User manual

Version 4.07

Content

V-Control Quick Start

Concept

V-Control is designed to control devices. A device can be anything that has a control interface. Typical control interfaces are serial (RS232, RS422 and RS285), Network (TCP, UDP or HTTP), MIDI, DMX, dry contacts, wet contacts etc.

So, from small sensors up to large LED screens you can control anything with V-Control.

Devices

The philosophy is to have a virtual device in V-Control for each real device that is being controlled. The virtual device (in V-Control its the device driver) is a representation of the real device.

To control the device, a device driver has commands. A command can be a Play command for a player, or an input switch for a projector. V-Control has many device drivers build in, and if one uses one of those drivers, there is no need to know what exactly has to be send to the device to perform the command. The driver offers a command such as “Switch Input to DVI”, and it generates the necessary string and send it to the device.

New in V-Control 4

The (virtual) device driver requests the status of the controlled device and offers commands to control it. The status of a device is represented by device variables. A projector for example can have a variable for Power state, showing if the projector is switched on and of. And a second variable for Input, showing which input is currently selected. The number of device variables depends on the device itself and is not limited.

To act on variable changes we create event handlers. Event handlers are listed in the Event List. Here we create a condition, and if the condition is true, the task that is linked to the event is executed.

V-Control 4 and old Device Drivers

Some of the V-Control 3 drivers will not compile on V-Control 4 (All that make use of “NextCmd” and “Delay”). In our Driver Database, most of the drivers work in V-Control 4. We marked some drivers with a “_v3” or “_v4” extension.

- Drivers without extension work with Version 3 and 4, but do not benefit from V-Control 4 capabilities. They are used in the same way as in V-Control 3
- Drivers with “_v3” extension work only in V-Control 3 and need an update
- Drivers with “_v4” extension work only in V-Control 4

Channels

A channel is a physically present communication interface, such as RS232, TCP, UDP etc. A channel is the object that is used to talk to the device that is attached to the V-Control PC. A Device in V-Control doesn't care what kind of channel is used. You can send the same command via RS232 or TCP, it depends only which channel is assigned to the device. To configure channels for a project use the Channel Editor

Tasks

Tasks are a group of commands. The simplest type of task is a cuelist. A task should be only a part of the whole show. You can compare tasks with functions, methods or subs in other programming languages. A task is always running in its own thread.

Cuelists

V-Control knows three types of tasks. A cuelist is just a list of cues. If a cuelist is started it runs from top to bottom and executes the commands in the list. Cuelists only know a limited number of branch commands such as “Repeat”, “CallAsFunction” or “CallAsThread”. Because a cuelist is limited in flexibility, it is very easy to use, and most of the work can be done in cuelist.

Scripts

Scripts have all the power one may need for complex tasks. They are programmed in basic, but compile before running.

Timestrips

Timestrips are comparable to timelines, but the representation is a table.

Any type of task can call any type of task. In a cuelist task one can call a script task or a timestrip task and vice versa.

Installation

The installation is very easy on all supported platforms. Unpack the compressed file in any directory where the user has **write access** (**not** c.\ Program Files or c:\Program Files (x86)) and launch the binary file. On Linux, it might be necessary to set the file permissions to allow execution.

On Raspberry Pi, libunwind8 is needed. Open a terminal and type

```
sudo apt-get install libunwind8
```

On Linux and Raspberry Pi it might be necessary to make the V-Control4_Lin file executeable.

Uninstallation: Delete the program directory, that's all.

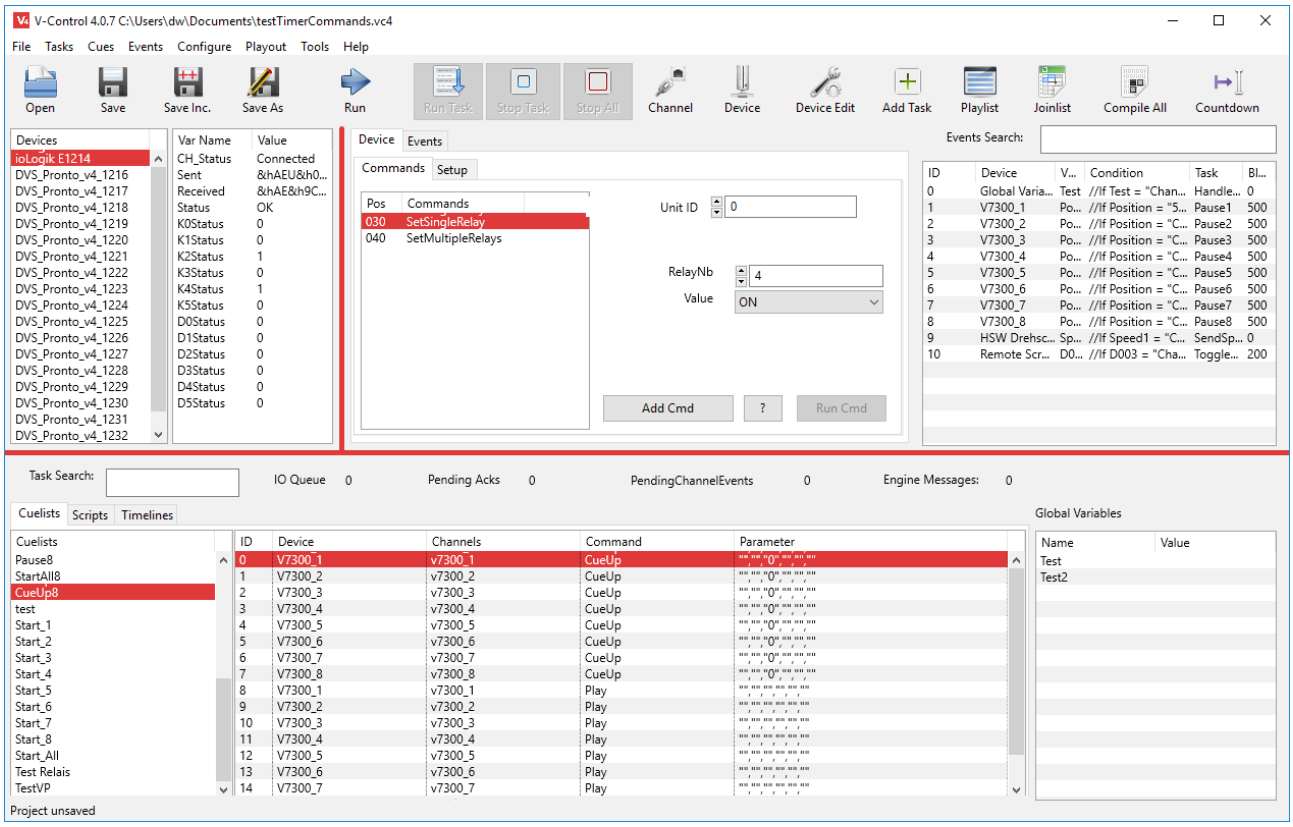
User Interface

Main Window

The V-Control user interface provides quick access to all configured devices, tasks and Events. All the three programming modes, as there are Cuelists, Scripts and Timelines, are accessible within one user interface.

The selected kind of Task (Cuelist, Script or Timestrip) is determined by the tab bar in the lower half of the application window.

All three programming modes can be used together and mixed within one project.



The red Lines

These lines are used to change the position and size of the GUI elements. Click and drag the lines to change the GUI.

Device List

A list of all available configured devices.

Device Variable List

Each device can have a set of variables. Variables are used to represent the

state of the device. They are usually filled automatically from the device itself. If a variable change, this can trigger an action.

Command List

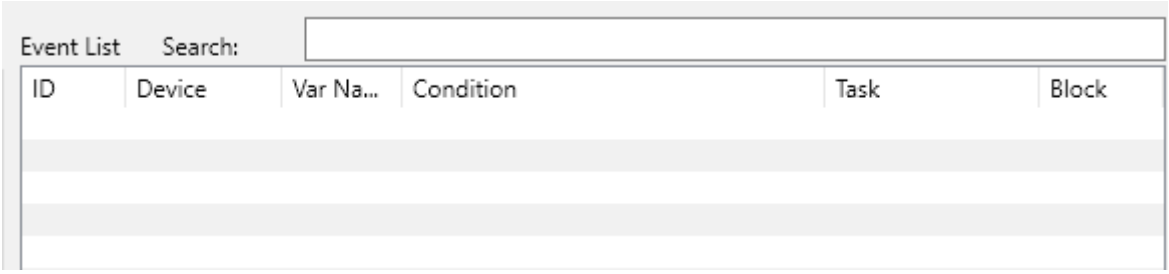
The command list contains all the available commands. If a command is selected it can show parameters for the command.

Command Parameter

Most devices and commands need parameters. A parameter can specify a device property (e.g. a bus address or name) or a command parameter such as *Input Channel* to change the input of a device. In V-Control, each command can have up to 6 parameter, two device parameters and four command parameters.

Event List

Here are all programmed events located. If a device variable change (that can be a new Timecode position during playback of a video player, an input change of a projector or a system time change) this event can be linked to a task. Each event that you want to handle is listed here.

