program Input Video Source on an ATEM switcher for example) and a value to be set when activated. See the images below for examples.

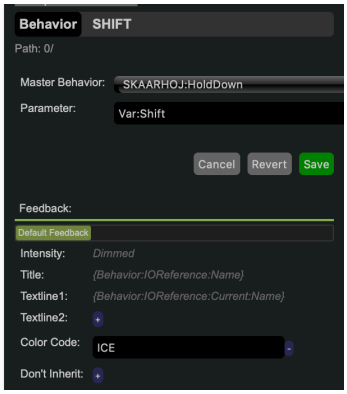




The Shift key would be defined with three simple settings: The master behavior "Hold Down", the parameter "Var:Shift" (referring to the variable that was created to drive the Shift layer visibility) and finally the ICE color inserted in the default feedback settings. See below images. Notice how painting a button with a given color in UniSketch was done by adding a "System: Local Color" action. In Reactor this is done far more intuitively as a true property of the behavior.



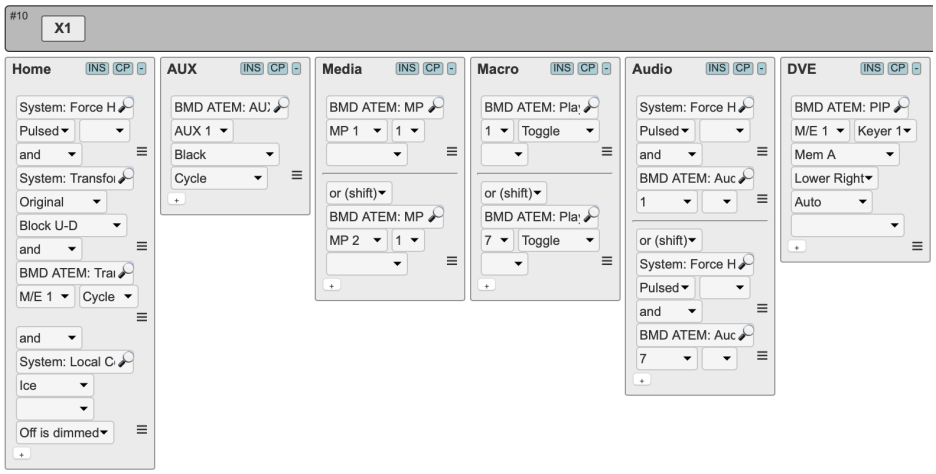




State

UniSketch

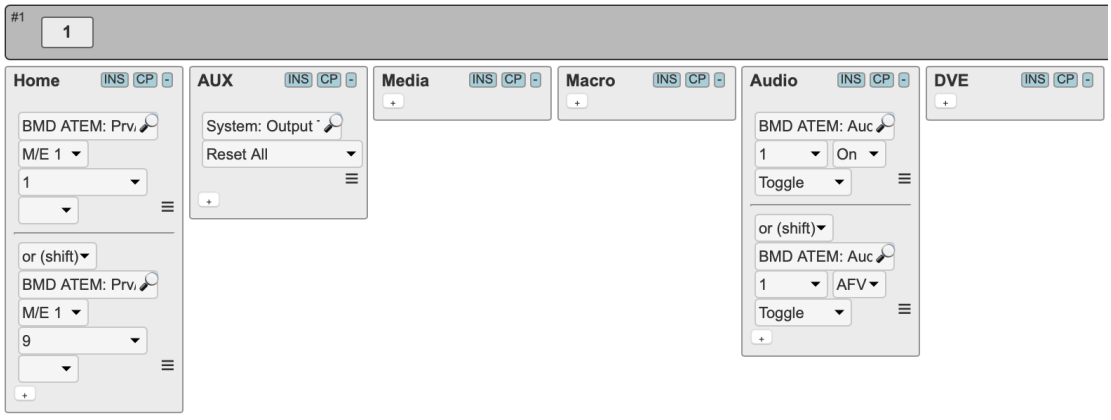
In UniSketch's matrix, Shift levels are managed inside the behaviors vertically. In the horizontal direction are the States as columns in which to specify behaviors for that State. It's actually a very straight forward idea, easy to use. Below is an example of a Rack Fusion Live with 6 states:



To change the state of the controller a button would be assigned with the action "System: State" and set its value to "State: 5" so it would cycle all states from 0-5 (6 in all). Shown below is the way UniSketch would convert a four-way button into a paging button with the left and right sides paging forth and back in the value.

In UniSketch the number of states for the controller would be fixed and therefore apply to all hardware components. This leaves empty behavior slots at places where no specific behavior is desired for the state:

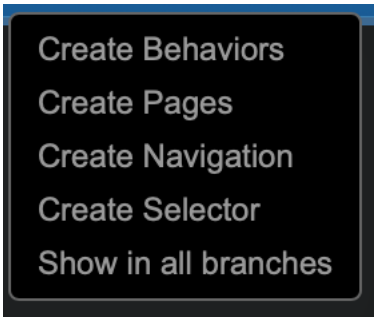
If a behavior slot is empty, the controller would fall back to the first state in all cases and a "System: No action" would be applied in cases where the user would prefer an inactive hardware component instead.



Reactor

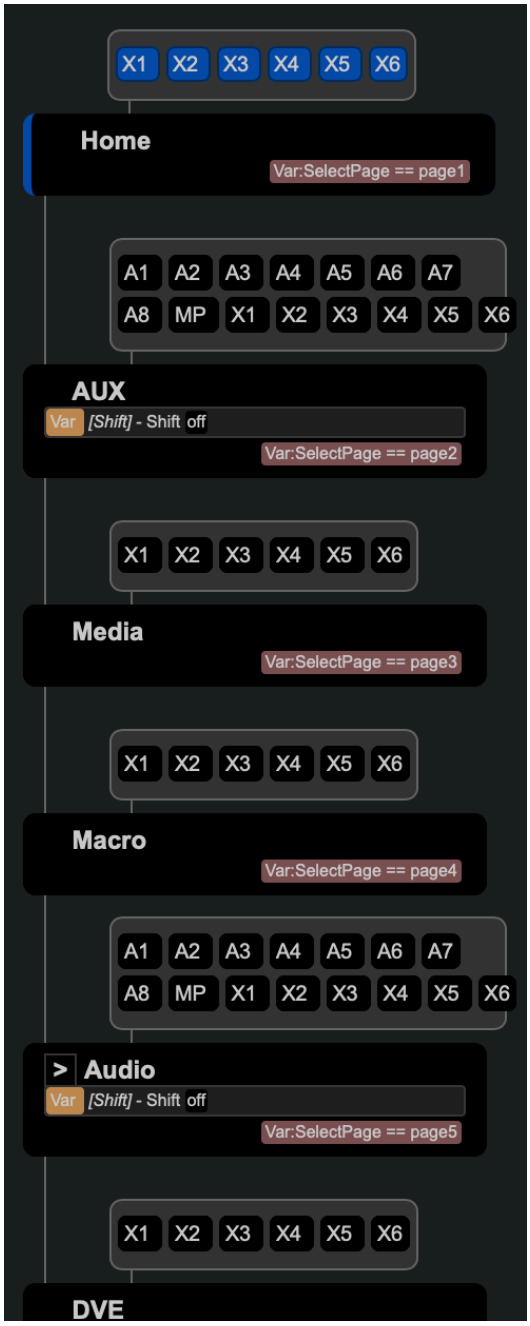
In Reactor states only exist in the same way Shift does by creating layers with conditional visibility. They are organizable in many ways.

The seven states from UniSketch would be implemented as six layers. This would usually be called “Pages”. This is the term used by the UI that facilitates creating such a structure:

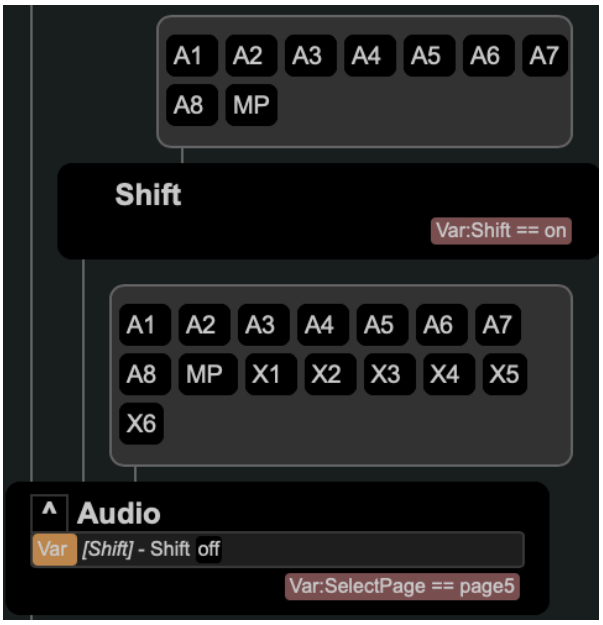


On each of the six layers we will find behaviors (X1-X6) for that particular “state”. To fall through to underlying behaviors, remove the behavior and the one on the parent layer will be active instead.

To have “states” affect more than X1-X6 add more behaviors for those hardware components to that layer (see layer AUX and Audio on the image to the right).

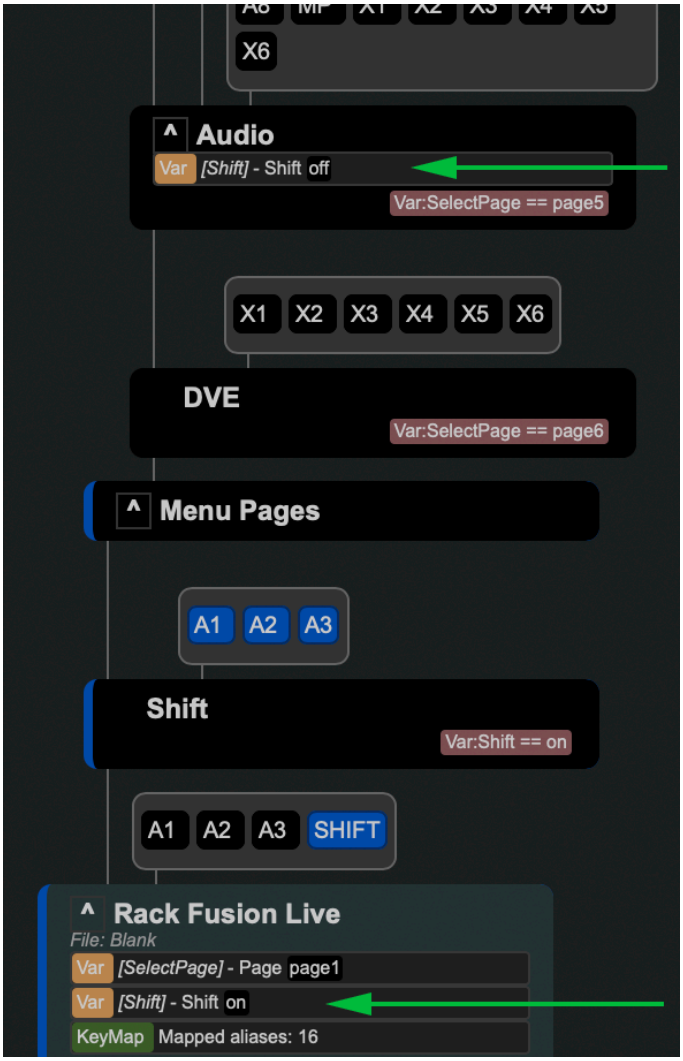


To have a Shift level inside create one inside, like it's done here for Audio.



When pages and shift levels are created a local variable is usually created to manage the structure. In this case the Shift variable is created locally on the Audio layer. This can be considered how Reactor implements Shift or State registers: By local variables in the context of the pages to be control.

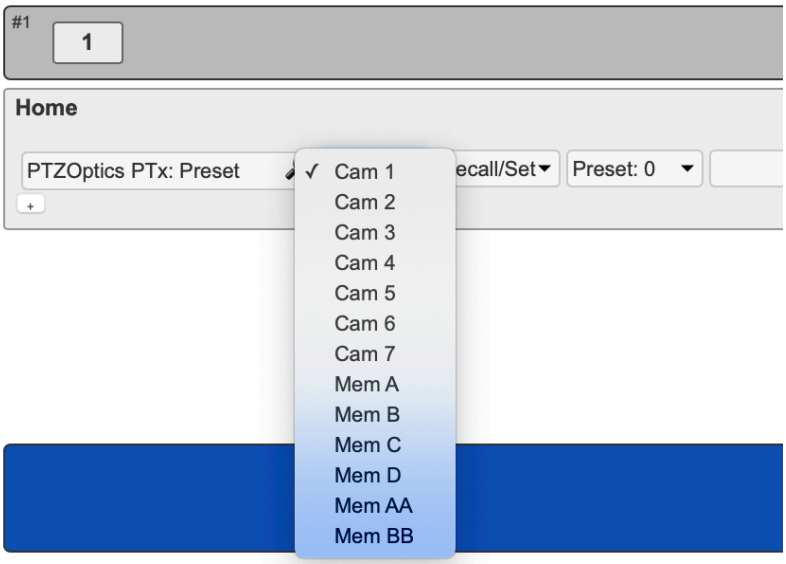
This has some awesome implications which includes the ease by which local or global shift levels can be made. For example, all it would take to make this locally created shift level for the Audio menu use the global shift level is to remove the local variable and it would fall through in the tree to the global variable by the same name. See the green arrows that points out the local Shift variable on top and the more global variable further down.