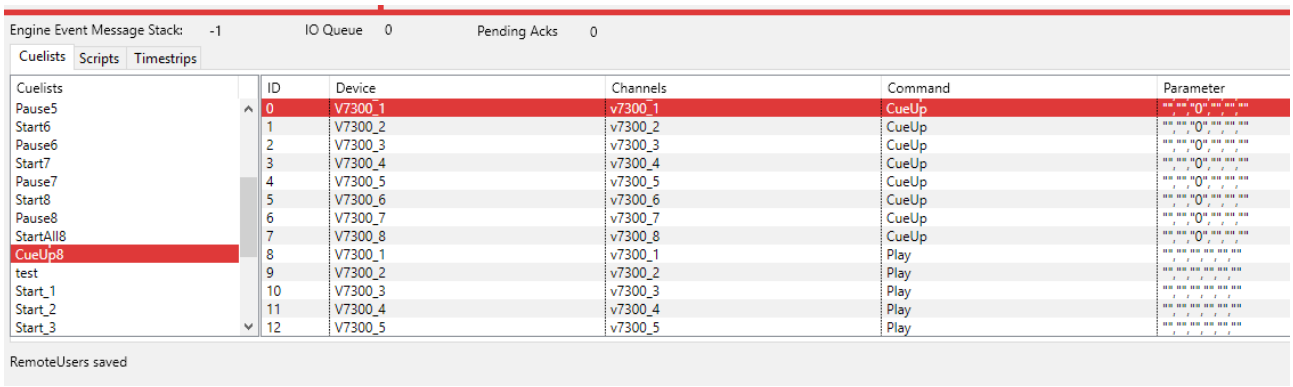


In complex project the eventlist can be quite long. To help finding events for debugging there is a search field. If text entered here matches any of a row content, then only these rows are diplayed. Hit the [ESC] key or delete all text in the search field to have the unfilterd list back.

Tasks

Task is a generic name for V-Control scripts. A task represents a program or subroutine. They contain the instructions for the attached devices (e.g. play a movie) and the program flow. V-Control knows three different types of tasks, Cuelists, Scripts and Timelines. To access the tasks, select one of the tabs at the bottom

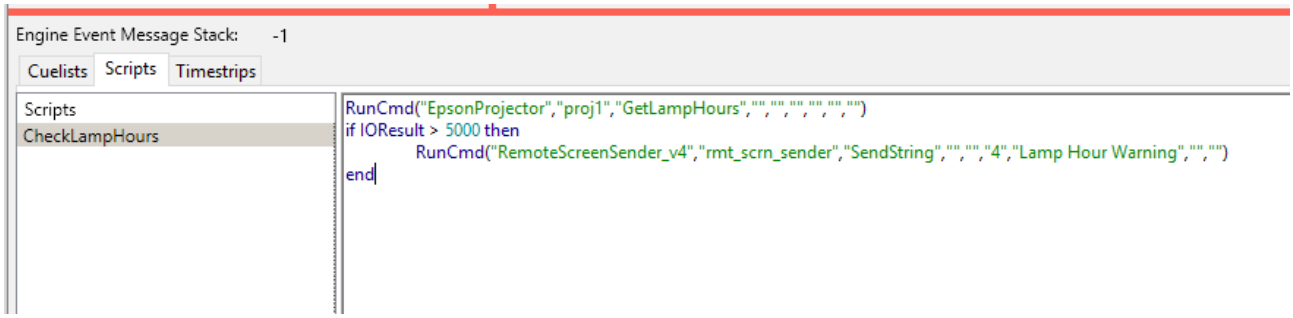
Cuelist Tasks



These tasks are very easy to use but have some limitations. Every program step is written in a table, and the table is executed from top to bottom. Branching is possible if calling other tasks from that list (CallAsThread, CallAsFunction). Also (nested) repeats are possible. Conditional branching is available in Scripts only. The biggest advantage of cuelists is that you can program very fast and you can't make as many mistakes as in scripts.

Script Tasks

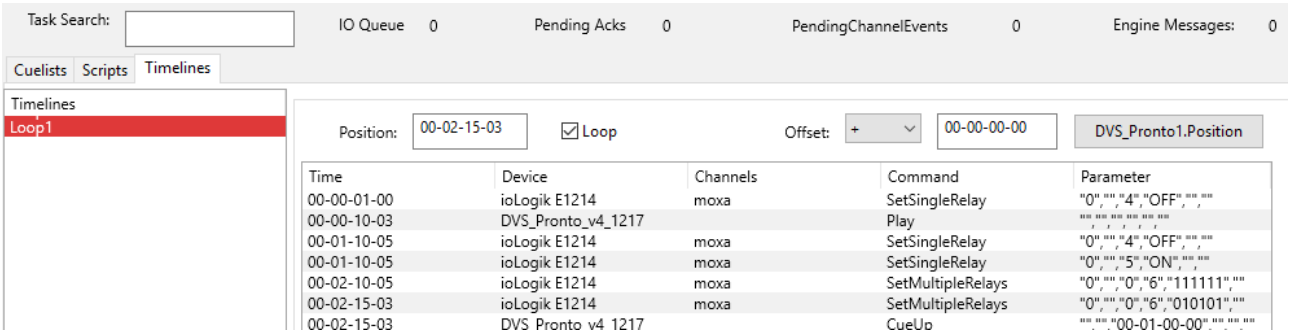
Scripts are real (and for the chosen platform native compiled) basic programs. In V-Control, the basic language is enhanced by commands giving access to the device drivers and channels.



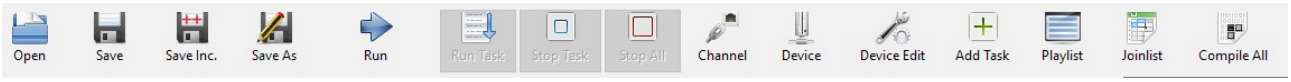
The screenshot below shows the Script GUI.

Timeline Tasks

Timelines are used for a timeline like programming. The Timeline is displayed as a table. Each entry starts with a time that represents the start condition of that cue. To allow multiple commands to be executed at the same time, two or more rows can have the same time as start condition.



Toolbar



Open

Opens a Project

Save

Saves a Project. The saving process is running on it's own thread in the background. It is not possible to terminate V-Control while a saving process is running.

Save Inc.

Saves a Project with an incrementing number. If the project name is “myProject.vc4”, then pressing this button saves “myProject_001.vc4”. Pushing the button a second time saves “myProject_002.vc4” and so on. **Attention:** *Save Inc.* overwrites existing files without asking. To avoid this, make sure that you always open the project with the largest number after V-Control is launched.

Save as

Saves a Project under a new name

Run

This button toggles. If unpressed, V-Control does not send or receive any commands. If pressed, V-Control will perform all events and commands.

Run Task

Runs a selected task. *F5* key does the same (only available in Run mode).

Stop Task

Stops the selected task. *F6* key does the same (only available in Run mode).

Note: This Action stops the selected task, but if the task is re-executed by an event, then it runs again.

Stop All

Stops all running tasks. *F7* key does the same (only available in Run mode).
Note: This Action stops all tasks, but if a task is re-executed by an event, then it runs again.

Channel

Opens the Channel Editor

Device

Opens the device setup page. Here a device driver can be selected from the device database.

Device Editor

Opens the Device Editor

Add Task

Creates a new task. The type of task (Cuelist, Script or Timestrip) is determined by the task tab that is visible at that moment.

Playlist

Opens a playlist to drive a show.

Joinlist

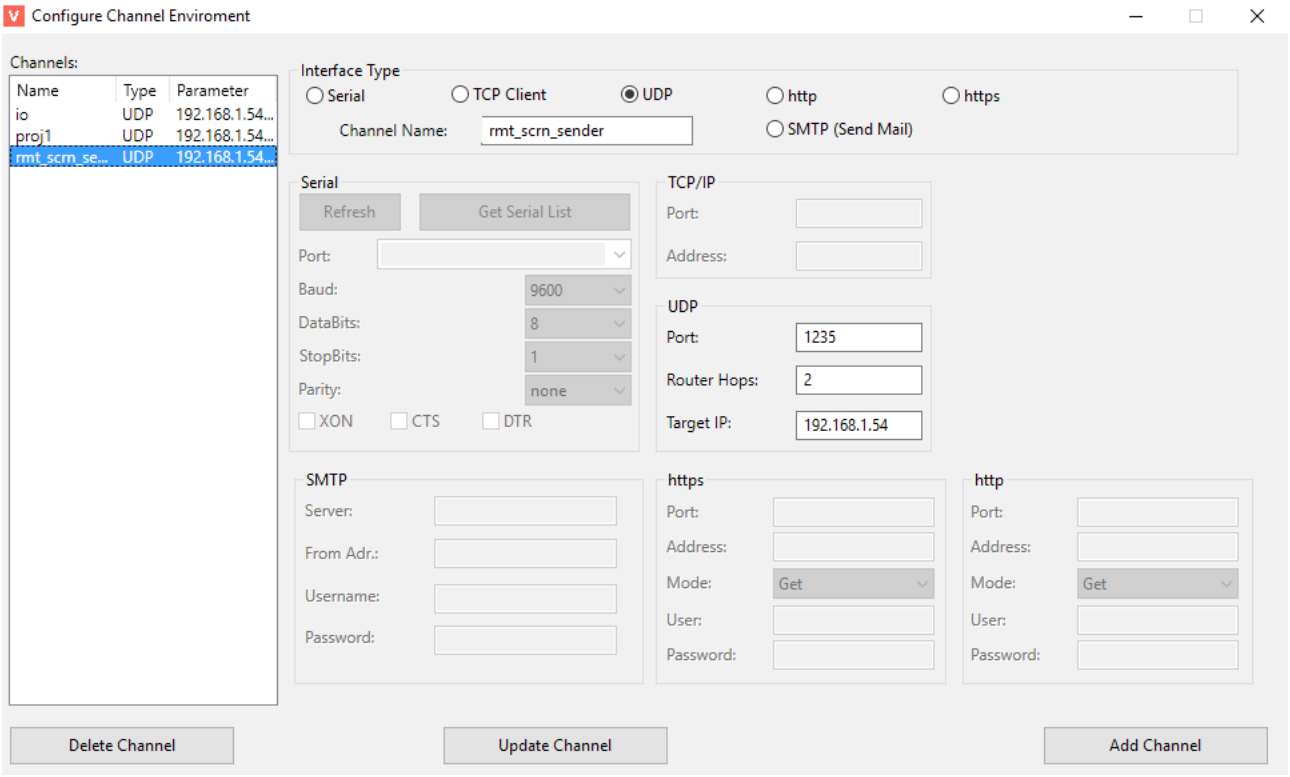
To receive button presses from external GUI's running on a tablet or PC, and to update GUI elements such as labels, buttons etc. V-Control uses Join Numbers to identify these widgets. In complex project there can be hundreds of widgets, and some widgets can have multiple join numbers. This list can help to keep the overview of all those numbers.

Compile All

V-Control scripts are compiled once automatically on their first start. This button does the same for all scripts before they run the first time.

Channel Editor

Basis for the communication with attached devices are the channels. A channel gives access to a physically interface - e.g. a RS232 port or a UDP port. The device drivers use these channels to communicate with the devices attached to the channels.

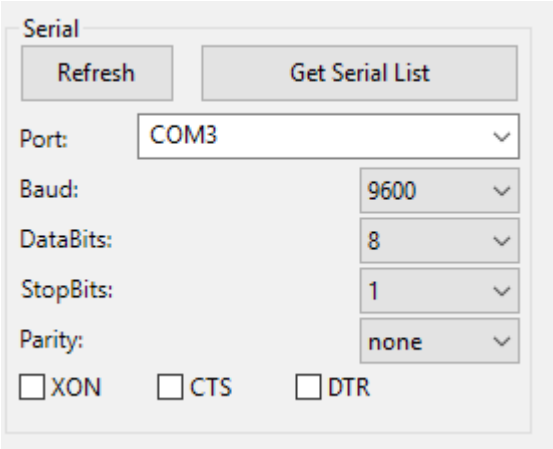


Via main menu *Configure* - *Channels* (or toolbar shortcut) the channel editor opens. Every channel is identified by its Channel Name. For that reason, not only interface parameters are needed to specify the channel, but also a unique channel name. The channel name should (but is not required) give a hint on how the channel is used. E.g. a channel used to control a Panasonic projector could have the name “proj1_left”, the channel for the second Projector is named “proj2_right” and so on. We recommend to name the channels in lower case letters, and the Devices with a leading upper case letter.

Before creating a new channel, it is necessary to specify its kind in the Interface Type area and enter a channel name as described above. To edit a channel, select its name in the left list and change its parameters. Then click Update Channel.

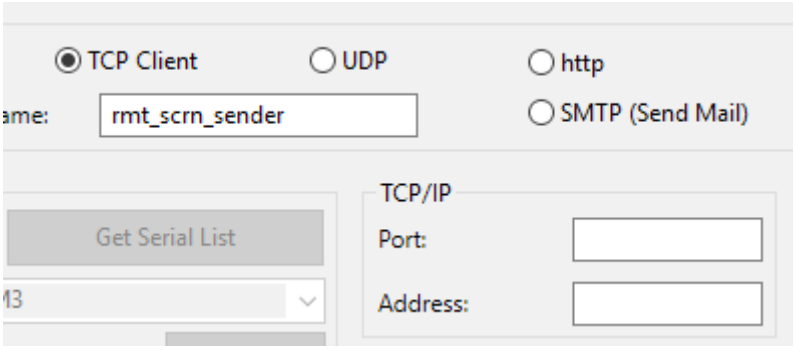
Serial Channel

Serial channels are used for RS232/422 and 485 interfaces. Get Serial List loads all previously scanned ports in the Port: list. If new serial ports attached to the system, Refresh executes a new port scan. On systems with many serial interfaces this could take a while.



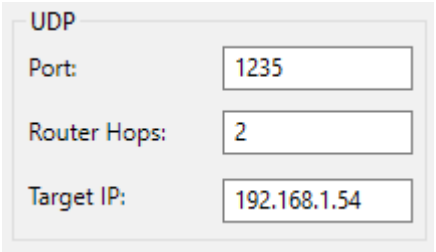
TCP Channel

To control devices via TCP, the Port and IP Address is needed



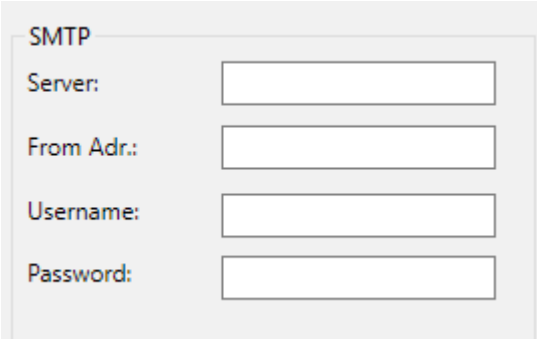
UDP Channel

Router Hops determine how many routers the UDP packet is allowed to pass.



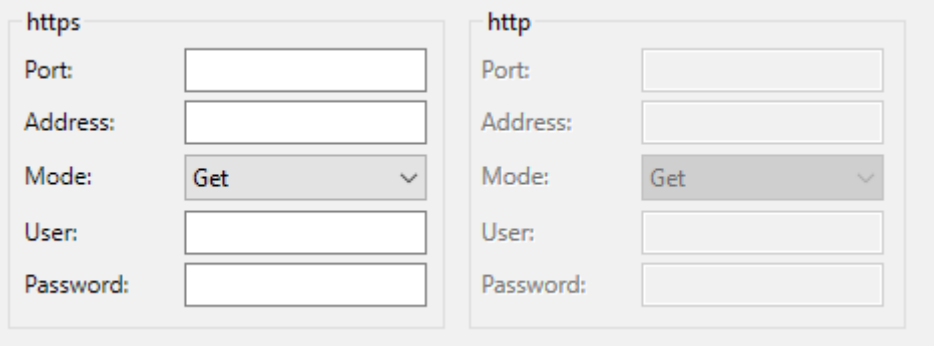
SMTP Channel

The SMTP channel specify's a mail server. This channel is used by the SMTPMailClient device to send emails.



Server: The mail server (e.g. smtp.web.de) From Adr.: The sender address (me@myspace.com) Username: Username to log in Password: Mail server password

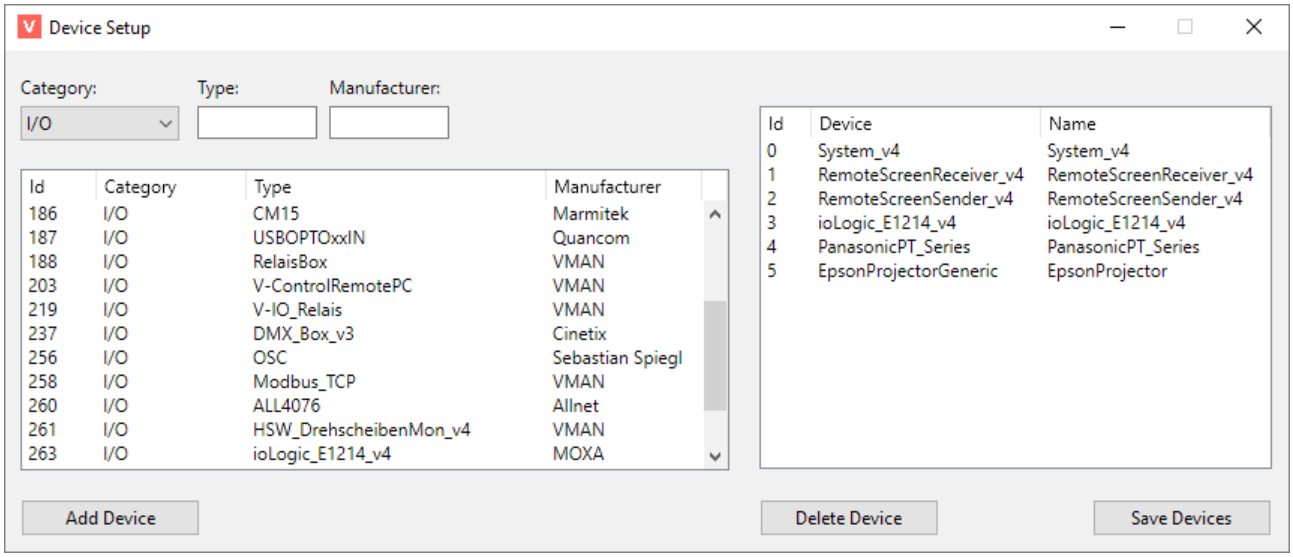
Http and Https Channel



HTTP and HTTPS channels can automatically log in if username and password is saved in the channel properties. They both can do http Get and Post requests.

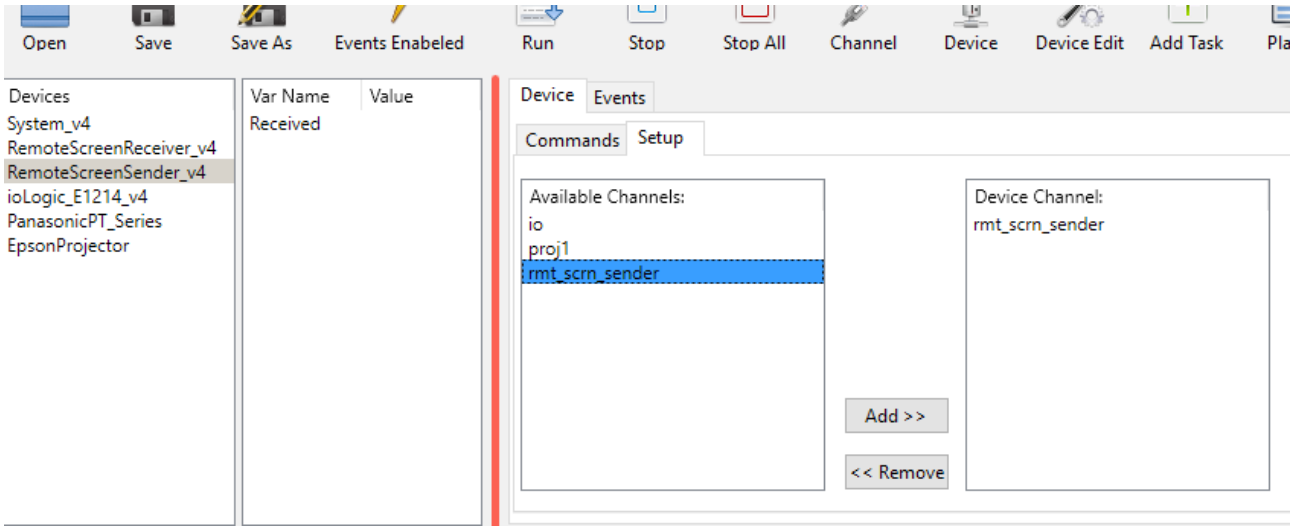
Device Setup

To configure the devices select *Configure - Devices* in main menu (or *Device* shortcut in the toolbar) . The Device Setup window opens as shown below. The right table lists all devices currently available in the project. The left table list device templates, whose search criteria matches one of the fields Category, Type or Manufacturer.