Memories

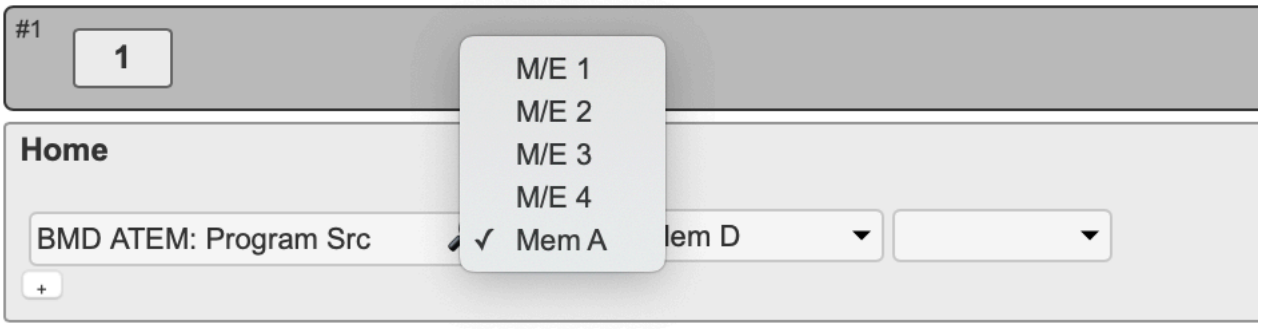
UniSketch

In UniSketch there are a number of memories named A-N which can contain simple integer values. These are used to select cameras. For example a PTZ camera action might use such a memory register to pick the camera it would send the pan and tilt commands to:



The camera selector on a panel would then set the relevant values to the memory registers. This would be one example:

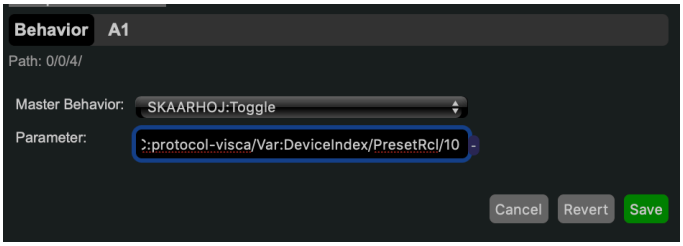
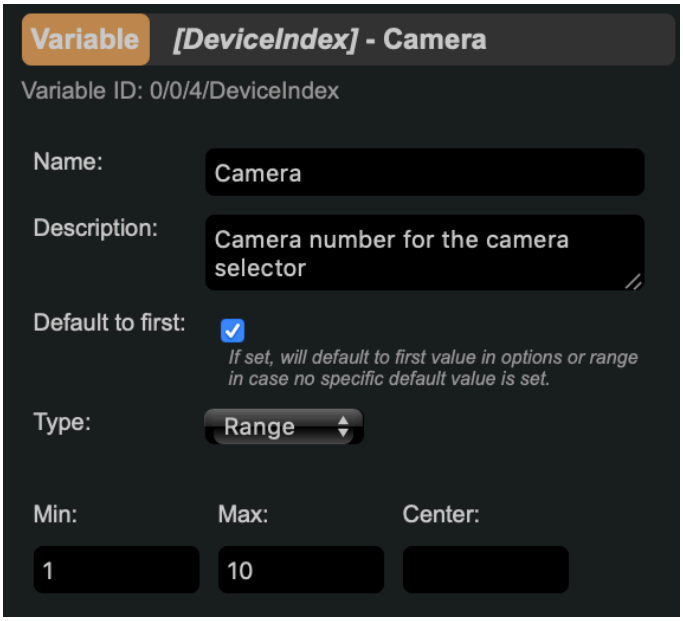
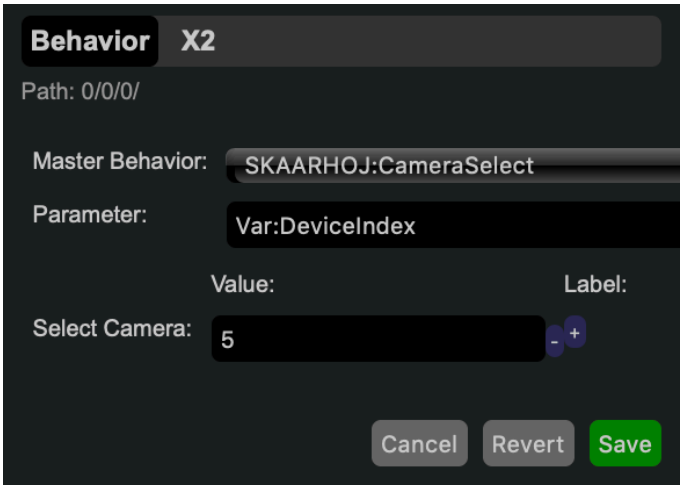
Another example would be letting the ME row and/or the input source of an ATEM switcher action depend on a memory:



Reactor

In Reactor there are no fixed memories, only any amount of variables created in the tree structure. It's a very important fact that variables are created somewhere in the tree: They are only valid and usable in the branch where they are created (with the exception of the effect from the Expand Scope feature). This means the same variable name can be defined in many parallel branches and this is the reason why included configurations will not interfere with each other.

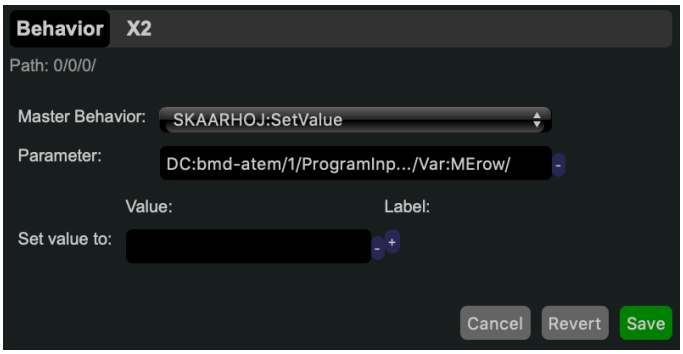
Furthermore, variables are in fact able to contain multiple values (like Mem AA and Mem BB in UniSketch), but let us leave that aside for now. Variables are natively strings, but can be used with integer values. Values should be alphanumeric with no spaces.

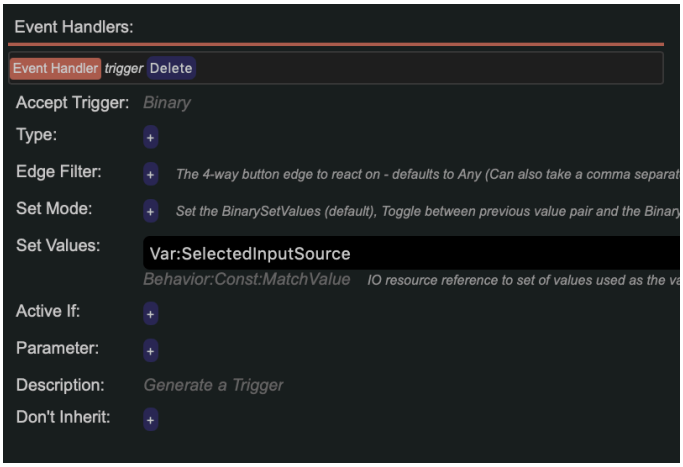
A camera selector would typically involve a variable often called DeviceIndex. This acts in the role of Mem AA in UniSketch to select the camera in place	
A variable, DeviceIndex would have to exist and could look like this for 10 devices (cameras)	
A basic camera selector would set the Device Index variable with the SKAARHOJ:CameraSelect master behavior. There is usually more used by a camera selected to have the buttons painted with tally color. This is why we use CameraSelect or CameraModelSelect instead of SetValue.	

If we were to set the ATEM Program source based on variables like in the UniSketch example, we would simply integrate the ME row in the Parameter string as a variable.

However, due to the special nature of this task (setting a source defined by a variable) there turns out to be no perfectly fitting master behavior. But we can still use SetValue. Just leave the "Set to value" blank and turn our eyes down to the "trigger" event handler: In studying the trigger we realize that the value that is set when we press the button is coming from "{Behavior:Const:MatchValue}" which is code for the value field we just left blank. All we need to do is overwrite this field with a reference to the variable, Var:SelectedInputSource (We are assuming we have created that for this purpose…).

Isn't that just awesome! :-) This is fully working already, but we are likely to expand the UI around it in the future to make it accessible for less tech savvy people too.

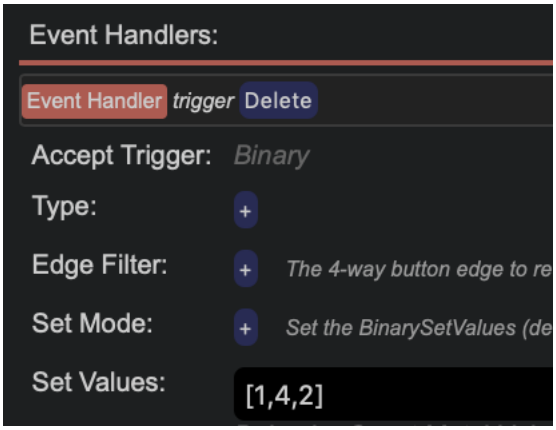


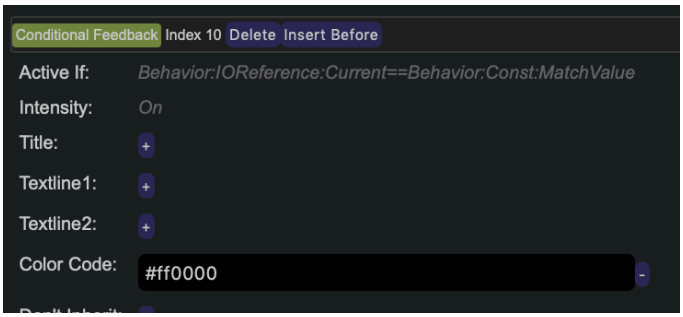
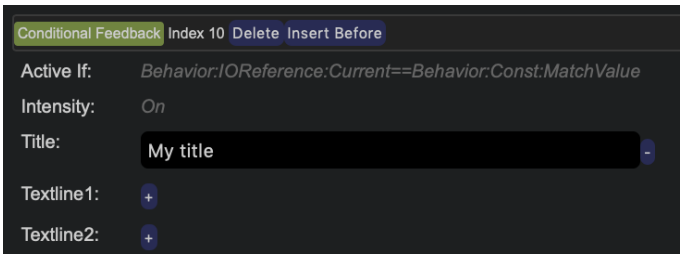


System Actions

Let us take look through the System Actions of UniSketch and comment on how each one of them are handled in Reactor. This could be a useful catalyst for comparing aspects of UniSketch and Reactor. First, here are the UniSketch System actions listed, then commented each one of them.

System: Shift Level System: State System: Memory System: Memory Group System: Memory Group Presets System: Cycle Memory System: Flag System: Flag Condition System: Camera Select System: Set Tally System: Auto Shift Level System: Local Shift Register System: Local State Register System: Local Color System: Local Graphic System: Local Label System: Local Display Color System: Local Display Font System: Transform 4-way Behaviour System: Force HWC Type	System: Tie to HWC# System: Range Limiter System: Value Scaler System: Custom Handler System: Synthesized Trigger System: Wait x/10s second System: Inactivate System: Stop connect System: Panel Brightness System: Panel Force Global Color System: Panel Sleep Time System: Alternative display System: Flash Light System: Web Config System: System Info System: IP Setup System: Live IP Change System: Device Core Enable System: Output Transformation System: Command System: No Action System: IB Mode
Shift Level	Fully implemented via layer visibility and variables in Reactor
State	Fully implemented via layer visibility and variables in Reactor
Memory	Fully implemented via variables in Reactor (pending nov 21: persistency of values between boots)
Memory Group	Fully inherent feature of any variable: It can store multiple values (UI support may not fully support that yet)

Cycle Memory	By default, a master behavior like StepChange will cycle through the values specified for any given parameter from end to end, but any arbitrary order can be specified as well by specifically setting that up as a multi-value IO Reference in the Set Values field of an Event handler. Example of how a sequence of value 1-4-2… is set up: 
Flag	Implemented via Flag Groups which is a tree feature. Flag groups operate with flags as Red, Green, Blue and White since flags often carry tally information. In many cases, a variable would be created to transport a single state around in the system.
Flag Condition	Flag Conditions would be fully substituted by the format of conditions in Reactor (a line like “Flag:Tally/Green/1 == true && Flag:Tally/Green/2 == true”…)
Camera Select	UniSketch’s attempt to create a generic camera selector action is fully substituted by master behaviors serving the similar job in Reactor
Set Tally	Not needed with user defined feedback in Reactor
Auto Shift Level	Setting a shift level by automation could be done by a virtual trigger, but more likely a fitting Active If condition would be written directly in the layer visibility. There is nothing blocking the driving of visibility of a layer with the value of a device parameter.
Local Shift Register	Not needed with the unlimited variables in Reactor
Local State Register	Not needed with the unlimited variables in Reactor

Local Color	Can be specified in both layers and on behaviors at any level in Reactor and inherited for clever application throughout the system. 
Local Graphic	Monochrome, Gray and Color icons and images can be added for graphical displays. Backend engine already exists, frontend UI for this is pending.
Local Label	Inherent part of any behavior in Reactor. Just type in the alternative label: 
Local Display Color	A part of the pending graphics UI
Local Display Font	A part of the pending graphics UI

Transform 4-way Behavior	In a sense, this is implemented in the event handlers of behaviors where trigger conversions are often used to convert to a “native” trigger type from any other incoming type. However, the way Blue Pill device core parameters are designed, they can generally accept any type of trigger anyway. For the curious mind, this is how a trigger conversion from pulsed, analog and speed triggers to binary could look like (coming from the master behavior Set Value): `"EventHandlers": { "trigger": { "Description": "Generate a Trigger", "AcceptTrigger": "Binary", "EventPreProc": { "P2B": { "InputPolarity": { "Default": { "OutputTrigger": "ActDown" } } }, "A2B": { "InputMapping": { "Default": { "Threshold": 500, "OutputTriggerRising": "ActDown", "OutputTriggerFalling": "ActDown" } } }, "S2B": { "InputPolarity": { "Negative": { "OutputTrigger": "ActDown" }, "Positive": { "OutputTrigger": "ActDown" } } } }, "BinarySetValues": { "Raw": "Behavior:Const:MatchValue" }, } },`
Force HWC Type	No need in Reactor. Event preprocessors are responsible for this. UniSketch has a hardcoded nature for how actions support different triggers and usually only a few most typical trigger types are implemented.
Tie to HWC#	Done through HWC Key Map mappings of HWC aliases to PanelID-HWC references
Range Limiter	Exists as a feature of event handlers
Value Scaler	Exists as a feature of event handlers
Custom Handler	For what it does in UniSketch (mostly, to bring the camera name to the RCPs display) it’s really not needed in Reactor.
Synthesized Trigger	Fully substituted by Virtual Triggers. See Virtual Triggers in Concepts of Reactor section.