If the desired driver appears in the left list, a double click or *Add Device* adds the device to the right list (project devices). *Save Devices* saves the right list and make the devices available.

Before the device is usable, it is necessary to assign at a Channel. To do that, follow the instructions in the screenshot below.



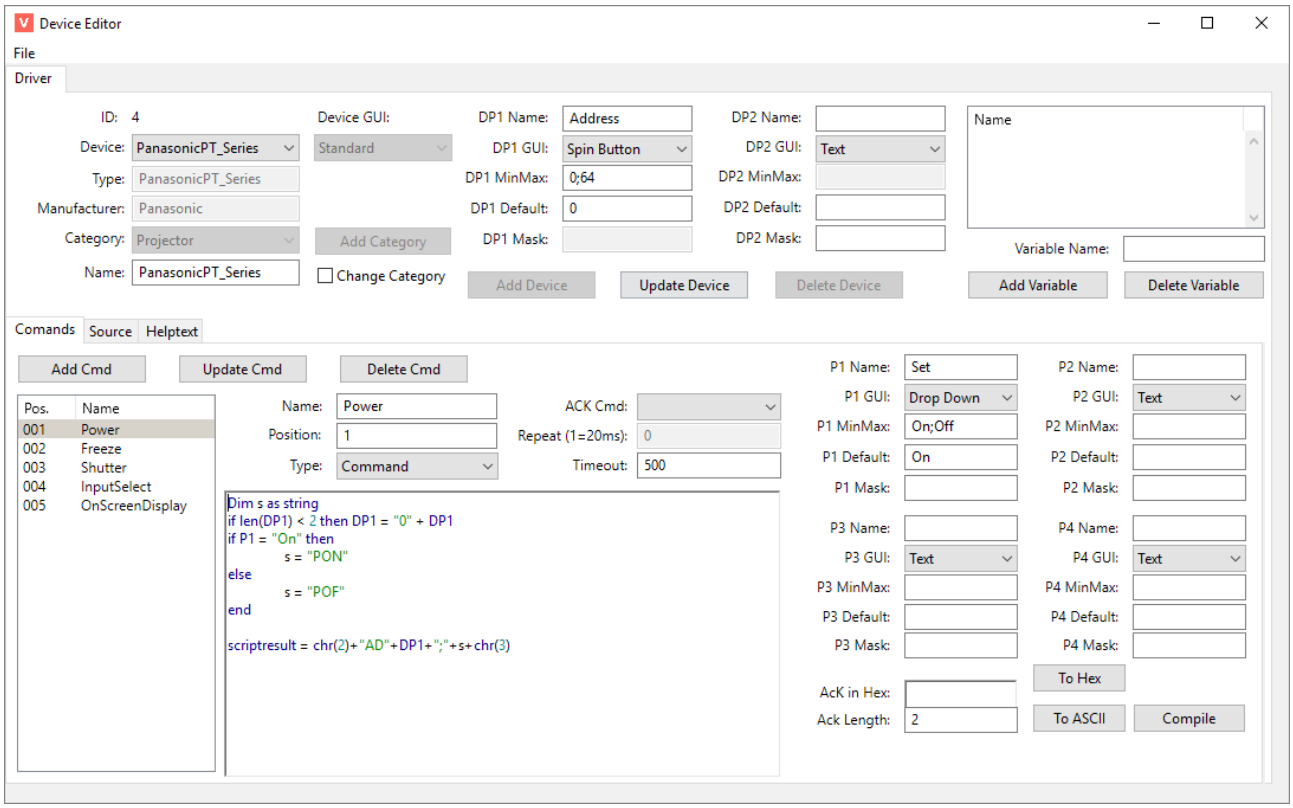
1. Select the device in main windows device list
2. press the setup tab (right from commands tab)
3. in the *Available Channels* list select the channel to assign to the selected device

4. press the *Add* button

Device Editor

It is possible to customize (if needed) the device drivers for a project. A complete description how to create and edit device drivers is found in the Device Editor chapter. In most cases it should be sufficient to change the device name. To do that, follow the instructions below.

The Device Editor is launched via *Device Edit* symbol in the toolbar or main menu *Configure -> Edit Devices...*

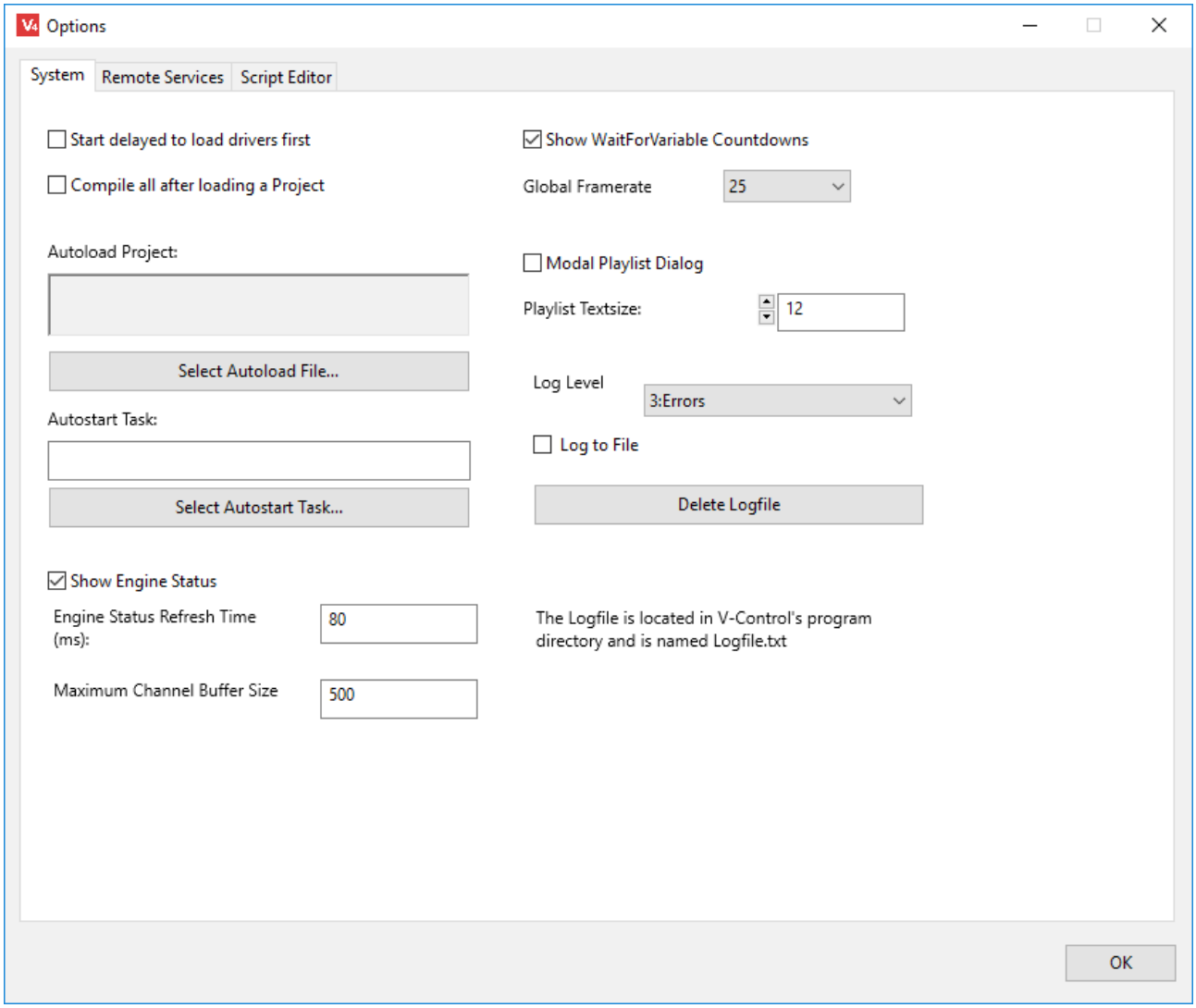


To change a device name enter / edit the name in the *Name* textfield. The device is selected in the Drop Down Box *Device:*. *Update Device* saves the new name in the current project.

The device Editor knows two modes. If launched via *Configure -> Edit Devices* or the *Device Edit* toolbar icon, all changes made in any driver stored in the project (*.vc4 file). The original driver template stay untouched. If launched via *Configure -> Edit Device Templates*, the template is edited, and the project devices stay untouched. This is important to keep in mind if editing device drivers.

Options

Via main menu *Configure -> Options* the program preferences are available.



System

- *Start delayed to load drivers first*:If V-Control runs stand alone and is started at system start automatically, it could be that network drivers or drivers for serial cards are not loaded when V-Control is ready. With this option V-Control starts and the waits before loading a project file.
- *Compile sll after loading a Project*: All Scripts, device Drivers and Event Scripts have to be compiled befor they can run. This happens “on the fly” if a script runs the first time. On weak machines such as Raspberry Pi or slow X86 machines this could lead to missing events or slow processing if a script runs for the first time. To avoid this, this option compiles all scripts after a project was loaded. Depending of the amount of devices, events and tasks, this could take some time. The status bar shows a progress message.