Wait 1/x Second	Implemented in event handlers event preprocessor (pending) on a small scale. On a larger scale, one would look towards the Presets engine for actual timed macro executions.
Inactive	Exists natively (lock panel)
Stop Connect	N/A
Panel Brightness	Pending
Panel Force Global Color	Just set a color on the Default Feedback of the root layer…
Panel Sleep Time	Pending
Alternative Display	N/A
Flash Light	Exists, but undocumented so far.
Web Config	Always on :-)
System Info	N/A
IP Setup	N/A
Live IP Change	N/A
Device Core Enable	N/A
Output Transformation	With reactors separated feedback and event handling, this work around from UniSketch is not needed.
Command	N/A
No Action	Just insert behavior with no content and it blanks out the component
BP Mode	N/A

Simulation

Reactor has a built in simulator too. There are two ways to do simulations in Blue Pill, through the Simulator page and in the Configuration page.

There are some limitations to simulations as multiple buttons are not able to be press simultaneously. It is not meant to be a replacement for a hardware panel but as a helpful tool.

Simulator

On the Simulator page, an interactive graphical display of each panel is present. The current mapping of each controller is visible. It is possible to completely simulate both the panel and the device from a stand alone Blue Pill to get an idea of what that mapping is like and what actions are available.

What is seen in the simulator will be reflected on any attached panels and will control any attached devices. Devices will be effected even if the panel is being fully simulated and is not physically present.

Simulation via Configuration Page

In the Configuration page, it is possible to toggle between simulation and configuration modes to test the work in progress. While in simulation mode, it is not possible to add to the configuration on the Configuration page.

Simulation Navigation

The simulators display the panels as they are arranged in the Graphical View of each panel group (see Panels sections for more information) and per group.

It is possible to limit which panels are seen in the simulation by selecting or deselecting the name op the panel at the top of the page.

Movement:

- Alt (Options) + mouse drag to navigate up, down, left and right
- Alt (Options) + mouse scroll -> adjust zoom level
- Click to interact with buttons and press down encoders (will effect connected devices)
- Mouse drag to move joysticks, faders and turn encoders (will effect connected devices)
- Scroll to turn encoders and joystick rings (will effect connected devices)

Packages

Blue Pill being a Linux system has a number of system settings and a package management interface.

Important Packages

The following Packages are needed for Blue Pill operation. The ibeam-proxy core is only needed in specific situations.

STATUS	PACKAGE NAME	PACKAGE DESCRIPTION	TAG	VERSIONS
Running	ibeam-hardware	Hardware Abstraction Layer for skaarOS		v0.0.9-pre8
Running	ibeam-init	SKAARHOJ skaarOS init system		v0.1.26-pre1
Running	ibeam-proxy	IBeam core connection manager for skaarOS		v0.2.6
Running	reactor	BLUEPILL Reactor, connecting hardware and device cores		v1.0.1-pre1

ibeam-hardware (“ibeam” is the early development name for Blue Pill) which is the application that facilitates access to the hardware features of the Blue Pill. On an RCP Pro with Blue Pill Inside the ibeam-hardware application creates the Raw Panel based access to the hardware components of the RCP Pro (so even when Blue Pill runs natively on a controller like RCP Pro, it is still using a network socket to talk to the hardware on the panel, so in that way it is no different from connecting to external panels!).

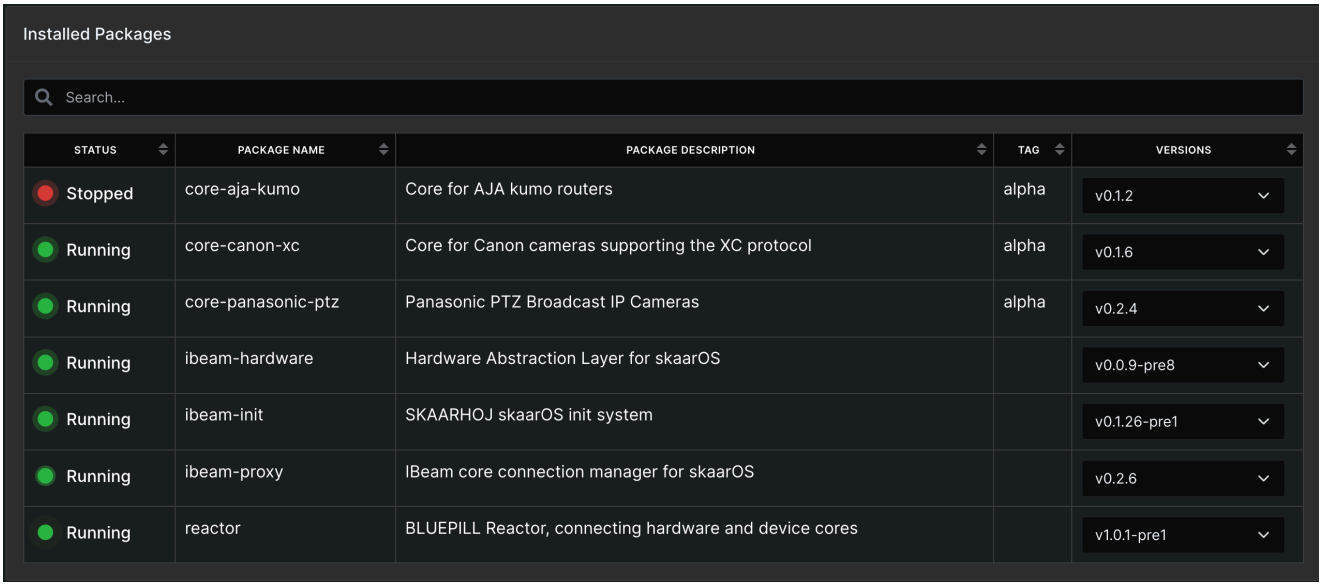
ibeam-init is the System management UI for skaarOS or what is on the screenshot above.

ibeam-proxy is a bridge used on this particular Blue Pill that will make the device cores on it available to the outer world. This is used when for an architecture where separate Blue Pills run the device cores locally to load balance or manage network latency - or in the simple case of Blue Pills with SKAARHOJs series of extension cables which bridges connectivity to legacy serial systems (RS-422, RS-485, RS-232 and analog voltages for lens control etc.). In those cases, having a device core run locally on that Blue Pill is not only necessary because it needs to talk directly to UARTs, but it also means that single-master serial devices (and, this is what most of them are…) are available for multi-master access via the device core TCP interface.

Reactor is what most will generally think of as the Blue Pill. This is the platform where panels, devices, and configurations are added.

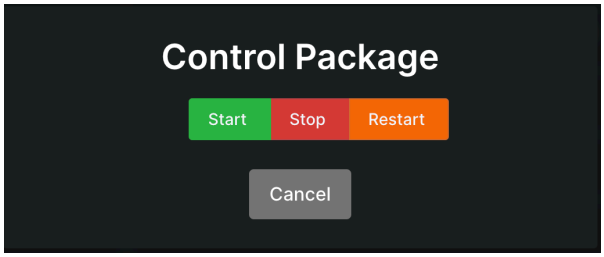
Installed Packages

The overview page shows which packages are currently installed and in which version and their status; whether they are running or not. In this screenshot the Blue Pill has device cores for AJA Kumo, Canon XC Protocol, and Panasonic PTZ cameras. They are all running except the AJA Kumo. An installed core will generally stop running when there are no devices that use it currently in the devices section. This is designed to save processing power while allowing for packages to be loaded onto the Blue Pill for later use.



Stopping and Restarting Cores

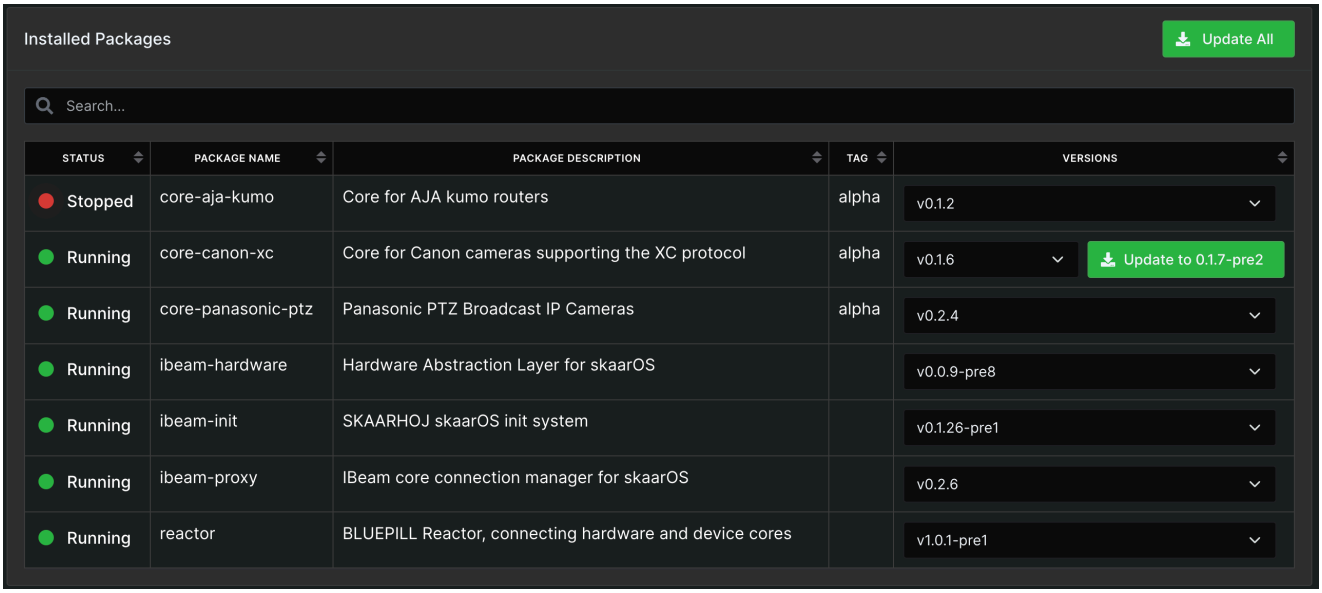
It may sometimes be necessary to manually start, stop, or restart a package. To do so, click on the status of the individual core in the status column.



Updating Packages

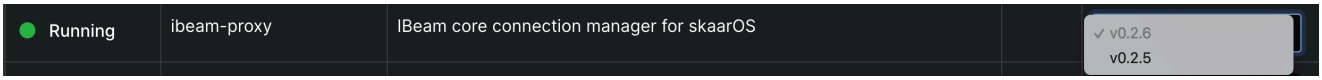
Updating packages or reverting to older versions of packages is simple.

Packages with updates available with updates available with have a green UPDATE TO X.X.X button next to the current version. Pressing the update button will update that individual package. Pressing the Update All button on the bar with Installed Packages at the top, will run all available package updates.



Please note, prerelease updates are only available when the Blue Pill is in Advanced Mode set on the System page.

To revert back to a previous version of a package click on the drop down in the Versions column and select the desired version.

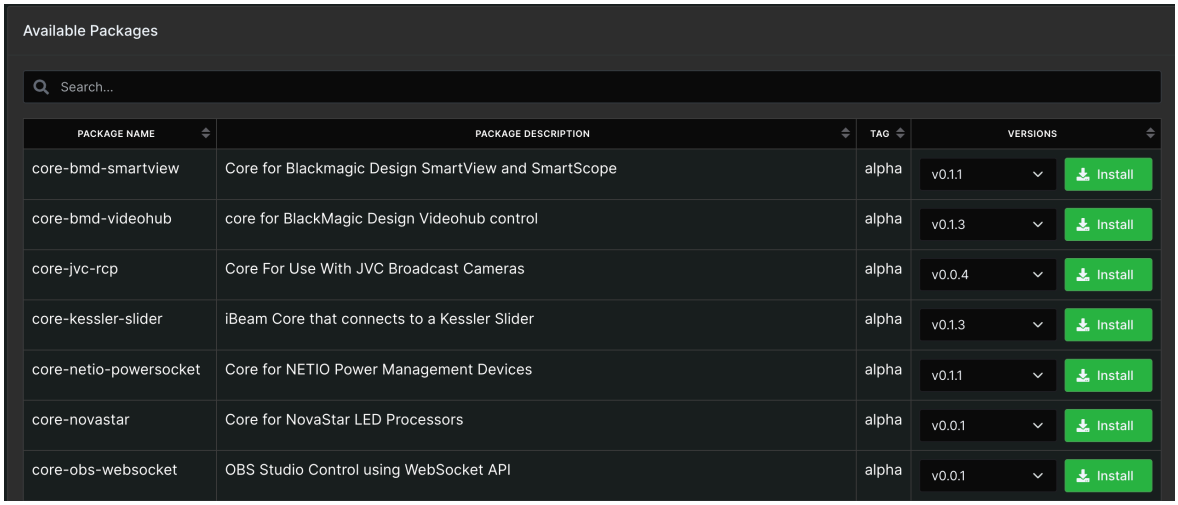


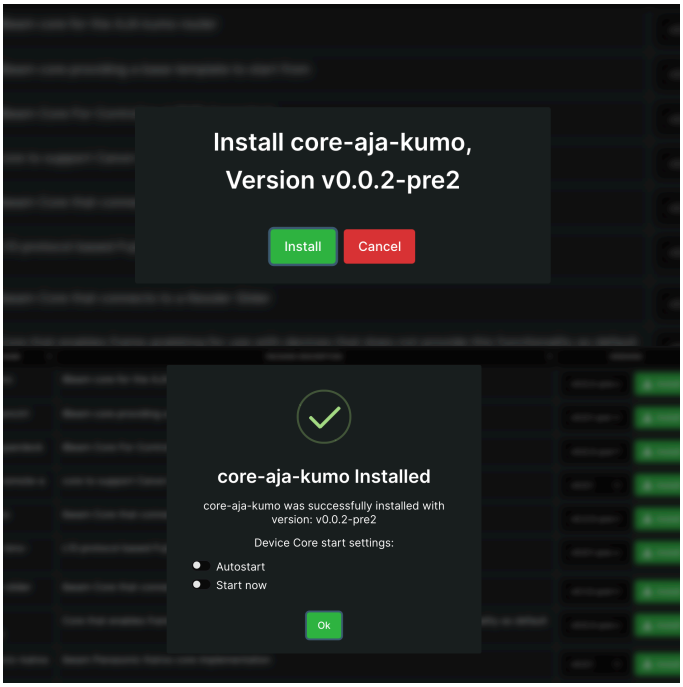
Available Packages

Not much different from the list of installed packages, are the available packages.

Installing these requires internet access as they are pulled from an online storage with applications we have published. It's really simple. When connected to the internet the needed package for a device core will generally auto install when a device associated with it is selected via the devices section of the Home screen.

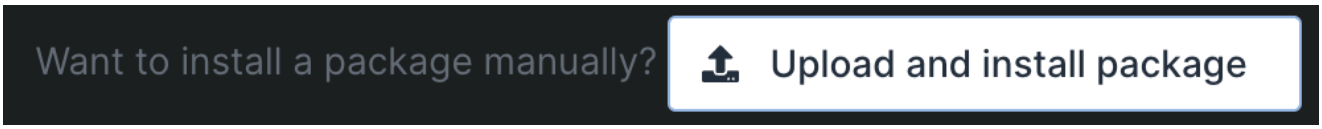
Packages can also be installed manually by pressing the green Install button.





Please note, prerelease updates are only available when the Blue Pill is in Advanced Mode set in the System page.

On rare occasion it may be necessary to upload and install a package manually from a file provided by the SKAARHOJ support or development teams. This would be done at the bottom of the Packages page.



Maturity Levels

Between the package description and package versions columns is the package maturity level. As Blue Pill is very new most of our packages are still in early stages of development or are still waiting feedback to increase the maturity level. As time goes on more of our packages will be fully mature.

STATUS	PACKAGE NAME	PACKAGE DESCRIPTION	TAG	VERSIONS
Stopped	core-aja-kumo	Core for AJA kumo routers	alpha	v0.1.2
Running	core-canon-xc	Core for Canon cameras supporting the XC protocol	alpha	v0.1.7-pre2
Running	core-panasonic-ptz	Panasonic PTZ Broadcast IP Cameras	alpha	v0.2.4

Sandbox, Early development

- Everything is allowed to change
- Core is not guaranteed to work
- Core is not guaranteed to be finalized
- Only Pre Release, not enabled in Reactor yet

ALPHA

- Core connect without issues
- Core is tested partly by the Developer
- There is not necessary a configuration yet
- The core has a description, category, and correct name
- Parameter names can changed if needed
- Only PreRelease but enabled in Reactor

BETA

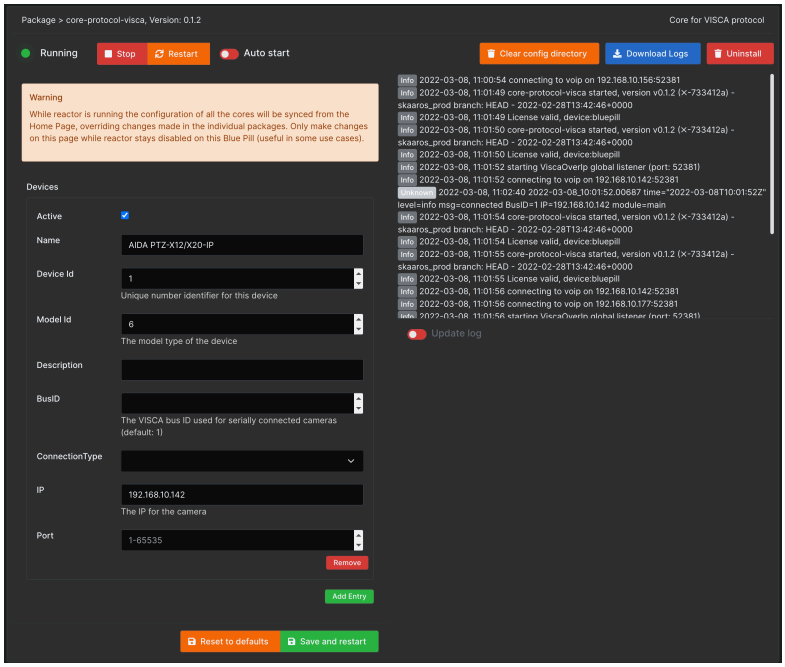
- R&D has implemented everything needed for stable release
- Device core is QC tested
- There is at least one Configuration for the Device Core
- The core has been proofread, Parameter names can not be changed any more
- The core has graphics
- Device test is implemented
- **The core is in Stable Release**

MATURE

- No bad customer feedback who give us a reason for rework
- At Least 4 weeks since state Beta was set

Package and Device Core Settings

Clicking an installed package brings opens the settings page of the package.



From the Package/Core Settings page it is possible to see the current status, stop or restart, and set Auto start. This can be a helpful tool if the core is not acting properly.

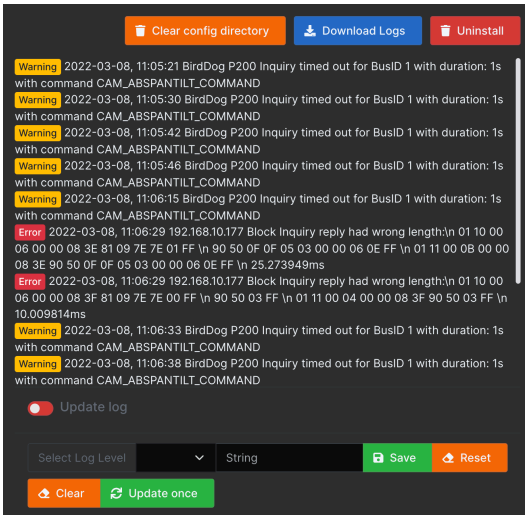
All connected devices to the package are visible, along with their current settings. While it is possible to update them from this page, **it is advised to add and adjust them from the Home page, as the data is synced from there.**

On the the left the package logs can be seen and downloaded. This can be used for troubleshooting and may be requested by the support or development team for diagnostics. By default the the logs do not auto update after initial connection but by toggling on Update Log, the latest information can be found.

Clear Config Directly will delete everything that was saved onto the Blue Pill by that specific package and return it to its default settings.

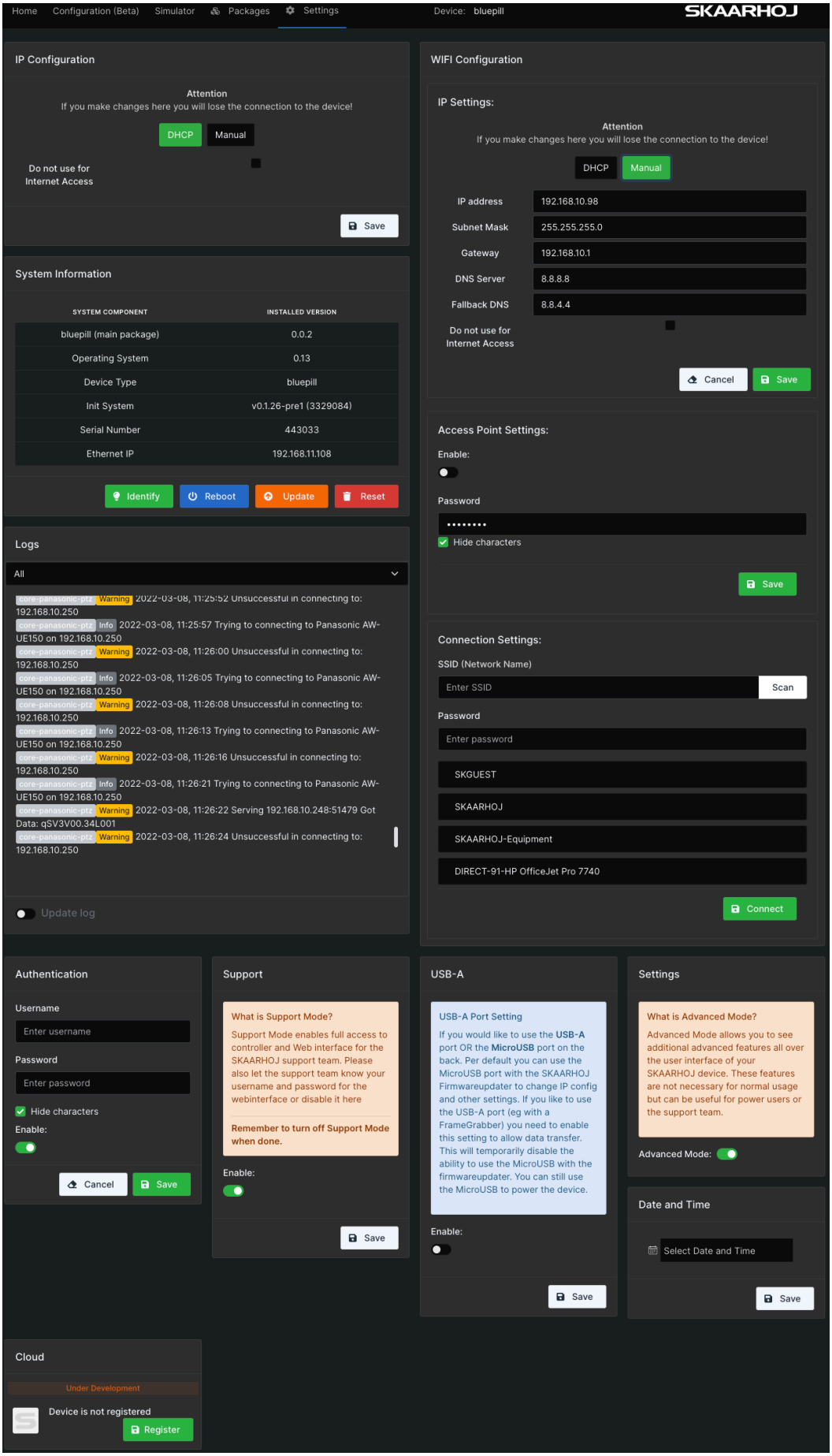
It is also possible to completely Uninstall a package from this page. Doing so will not remove previously connected devices, but will cause them to be disconnected.

When setting the Blue Pill into advanced mode, filtering options become available for the core logs.



Settings

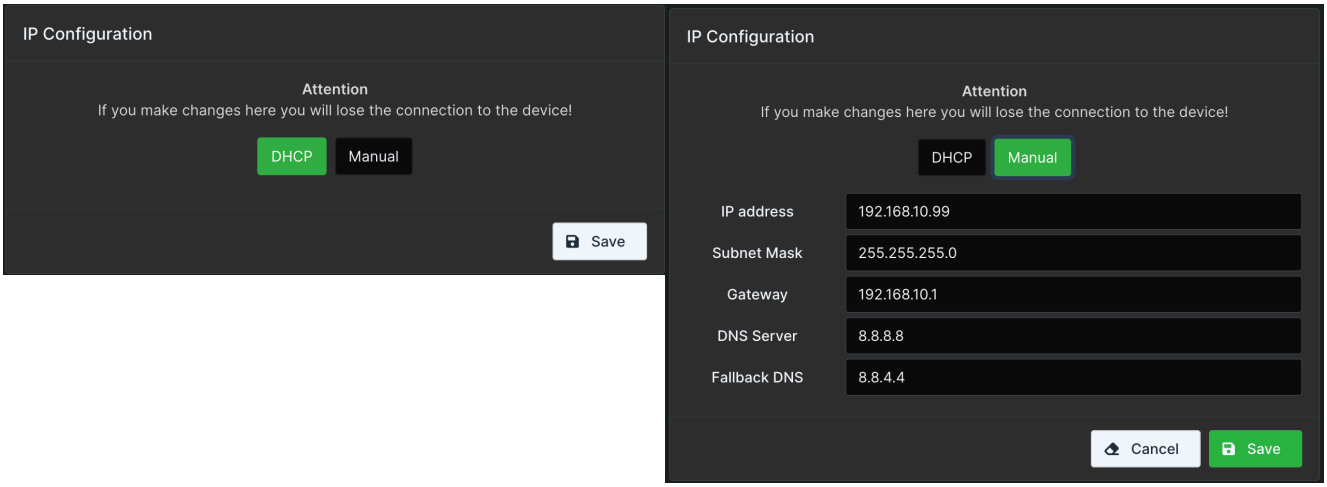
On the System screen is additional system related information.