- *Select Autoload File*: selects a file that is load during startup.
- *Autostart Task*: launch the selected task if the Autostart File was loaded. **Note:** Only if an *Autostart Task* is selected, V-Control switches to Run Mode automatically. If you don't need a task running at startup, but want to enter Run Mode automatically, create a task that does nothing and select it here.
- *Show Engine Status*: If checked, the load of the control engine is displayed in the main UI. *Engine Event Message Stack* shows the number of messages that are waiting for processing. *IO Queue* shows the nomber of commands in the command buffer waiting for sending. *Pending Acks* shows the nomber of Acknowledges the engine is waiting for. *Pending Channel Events* shows the number of Events waiting for processing.
- *Engine Status Refresh Time*: Determines how often the GUI is updated with Variables and logging messages. Keep this at the default value unless you have a good reason to change this.
- *Maximum Channel Buffer Size*: Determines the input buffer of a channel. If a channel receives data, and this data is nt processed, the inbut buffer will continiously grow. This value determines when the buffer is cleared.
- *Show WaitForFariable Contdowns*: If a WaitForVariable command waits for a numeric or time value, and the comparsion is >, >=, < or <=, then a window with a countdown appears.
- *Global Framerate*: This framerate is used in the Timeline. The framerate is also used in Timecode calculations.
- *Modal Playlist Dialog*: by default, the playlist is no modal dialog (means that you can access other V-Control elements while the playlist is open). If this is checked the playlist is modal. This is useful if keyboard shortcuts are mapped to playlist items and you don't want the playlist to lose its focus.
- *Playlist Textsize*: changes the font size of the Playlist.
- *Logging Level*: determines how many logg entrys are generated. Default is 3 (only errors). Level 2 is warnings (Timeouts), Level 1 is commands and level 0 is also all internal commands for device status acquisition. With loglevel 0 you'll get a rapidly filling list. The log window opens automatically if a new logmessage corresponding to the selected loglevel appears.
- *Log to File*:writes all messages to a textfile

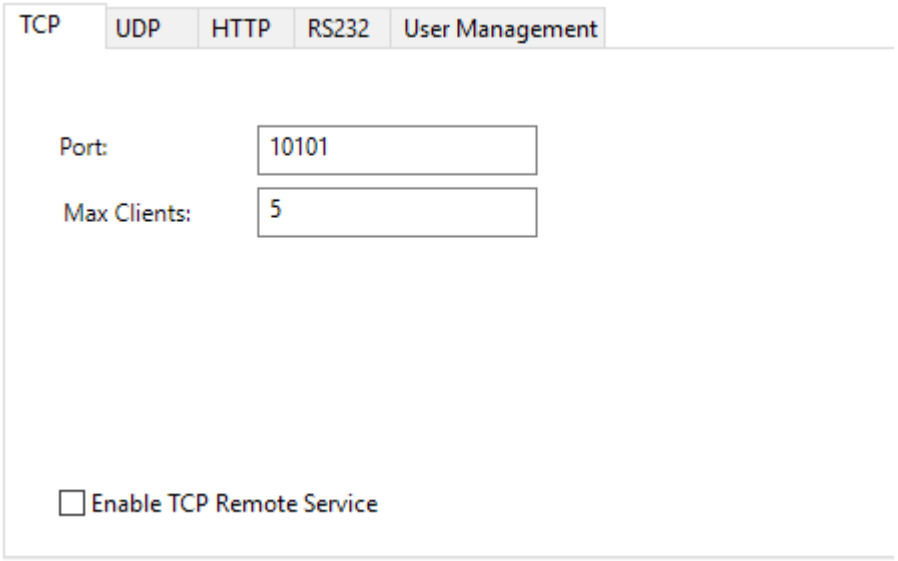
Remote Services

Remote Services are used to make V-Control functions available for other applications. They can use Ethernet (TCP/UDP/HTTP) or RS232 to communicate with V-Control. The communication protocol is described in the Node Protocol chapter. To make Remote Services available via network, it might be necessary to configure the firewall of the System. All kind of remote services described below can be used at the same time.

TCP Remote

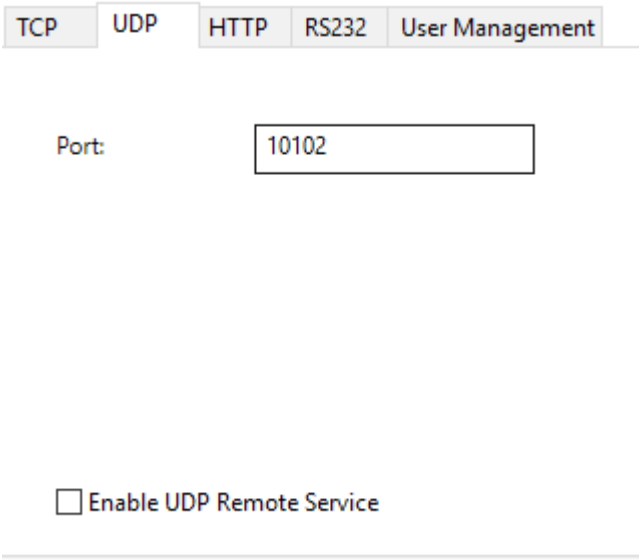
The TCP service can be used by any application that is capable to connect to a

TCP Server and send the command strings as described in the Node Protocol chapter. The Port is 10101 by default an can be changed. Max Clients determines the maximum number of simultaneously connected clients. The check-box *Enable TCP Remote Service* starts the service.



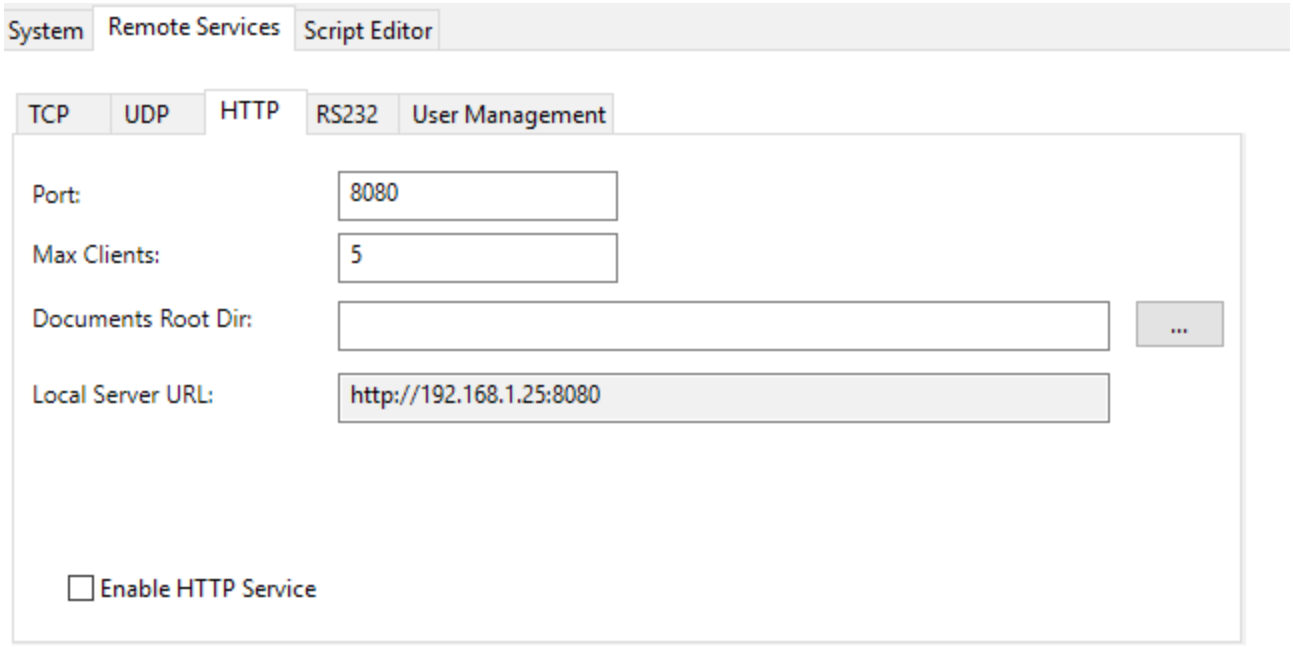
UDP Remote

The UDP Service uses the same protocol than the TCP Service. If both services used at the same time, it is necessary to use a different port. For that reason, the default UDP Port is 10102.



HTTP Remote (Webserver):

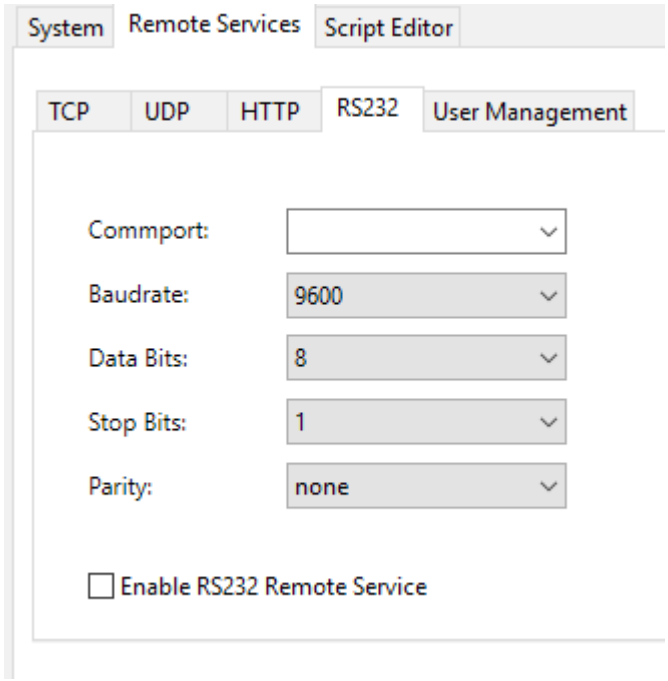
V-Control comes with a build in web server.



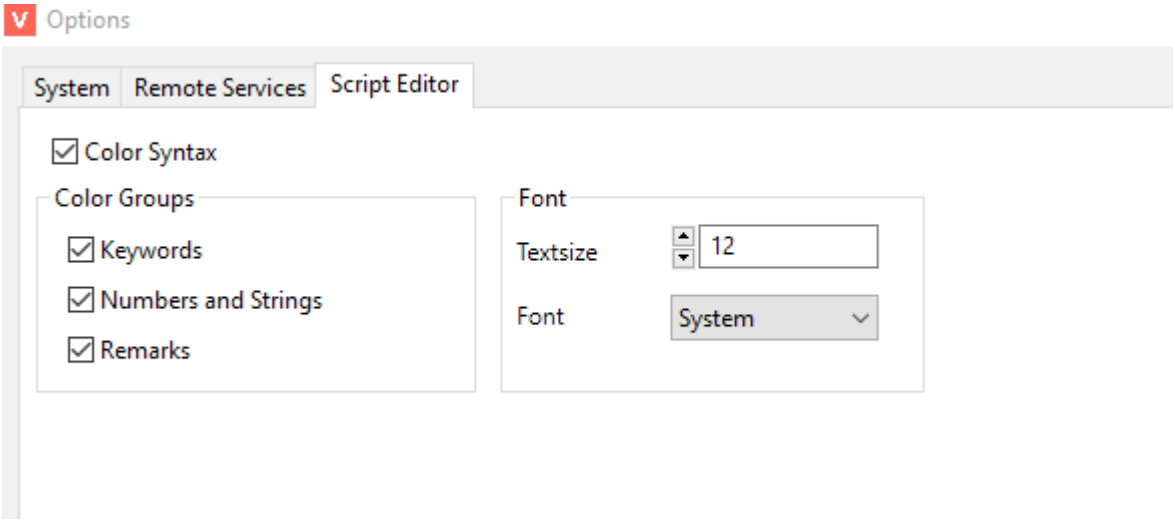
The HTTP Port is 80 by default. In this case it is sufficient to enter the IP address of the V-Control system in the browsers address field. If the web service is not available, check the firewall configuration of the computer or try another port. If another port than 80 is used, the port number has to be added to the URL (e.g. http://192.168.1.5:8080) if Port 8080 is used alternatively. *Documents Dir* contain the path to the html documents directory.

RS232 Remote

If there is no network available, it is possible to use a RS232 connection. The RS232 service uses the same protocol than TCP or UDP services.



The Script Editor Page



This dialog is used to configure the Script Editor

- **Color Syntax:** Switch Color Syntax on / off
- **Keywords:** Basic keywords are marked blue
- **Numbers and Strings:** These values marked green.
- **Remarks:** Code lines containing *are marked red from the occurrence of the* letters
- **Textsize:** The size of the font
- **Font:** The type of the font

Cuelist Tasks

Tasks contain the instructions for the attached devices and the program flow. V-Control knows three types of tasks, Cuelists, Scripts and Timelines. Cuelists are very easy to use without real programming. Scripts contain (native compiled) Basic scripts, which makes them very powerful for all kind of applications. Timestrips are a kind of timeline and can be useful in Show Control.

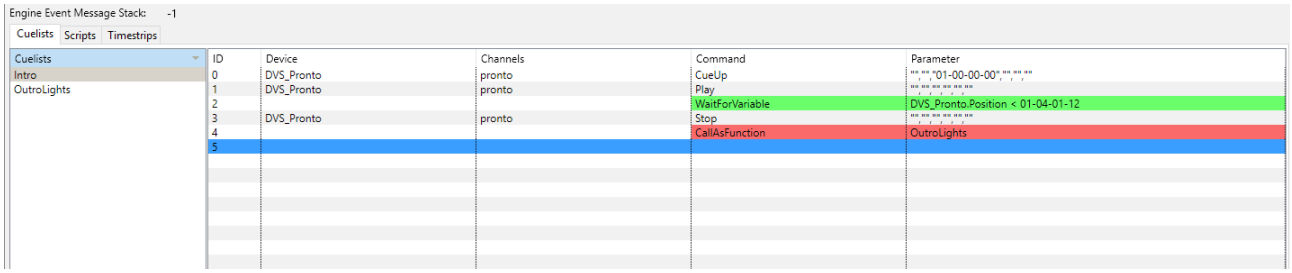
Multitasking and blocked Devices

V-Control can run many tasks at the same time. If multiple tasks running together, the programmer has to take care that not more then one task at a time is using the same resource. A resource is a channel or a device.

If one task is accessing a device, and another task wants to access the same device at the same time, the first task wins, the command of the second one is stored in a queue and performed if the device is free.

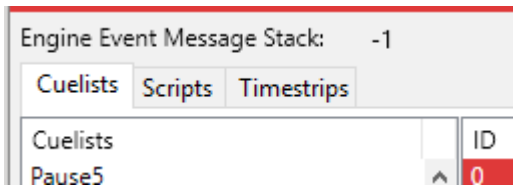
Cuelists

In Cuelists, the commands stored in a table that is processed from top to bottom. It is possible to branch to other tasks and call them as function or thread, but conditional branching is available in Scripts only.

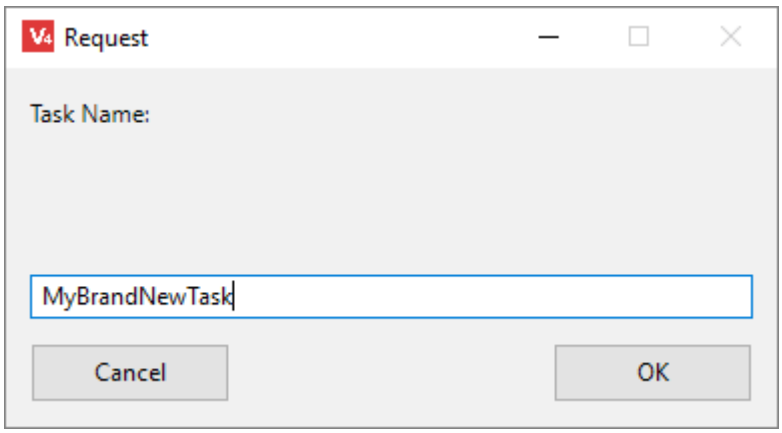


Create Cuelists

To Create a Cuelist, select Cuelists in the Task area.

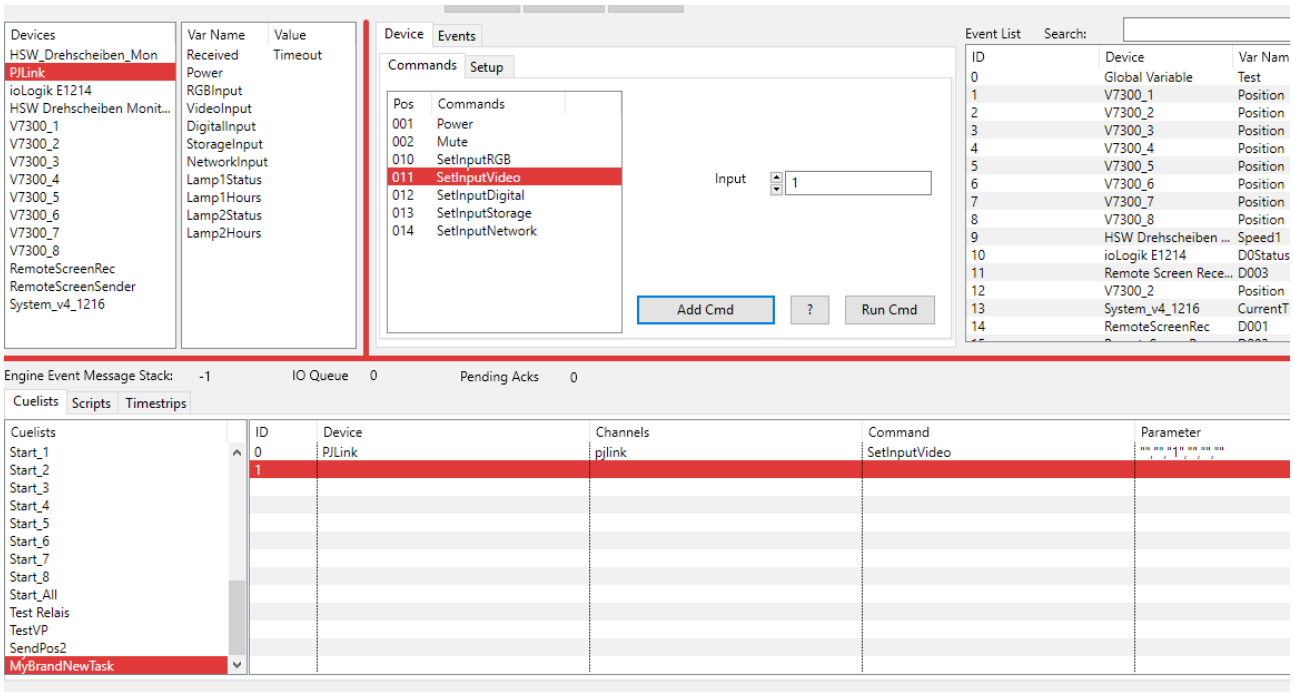


Then select *Tasks -> Add* in main menu or via Cuelist context menu (right click) select *Add Task*. Alternatively the *Add Task* tool button or Ctrl+A does the same job.