IP Configuration

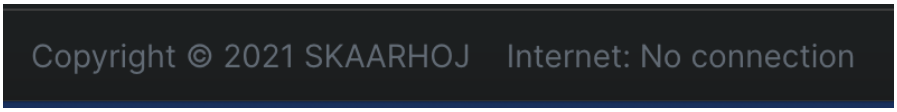
There are two options for setting the IP configuration of the Blue Pill. By default it is set to DHCP. To switch to manual IP Configuration, select Manual, enter the desired IP network information, and press save.

Please note, changing the IP information will cause the web interface to lose connection as the IP address has been re-assigned.



In Advanced Mode, the option exists to have the device not connect to the Internet. Please note, internet access is needed to install, update, or revert packages.

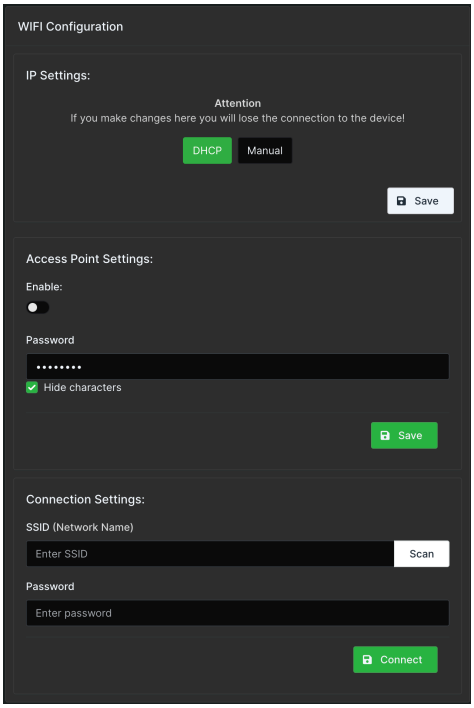
What Internet access has been restricted the message Internet: No connection will appear at the bottom of the page next to our Copyright.



WiFi Configuration

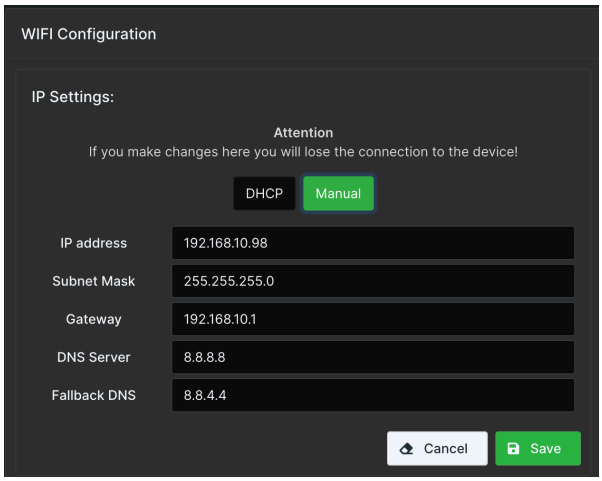
There are multiple ways the Blue Pill can interact with WiFi. There is the IP Settings to use the Blue Pill as a WiFi Hot Spot, the Access Point to enable a local access point to connect to the Blue Pill, and Connection Settings to have the Blue Pill connect to a local WiFi network.

WiFi is not intended as the main way to use the Blue Pill. Blue Pill is encapsulated in metal and the reach of the WiFi antenna will be very short, so consider it for very localized access, but sufficiently useful to connect to the WiFi of a camera just next to. Or to enable the WiFi access point for mobile device management.



IP Settings

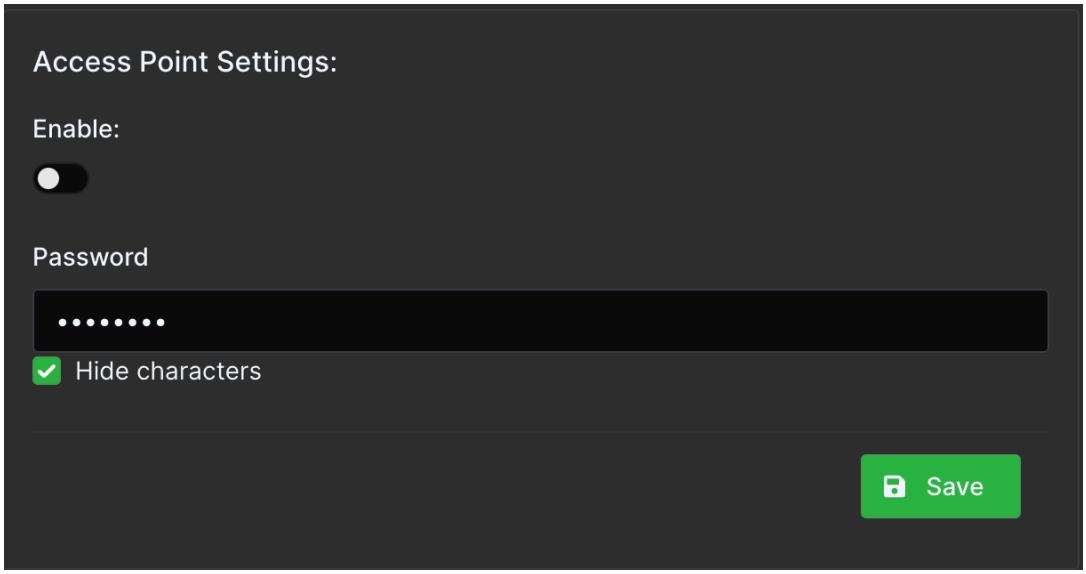
Used to connect to the Blue Pill via WiFi. This would make sense in some some use cases such as using the Blue Pill with extension cables or when connecting multiple Blue Pills. This can be set to DHCP or Manual.



In Advanced Mode, access to the internet can be restricted. Please note, internet access is needed to install, update, or revert packages.

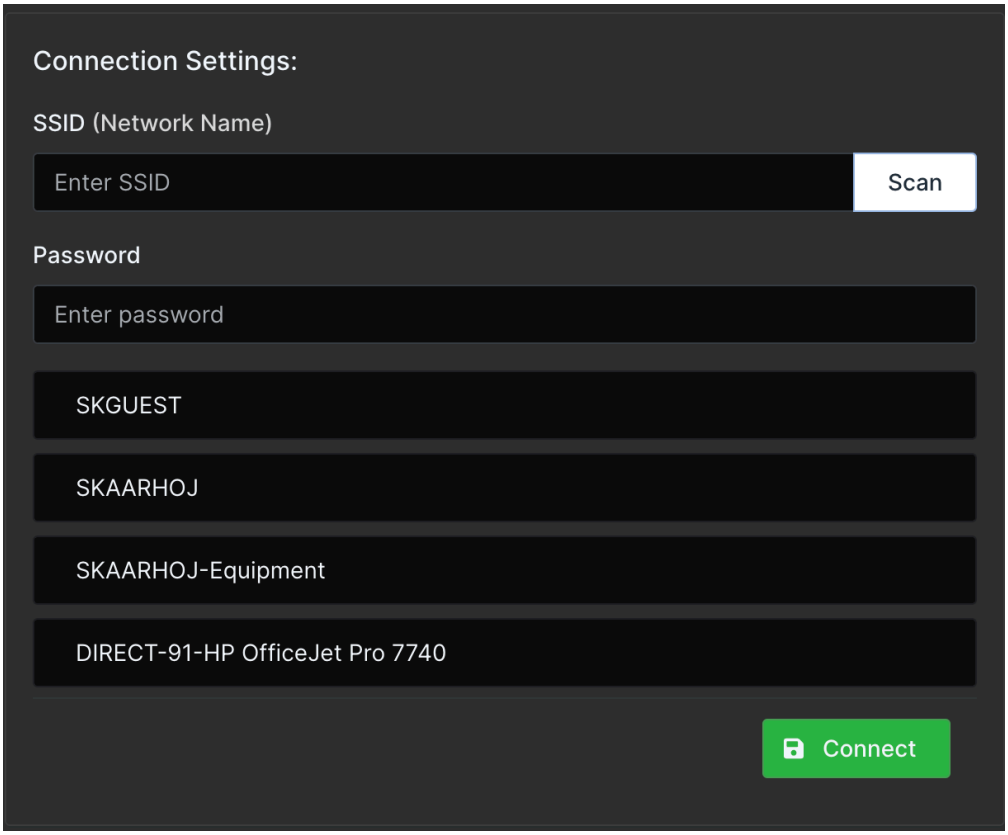
Access Point Settings

Used to access the Blue Pill via an internal WiFi to establish a connection and set the needed network information for regular connection. In this section the Access Point can be enabled/disabled and the password set. By default the password is **skaarhoj**. See the section on Connectivity for more information.



Connection Setting

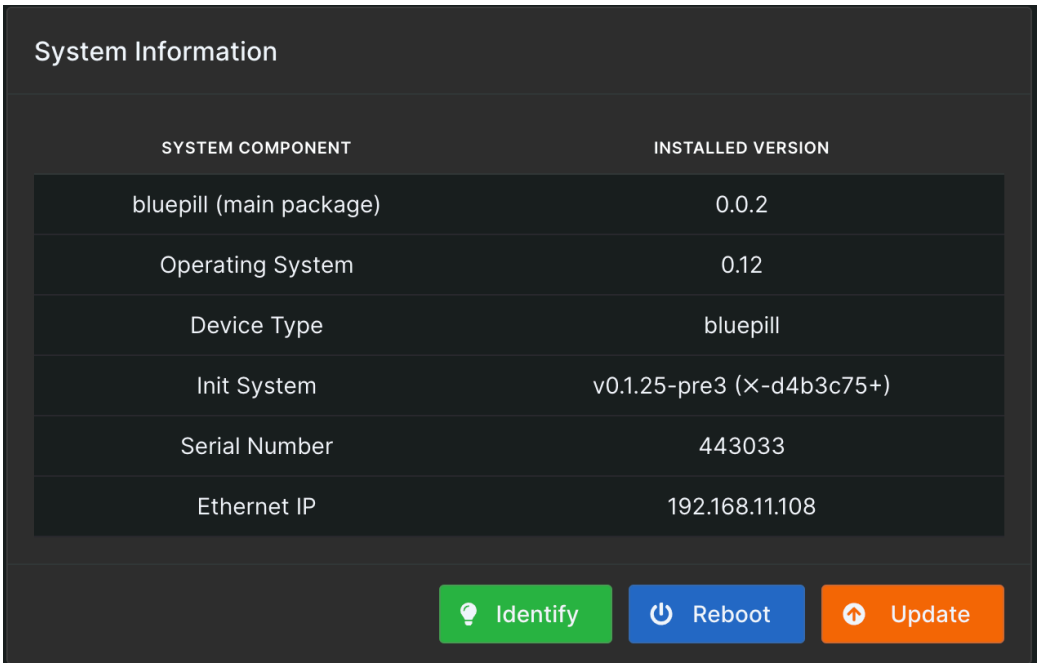
Used to connect the Blue Pill to the local WiFi network. Pressing Scan will search for available networks. Clicking a network that appears after a scan will allow for the entering of the needed password to establish the connection.



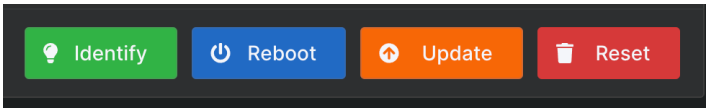
Please note connecting the Blue Pill to both the wired and wireless networks for internet access can cause instability in the connection. To have both active, please enable Advanced Mode to restrict one of the connection to No Internet Access.

System Information

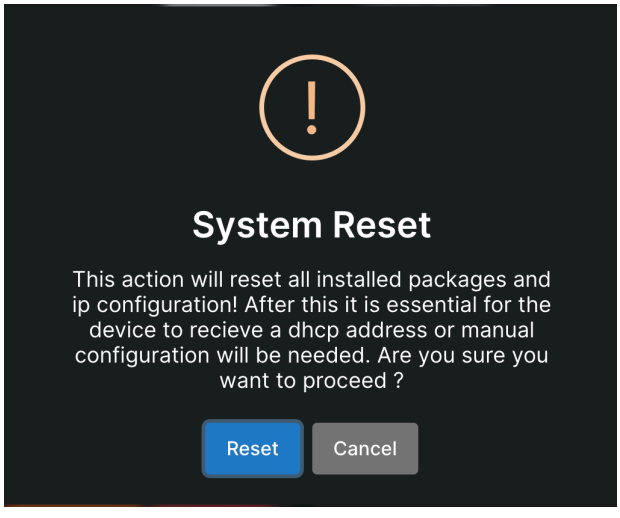
The overall System Information is displayed in this section including the Installed Version number for the main components, the serial number, and the current IP information. There is the option to Identify the unit, by having the small led next to the ethernet port light red for 30 seconds, reboot the unit, or update the unit. When updating the option exists to use the next stable version or it is possible to test prerelease versions.



In Advanced Mode there is the option to perform a hard reset of the Blue Pill.