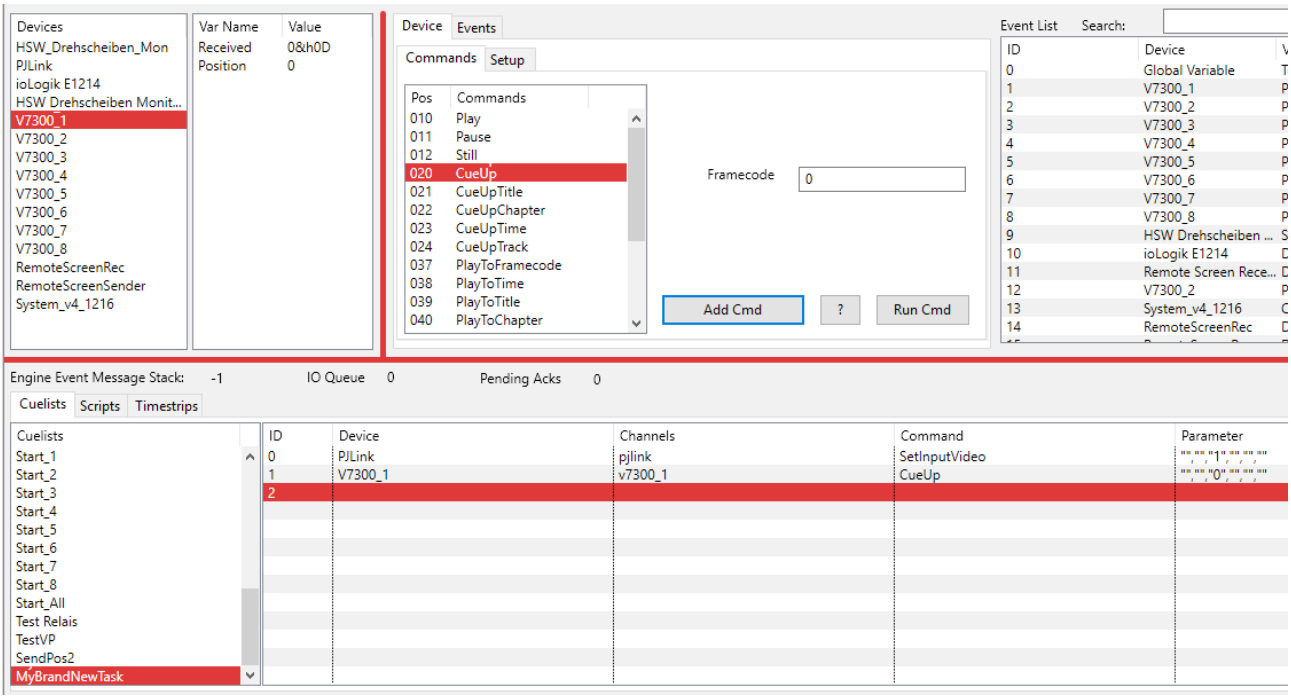


Each task needs a unique name that is entered in the dialog box as shown above. Then we have a new (Cuelist) task that is selectable and ready to take the instructions.

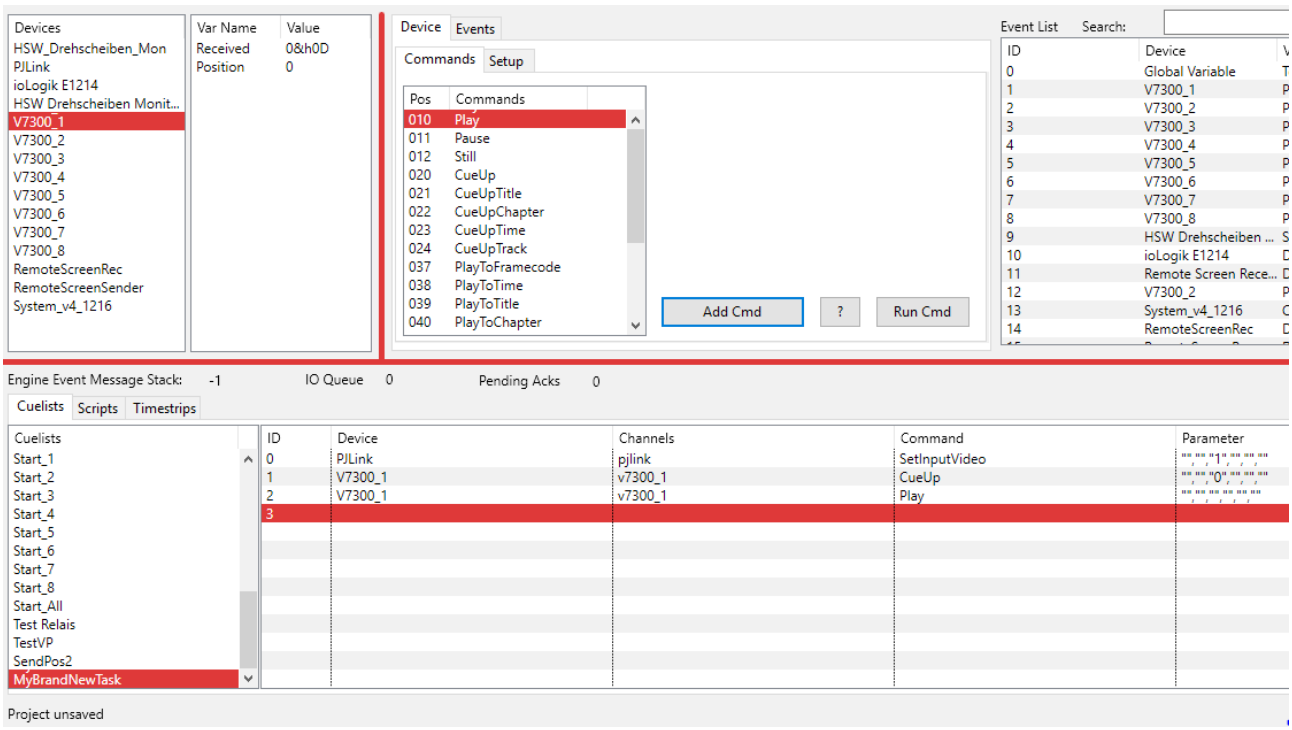
So lets add the first command. In the *Device List* select a device (PJLink in this case). Select a Command in the *Command List* (SetInputVideo here) and change the parameter to match your needs. Click the *Add Cmd* button to add the command in the list.



The next command is a CueUp instruction for a Player. Same procedure here. Select the Device, select the command, enter a parameter and click *Add Cmd*.



Finally we need a Play command



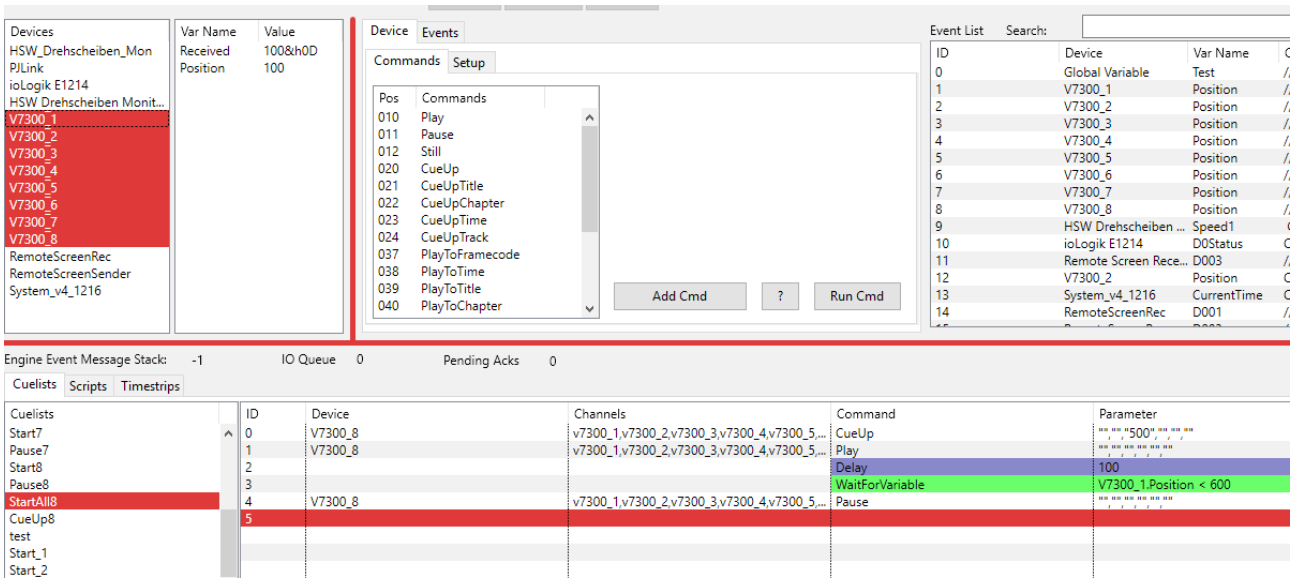
Now the task is finished. The projector switches to a video input, and the player starts playing from the beginning.

To start this task switch to *Run* mode, select the task and click the *Run Task* button.

Send commands Synchronous to more then one Device

This is only possible if all the involved devices use the same driver. If you have a couple of players, all from the same type, then you can select some or all of

them in the *Device List*. The Command that is send to the Players is generated by the last selected device.



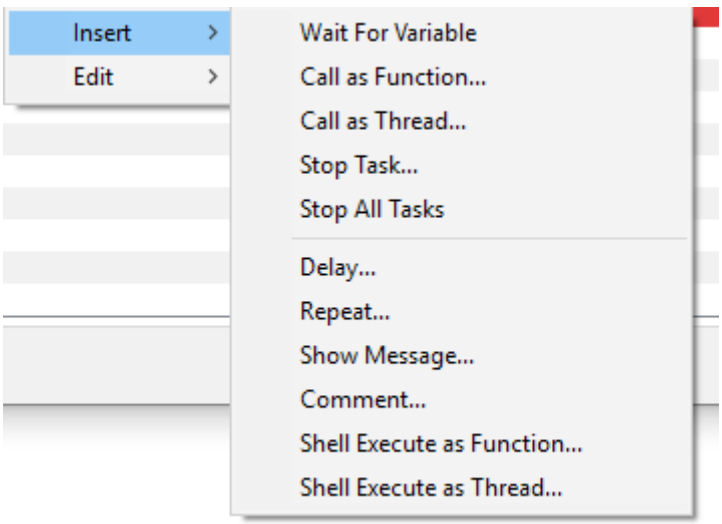
Notice that the Channel Parameter in the Cuelist now has all the channels of the involved devices. So the command string is generated by V7300_8, but send to all selected devices.