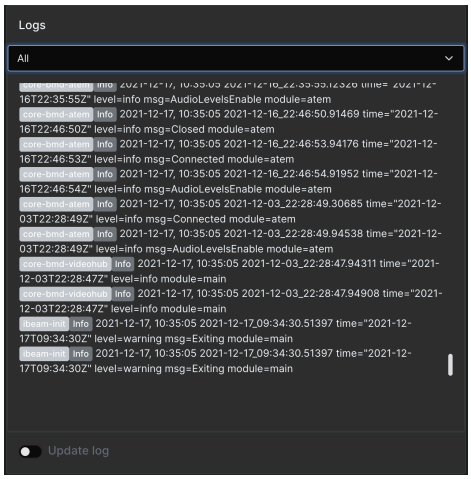


Please note, doing a Reset will erase all installed packages, IP configuration, and any projects. This will return the Blue Pill to factory settings. It is not possible to recover any lost information after a Reset which is why it is hidden in Advanced Mode.



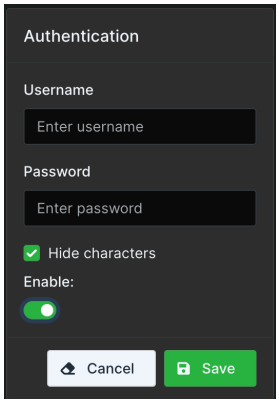
Logs

The logs are used mainly by the development team to help debug any problems. Each device core has its owns logs section but it is can all be seen in the systems logs section. Clicking the drop down will allow for the filtering of specific device cores/packages.



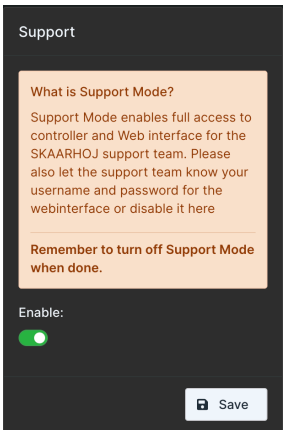
Authentication

Authentication allows for the setting of a username and password to restrict access to the Blue Pill web interface. By default the Username is **admin** and the password is **skaarhoj** This feature can also be disabled.



Support

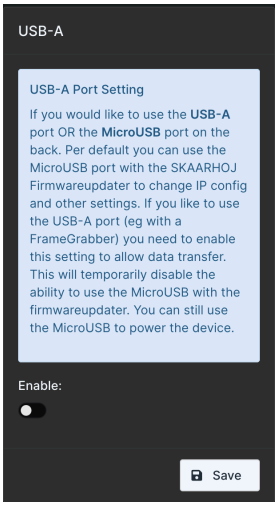
Enabling Support mode will allow the SKAARHOJ support team to remotely access the Blue Pill. By default this is disabled.



Please note, internet access needs to be enabled for support to access the Blue Pill.

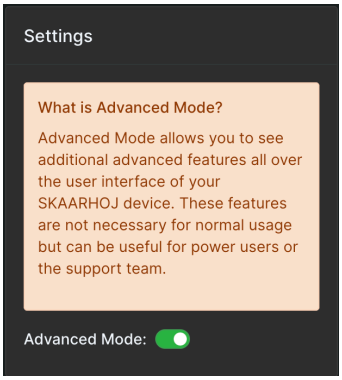
USB-A

USB-A switches between the Micro USB port (on by default for service such as connection to the SKAARHOJ firmware application or re-flashing the entire unit in severe cases) and the USB-A port which is used for example by our FrameLink application to grab and utilize frames for thumbnail display in panels. See the Extensions section for more information about the FrameLink application.



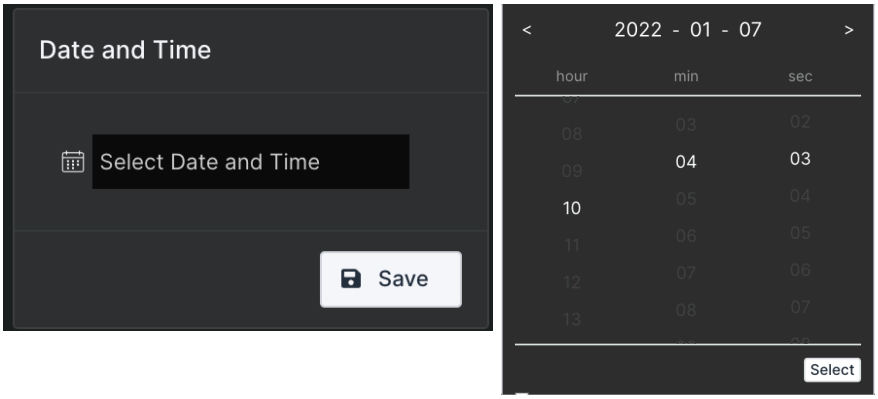
Advanced Mode

Advanced Mode will open up a few more UI features which are considered more advanced and therefore hidden away in the standard case. Advanced Mode is tied to the browser and not saved in the Blue Pill. So if it is open on Chrome, Fire Fox, and Safari, they can all have different level of options available.



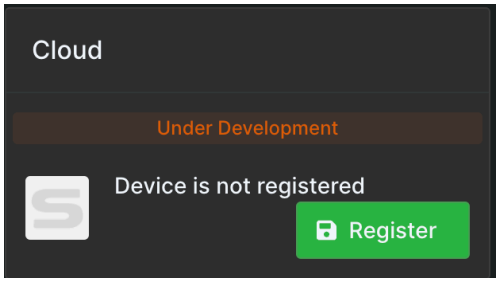
Date and Time

Date and Time allows for setting the current date and time on the Blue Pill.



Cloud

Coming Soon.



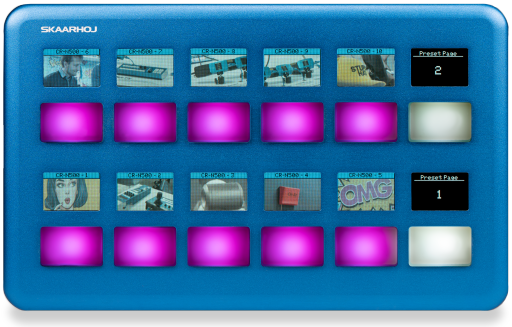
Blue Pill Extensions

Frame Link

The Skaarhoj Frame Link package combined with an HDMI Video Capture card allows for thumbnail presets images on compatible SKAARHOJ panels like the Frame Shot from any HDMI source.

At time of writing we have only tested the HDMI Video Capture stick below so would not be able to say how other models perform.

Please note, some PTZ camera models natively support thumbnail presets like the Canon CR-N500. The



FrameLink package and the HDMI Video Capture stick are not needed for those models of cameras. This is only for cameras or other devices that do not natively support this feature.

The breakdown of the 21 FrameLink Window options is seen below. The incoming HDMI feed can be broken up into either full picture, a 2x2 grid, or a 4x4 grid.

1

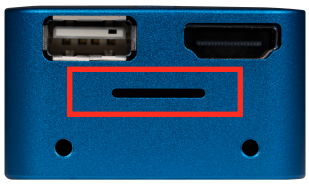
2	3
4	5

6	7	8	9
10	11	12	13
14	15	16	17
18	19	20	21

Entering the corresponding squares number into the FrameLink Window section of the Camera Selector's Constant Set will use that portion of the image as the thumbnail for the preset. In general the grids are for use with a multiviewer.

Extension Cables

The Blue Pill will have a number of possible extension cable options. These will be used to connect to devices that may not have native IP capability like some lenses or cameras using RS-242 or RS-244. This will use the extension cable slot on the side of the Blue Pill.



These possibilities are still in development with a list being made available as soon as they begin to be rolled out.

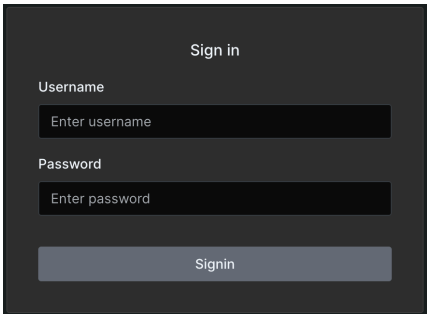
Connectivity

Accessing Blue Pill

DHCP or Static IP

The Blue Pill's user interface is accessed via the device's IP Address and any web browser. The IP address can be found on the display after it is plugged into a network connection with PoE or a network connection and a power supply (5V Micro USB).

Entering the IP address into the address bar of a search engine will bring up a prompt for username and password. The default is username: **admin** password: **skaarhoj**



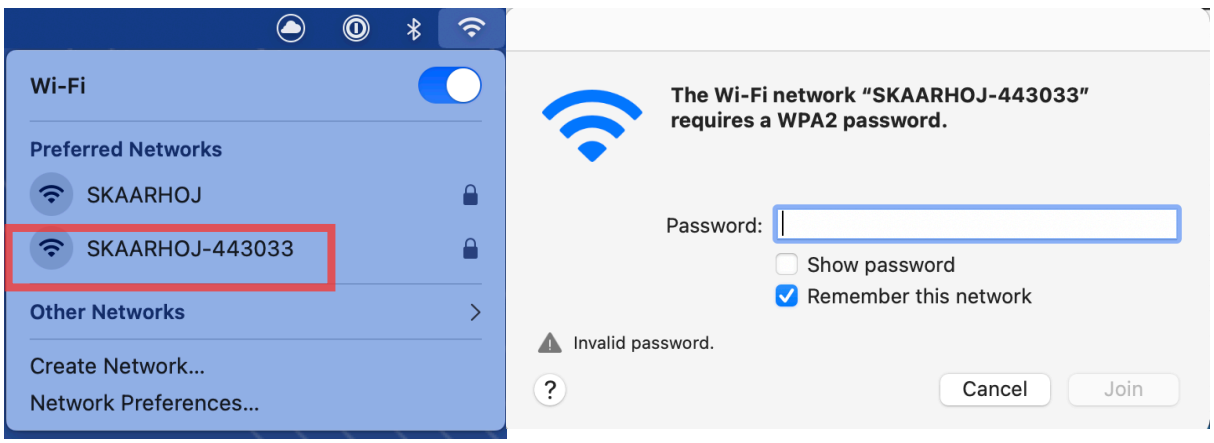
Wi-Fi Access Point

If the Blue Pill is not displaying an IP address, the web interface is accessible by enabling the internal Wi-Fi access point.

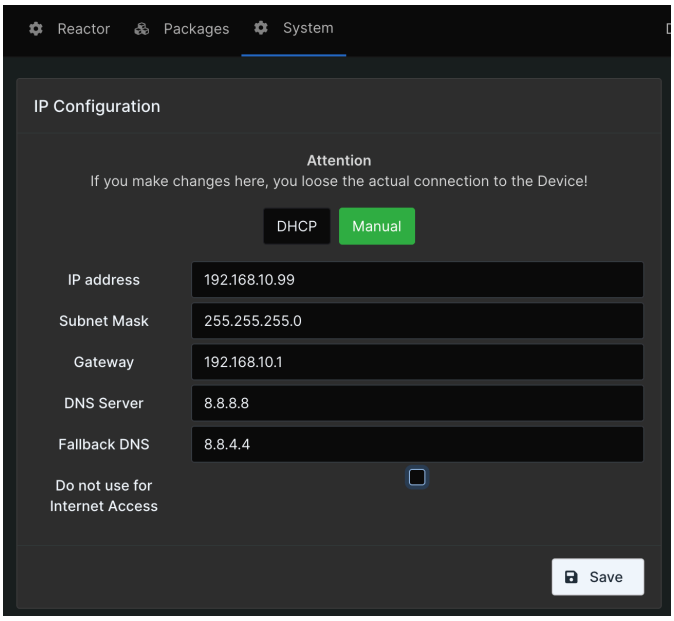
To enable the Wi-Fi access point, gently press and hold the config button on the side of the Blue Pill for about 3 seconds (Found next to the Ethernet jack. Use a flat screw driver, a paper clip or similar). When released, it will enable the internal Access Point and the LED will light up purple. It will show up in the Wi-Fi networks list as SKAARHOJ-XXXXXX (Blue Pill's serial number).

The default password is: **skaarhoj**

The web interface is then accessed at the IP address: **192.168.4.1**

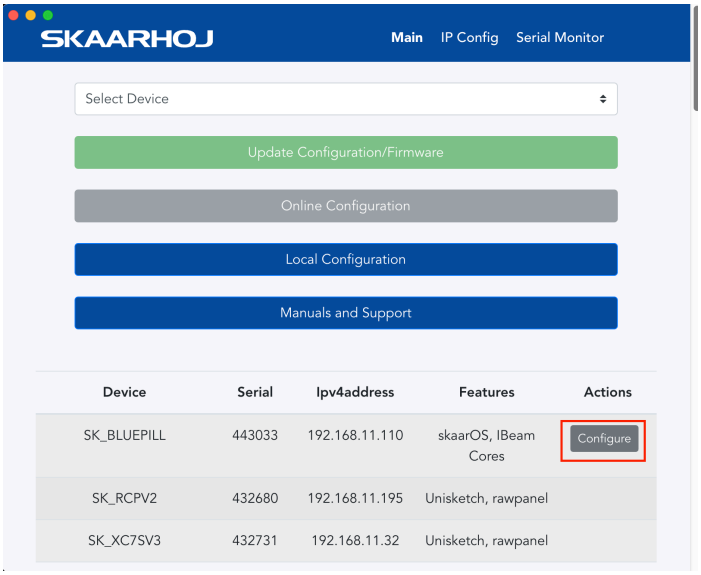


After accessing the Blue Pill it is best to navigate to the System Menu/System page to set a static IP address. Once saved, the new IP address will appear on the Blue Pill's display, it may be necessary to reboot or power cycle the device afterwards.



Link from SKAARHOJ Firmware Updater

If the SKAARHOJ Firmware Updater open on a computer running on the same subnet as the Blue Pill, the Blue Pill should appear below the main controller access buttons of the updater. Clicking on Configure next to the panel's information will open the web interface directly. The Blue Pill does not need to be connected to the computer via USB.

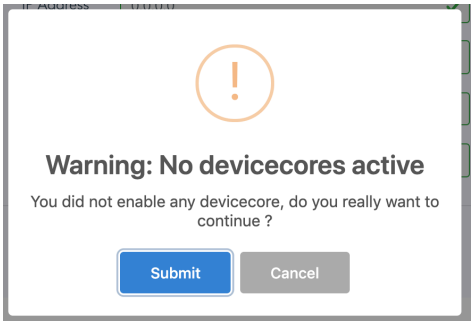


SKAARHOJ Firmware Updater and Micro USB

If a network connection to the Blue Pill in not available, the IP address can be set using a Micro USB cable. In this case the Blue Pill will appear in the "Select Device" dropdown and the "IP Config" tab in the Firmware Updater can be used to set the IP address (same procedure from UniSketch):

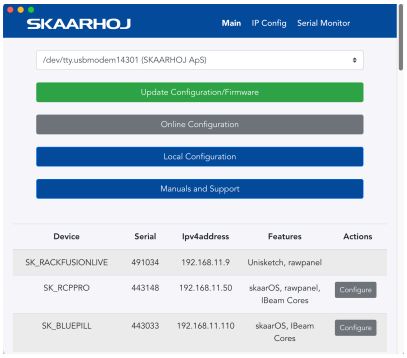
In this case it was identified as "/dev/tty.usbmodem4430361" (on MacOS) and pressing IP Config will open the IP set up page:

If/When the warning below appears, press "Submit" and reboot the Blue Pill:



Connecting UniSketch Controllers

The Blue Pill works as a wonderful add-on to connect and control UniSketch controllers like never before. There are three ways to set a UniSketch controller to allow for connection to the Blue Pill. Two of the methods will involve using the SKAARHOJ Firmware Updater available here: (https://www.skaarhoj.com/support/firmware-updater). The controller will need to be connected to the computer by USB cable. After the controller is set to Blue Pill Mode, it will no longer need USB connection for programming. This will be done through the Blue Pill web interface.



Blue Pill Mode - Via Serial monitor

Starting with UniSketch v2.5.14, it is possible to put the panel into Blue Pill Mode with a command in the serial monitor. Using this method the controller will be assigned an IP address only via DHCP.

To enter Blue Pill mode, type in: **TakeTheBluePill** in the serial monitor. The mode will be confirmed:

```
.Blue Pill Mode enabled! (Raw Panel Server Mode), resetting..
```

The IP address of the controller will now be confirmed in the boot up information for the controller:

```
*****************************
SKAARHOJ Controller Booting
*****************************
SK_VERSION: v2.5.14
SK_MODEL: SK_RACKFLYUNO
SK_SERIAL: 433769
EEPROM Size: 8192
Blue Pill DIRECT MODE enabled
I2C 400 kHz mode activated
Init LEDs and buttons
Preset 1 loaded
MAC address: 92:A1:DA:D0:73:4A
Requesting DHCP address... OK
IP address: 192.168.11.222
Subnet mask: 255.255.254.0
Gateway: 192.168.10.1
DNS: 192.168.10.1
mDNS Service started, announced for port 80
Boots Count: 191
Uptime: 250 hours, 12 minutes
Screen Saver: 99 hours, 41 minutes
Usage Stats Flags: 01
*****
Blue Pill DIRECT MODE enabled (DHCP + Raw Panel Server Mode on por
t 9923)
*****
DeviceCore #1: UniSketch TCP Client0, IP = 0.0.0.0:9923
True Encoder Button Action: 1
UNISKETCHTCPCLIENT: Server Mode = ON
Compiled: Dec  1 2021 09:14:53
setup() Done
-----------------------------
59
.UNISKETCHTCPCLIENT: Reset sleep timer last trigger
```

The IP address will also be displayed on the panel along with “Waiting for Blue Pill”

An advantage of Blue Pill Mode is, that it will not destroy any existing configuration on the panel and can easily be exited from again by writing "**reset**" or pressing the reset button in the serial monitor. Blue Pill Mode is meant as a quick way to get in and out of Blue Pill Mode when using UniSketch panels in both ways regularly.

Blue Pill Mode - Via Special Key

Blue Pill Mode can be enabled and disabled without the serial monitor on most UniSketch panels running firmware version 2.5.14 and above.

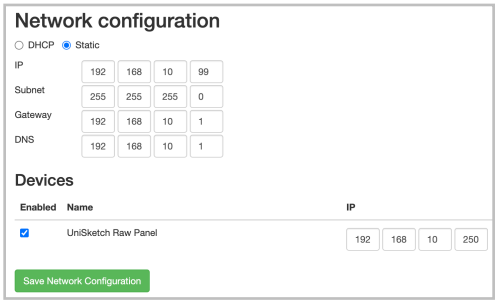
- *Enter Blue Pill Mode:* During power-up, when the color animation first appears across the buttons, press the button in the lower left corner of the controller twice. When the color animation completes, it should sweep across the controller with just blue color - and it will reboot Blue Pill Mode
- Exit Blue Pill Mode: Same procedure as above. When Blue Pill Mode is exited, the color sweep to confirm it will be white instead of blue.

The IP address will also be displayed on the panel along with "Waiting for Blue Pill" the same as when using the serial monitor to enter Blue Pill Mode.

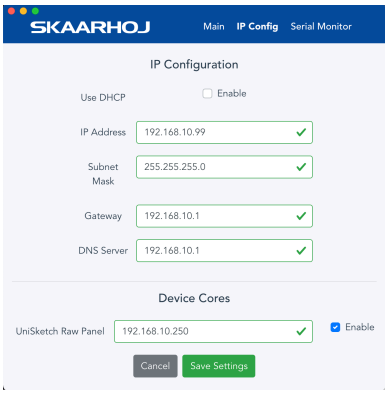
Fixed Configuration

If a static IP is needed it is possible to do so by selecting the default configuration called Blue Pill Mode and loading it onto the controller. This uses our SKAARHOJ Raw Panel integration with the needed device core options already selected. We recommend this type of configuration for any static or long term installation.

Once it has been selected the IP address can be set either in the Network Configurations section in the Simple Configuration page before the firmware is loaded onto the controller via the Update Configuration/ Firmware button on the updater.



Or it can be set in the IP Config tab of the SKAARHOJ Firmware Updater.



Please note, in both methods, the IP address for the UniSketch Raw Panel device core does not need to be set to a specific IP as it has been set to Server Mode and is being connected to and not connecting to anything. Having it Enabled is enough.

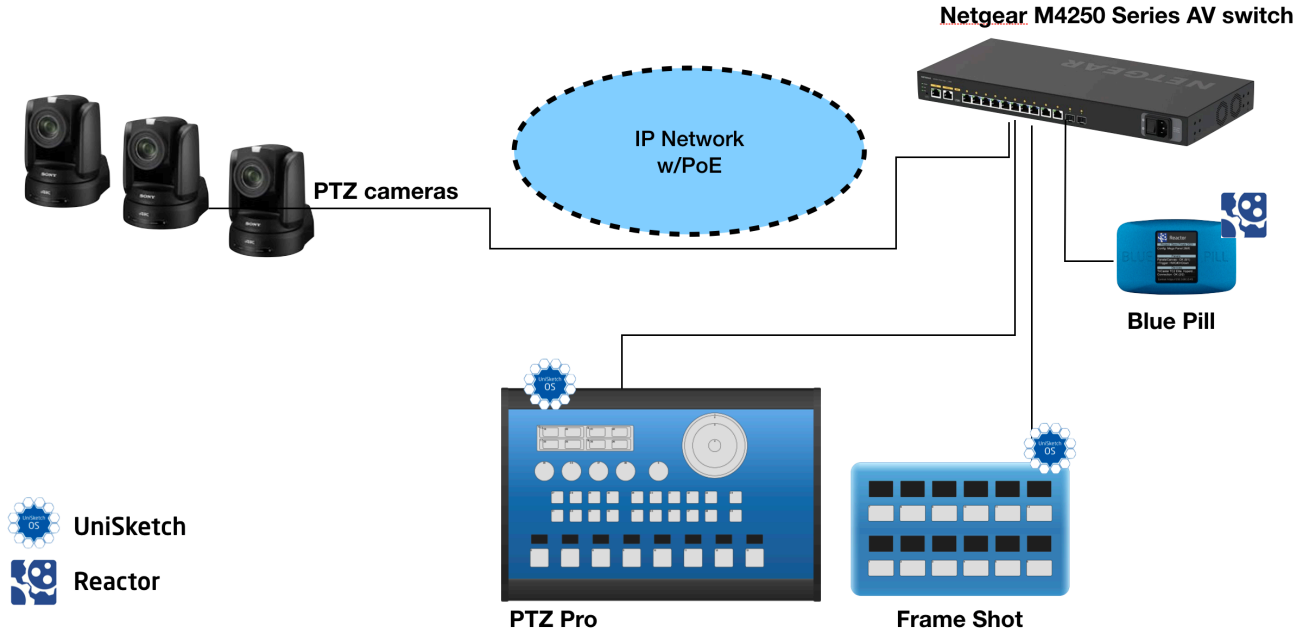
The IP address will also be displayed on the panel along with “Waiting for Raw Panel”

Network layout

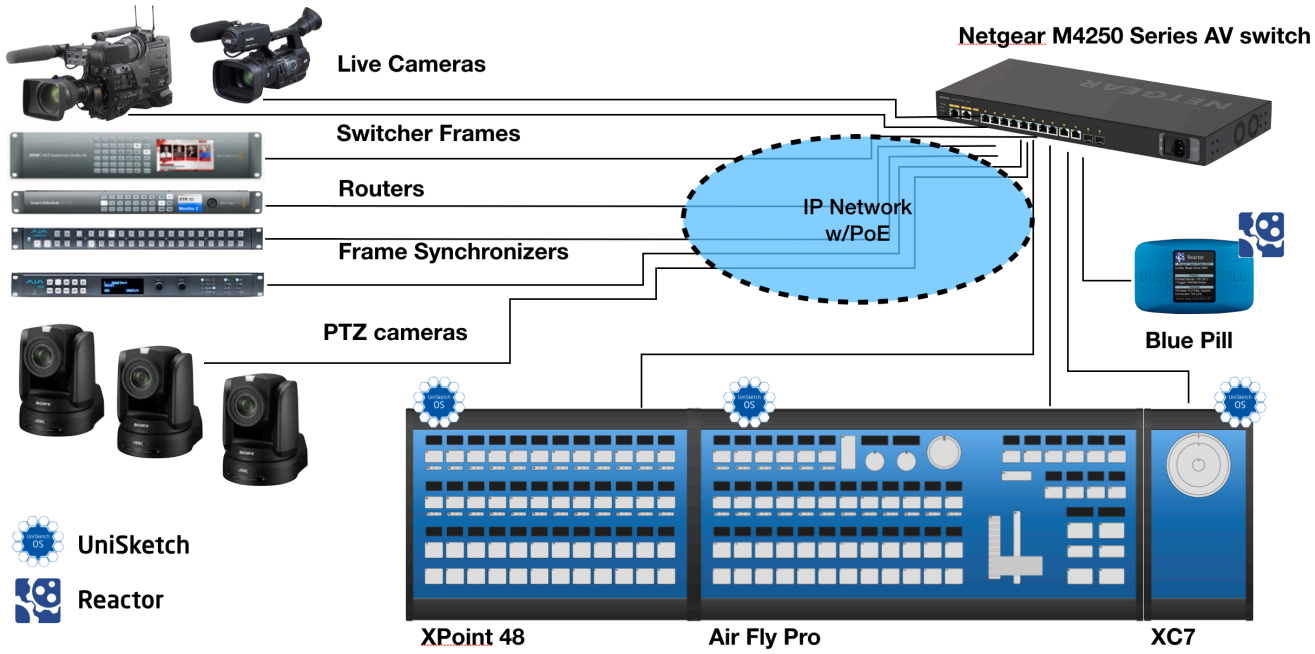
We recommend connecting SKAARHOJ UniSketch controllers and Blue Pill via professional PoE network switches, such as NetGear’s M4250-series which are designed for AV network traffic such as NDI video.

Please note, we are not networking experts and are not able to assist in network set up.

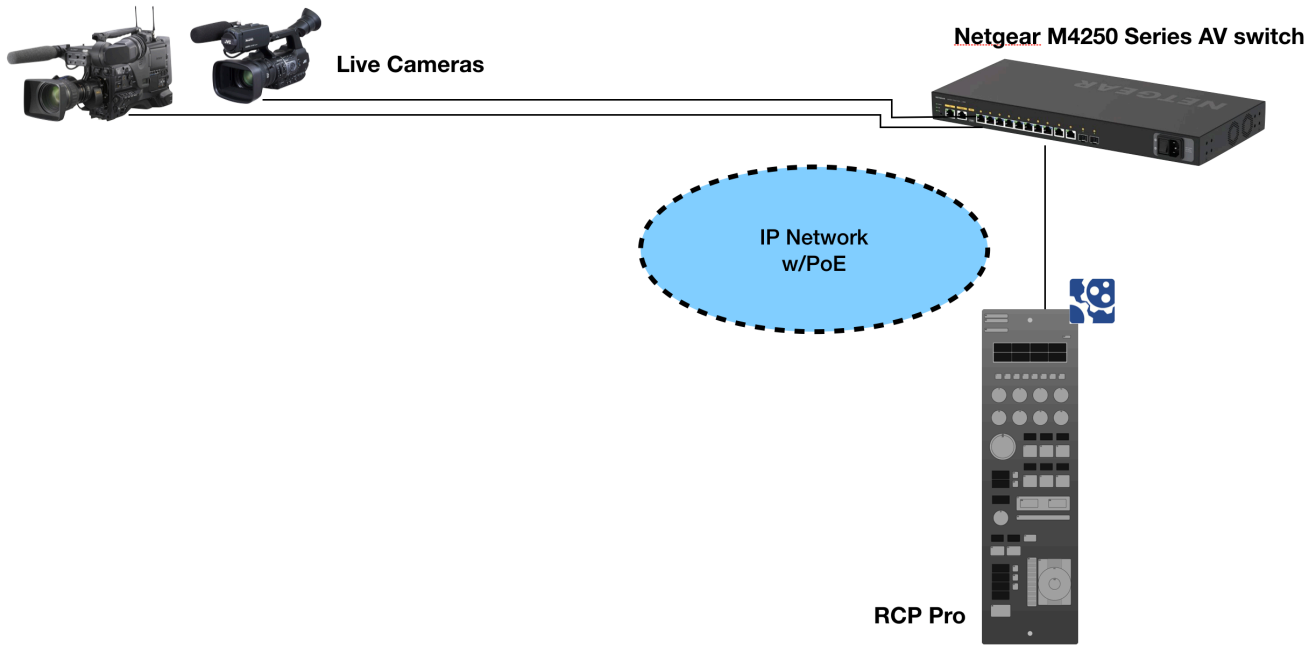
A PTZ Pro, Frame Shot and Blue Pill configuration could look like this:



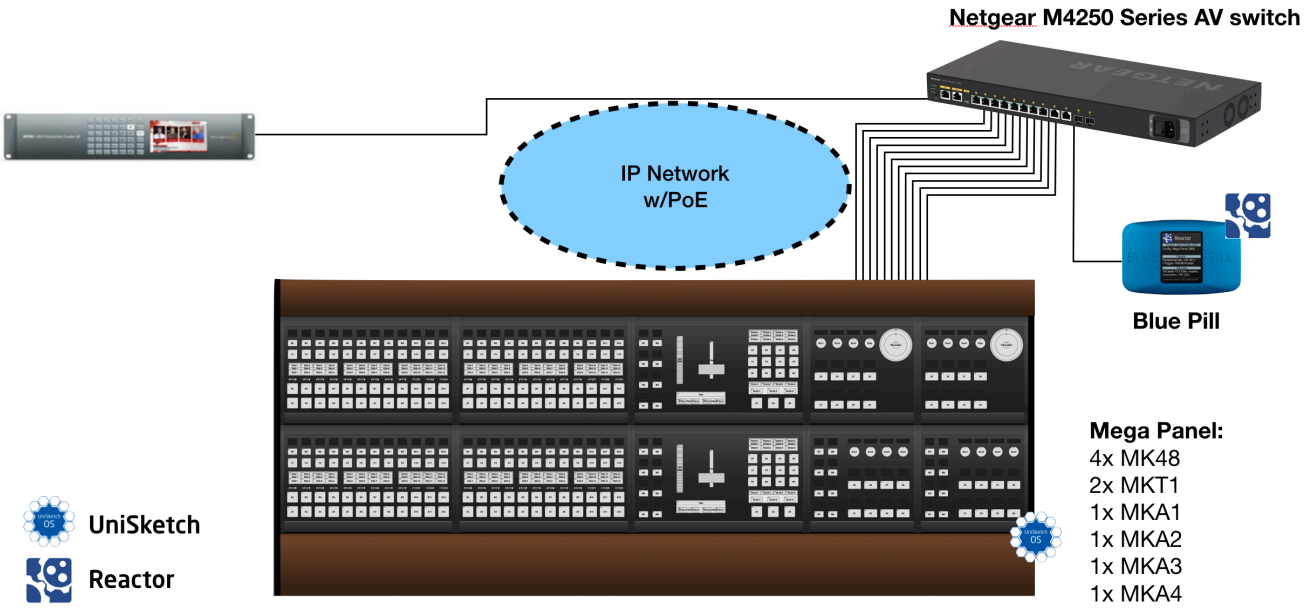
A setup with mixed cameras, switchers and routers could look like this:



Below is an example of how simple the network really is when the RCP Pro has Blue Pill Inside:



The Mega Panel is just an extended case of modularity with a large number of SKAARHOJ panels:



Network Recommendations

- SKAARHOJ Blue Pill controllers have a 1 GBit network interface
- 5W-30W PoE (+)
- PoE Standard: IEEE 802.3af/t

Remember a SKAARHOJ controller and client must be on the same subnet (192.168.10.* or one defined for the local network used by the controller). With multiple SKAARHOJ units connected to the same network they need to have different IP addresses!

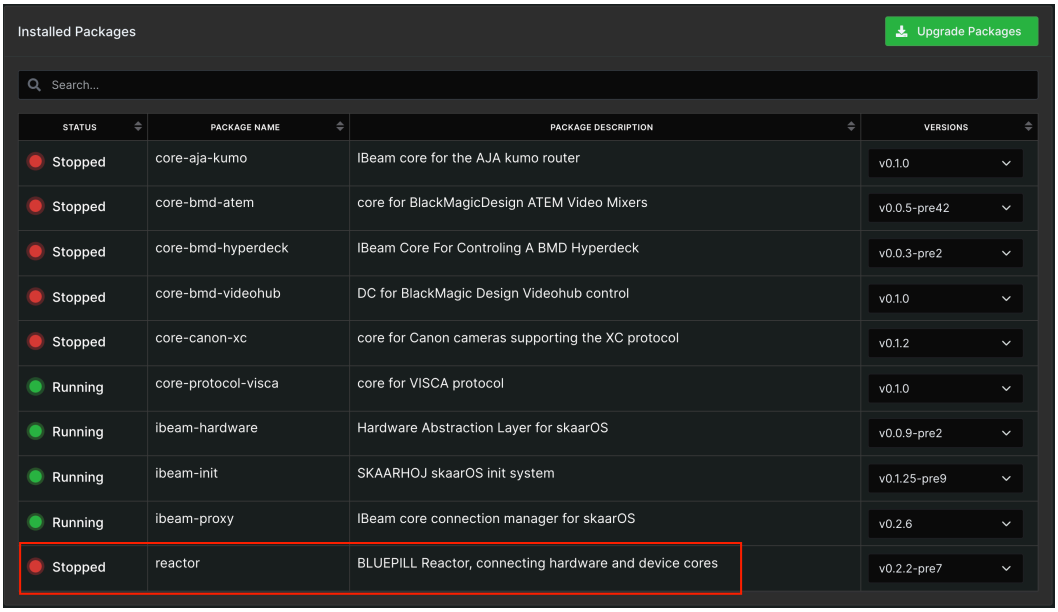
Power over Ethernet (PoE) Specifications

We use the PoE industry standard 5W-30W PoE (+) IEEE802.3af7t. To power our controllers using PoE it is important the switch supports this standard. Please notice some manufactures such as Ubiquity have their own non-standard 24V type of PoE which is incompatible with our controllers. Especially pay attention to the standard when using a PoE injector.

Troubleshooting

Reactor Icon Missing at Top of Page

If Reactor is missing at the top of the page, find REACTOR in the Installed Packages and restart it by clicking on the word STOPPED, then Restart in the pop-up that follows. Refresh the browser page after.



Additional troubleshooting steps coming soon….

Hardware

Operating Temperatures

5V Model

The 5V model of Blue Pill has a max internal operating temperature of 80°C and should be kept in a place with good ventilation or air flow and not confined to small enclosed environment (like a box) during operation. Operating the Blue Pill in a manner where the internal temperature could reach above 80°C can cause the unit's performance to decrease. Under normal operating conditions, the Blue Pill should not get so hot as it can not be held, though the enclosure for the unit will get warm as it works as the heatsink for the CPU and built-in power supply.

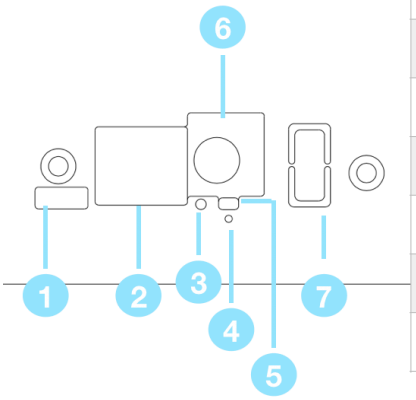
Graphical Display

The Blue Pill has a graphical LED Display. By default this is set to go into a sleep mode to prevent image burn-in. The unit is still operational during sleep mode, it is only the display that effected. Burn-In caused

after disabling sleep mode is not covered by our standard warranty.
To wake the Blue Pill up from sleep mode, it should only be necessary to tap the unit. It is not necessary to shake it like a polaroid picture.

Back Connections

Blue Pill Inside Units

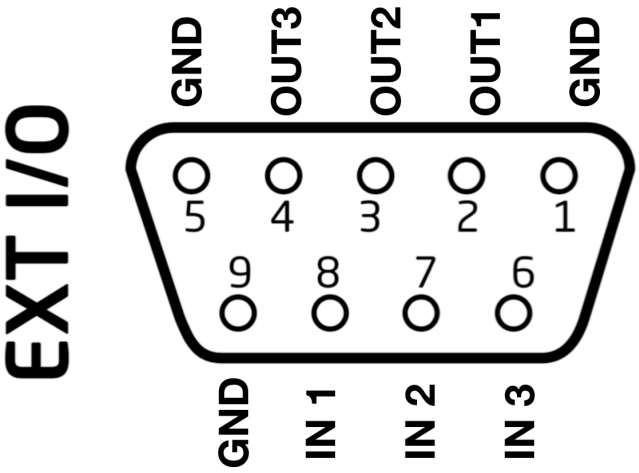


1	Micro USB Port for serial communication with SKAARHOJ Firmware Updater
2	IP Network RJ45 Port for IP control and 5W-30W PoE (+)/PoE Standard: IEEE 802.3af/t
3	Status LED for monitoring and debugging
4	Reset button. Use only under the guidance of SKAARHOJ support
5	Config Button to enable WiFi Access Point. See Accessing Blue Pill section
6	12V DC Power Supply for connection to the supplied DC power adapter
7	USB-A Port. See Settings page for more details

Blue Pill Units

Coming Soon

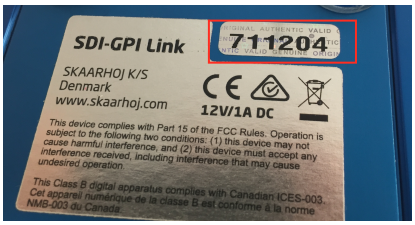
DB-9 Connector for RCP Pro



Contact Support

It is always possible to contact us for support questions - write an email to support@skaarhoj.com and we will do our best to accommodate the request.

In order for us to provide the best support please state:



- Which SKAARHOJ unit it is about
- The serial number of the device (small silver label with 6 digits)
- The nature of the problem
- Which hardware device(s) are to be controlled and their firmware version
- If the device's web interface has been successfully accessed
- The connected computer's operating system

End notes

Reflections and Acknowledgements

While the Covid-19 pandemic has been a depressing catastrophe for the whole world, it also brought about an opportunity for intense focus unspoiled by interruptions of the usual trade show activity during a year. There has been ups and downs for most companies. SKAARHOJ has experienced a balanced mix, but personally I have had an amazing year coming back to intense software development. So leaving the depressions of Covid-19 aside, I have rarely felt so excited about my company's technology as in the process of creating Blue Pill and Reactor. I learned a programming language from scratch (almost everything written on the backend for Reactor and Blue Pill is written in Googles language, Go) and was overwhelmed by its approach to coding. I owe deep thanks to those of my engineers that lead us on this path. It was not only the right technological choice, but also life transforming for me.

Here, I want to thank my global engineering team for the immense effort they have put into this product. In less than a year, we have gone from just ideas to a conceptually matured panel management application, industrial Linux build, and many powerful device cores that will support our hardware into the future for decades and which will be a worthy basis for our claim to *invent the future of media production control*. You have create fantastic UIs, awesome device cores, incredible architectural and infrastructural features to back it up. Although I have done a lot of it myself and been deeply involved in the whole thing, I couldn't have done it without you. Absolutely not. Thanks for being a part of this journey with SKAARHOJ and I'm looking forward to continuing the adventure together!

- Kasper Skårhøj, Company owner, founder, CEO, CTO and geek

November 2021

WEEE Information

Figure 1

For private households: Information on Disposal for Users of WEEE
This symbol (figure 1) on the product(s) and / or accompanying documents means that used electrical and electronic equipment (WEEE) should not be mixed with general household waste. For proper treatment, recovery and recycling, please take this product(s) to designated collection points where it will be accepted free of charge.
Alternatively, in some countries, it may be possible to return the products to the local retailer upon purchase of an equivalent new product.
Disposing of this product correctly will help save valuable resources and prevent any potential negative effects on human health and the environment, which could otherwise arise from inappropriate waste handling.
Please contact the local authority for further details of the nearest designated collection point.
Penalties may be applicable for incorrect disposal of this waste, in accordance with national legislations.

For professional users in the European Union
To discard electrical and electronic equipment (EEE), please contact the local dealer or supplier for further information.

For disposal in countries outside of the European Union
This symbol is only valid in the European Union (EU). To discard this product please contact local authorities or dealers and ask for the correct method of disposal.