Edit a Cue

To edit a Cue, double click on it. If the Cue contains a control command for a device, the “Add Cmd” button changes to “Update”. You can now modify the command and press the “Update” button“

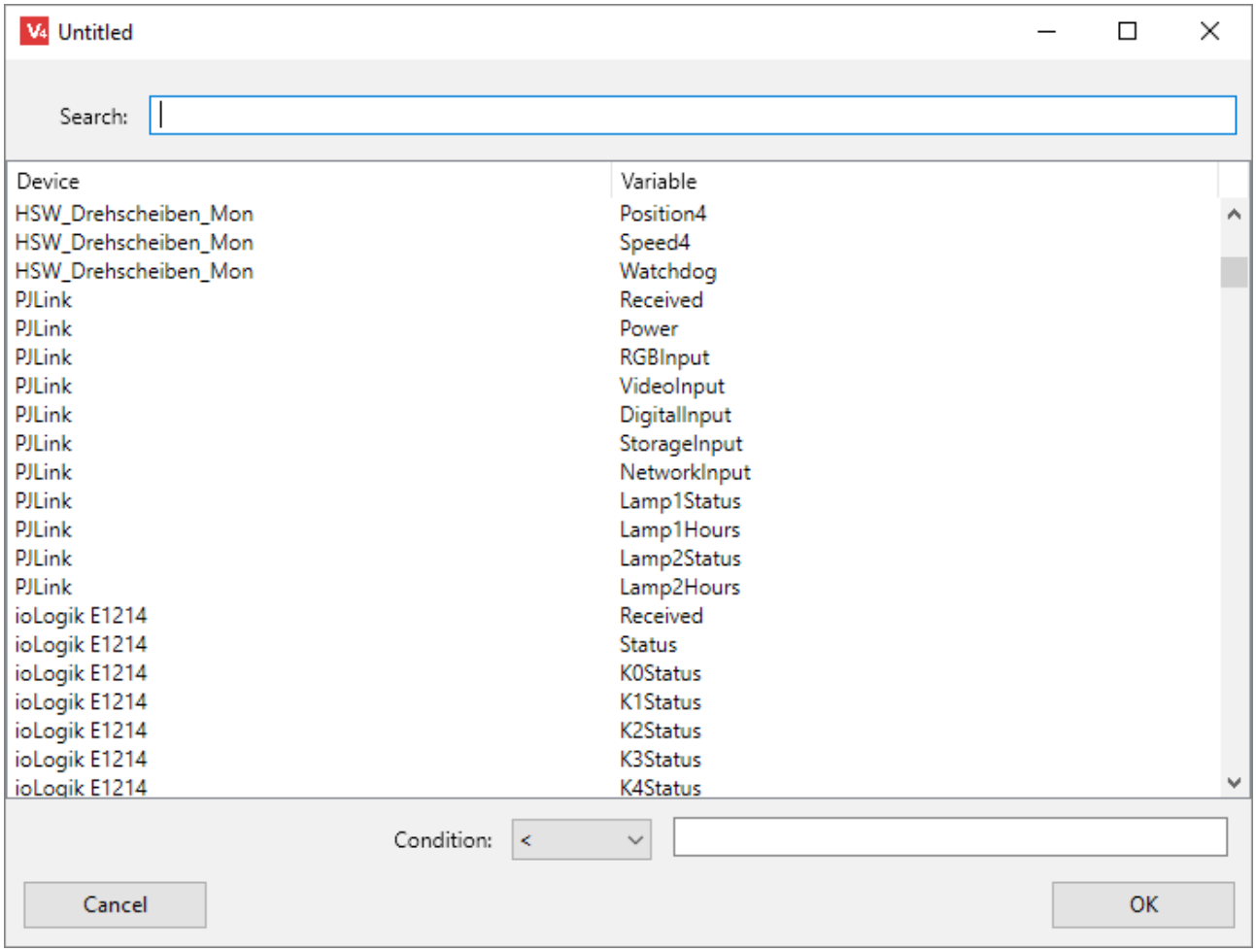
Non Device Commands

In the example above we saw how to add commands to a cuelist that concerns devices. But with only that, a cuelist is to limited. If you right-click into the cue list, you will get a context menu with further commands.



Wait For Variable

One option to control a cuelists flow is the WaitForVariable command. If inserted, a window pops up to select the variable to wait for.

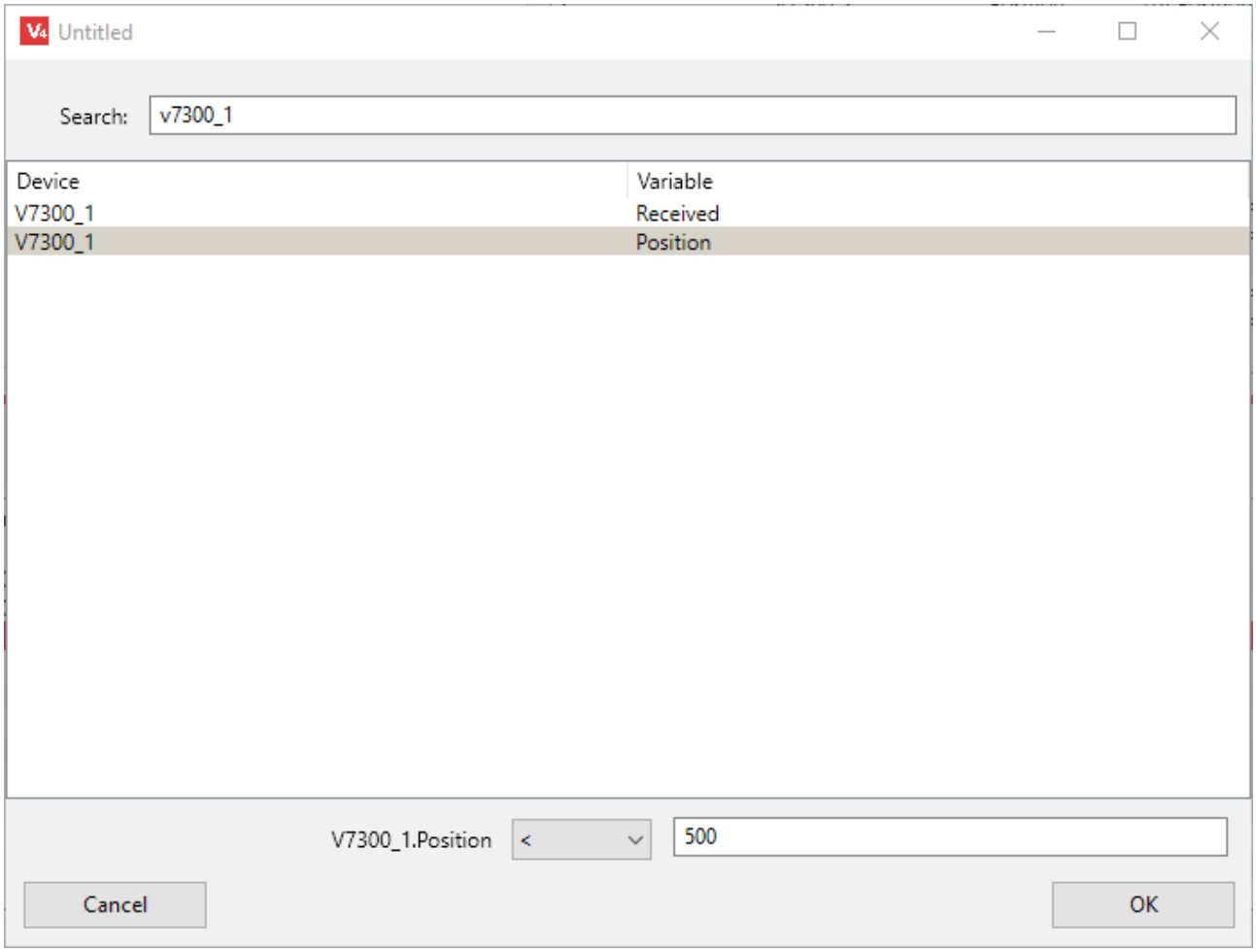


Let's complete the example above with the “MyBrandNewTask” cuelist. Remember that we send the player a CueUp command and a Play command. What's missing is a condition to stop playback and switch back the projectors input to another source.

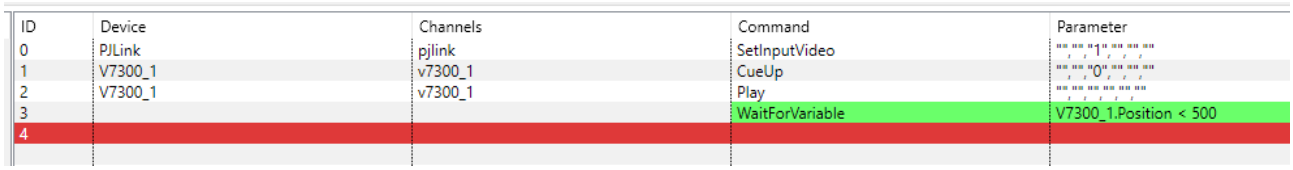
In the *Search* field type the device name (V7300_1). now only variables are shown that belong to this device. Select the variable *Position*. Next we need a condition how long to wait for this variable.

At the bottom of the window there is the selected variable (V7300_1.Position in this case) and a drop down box with comparing parameters.

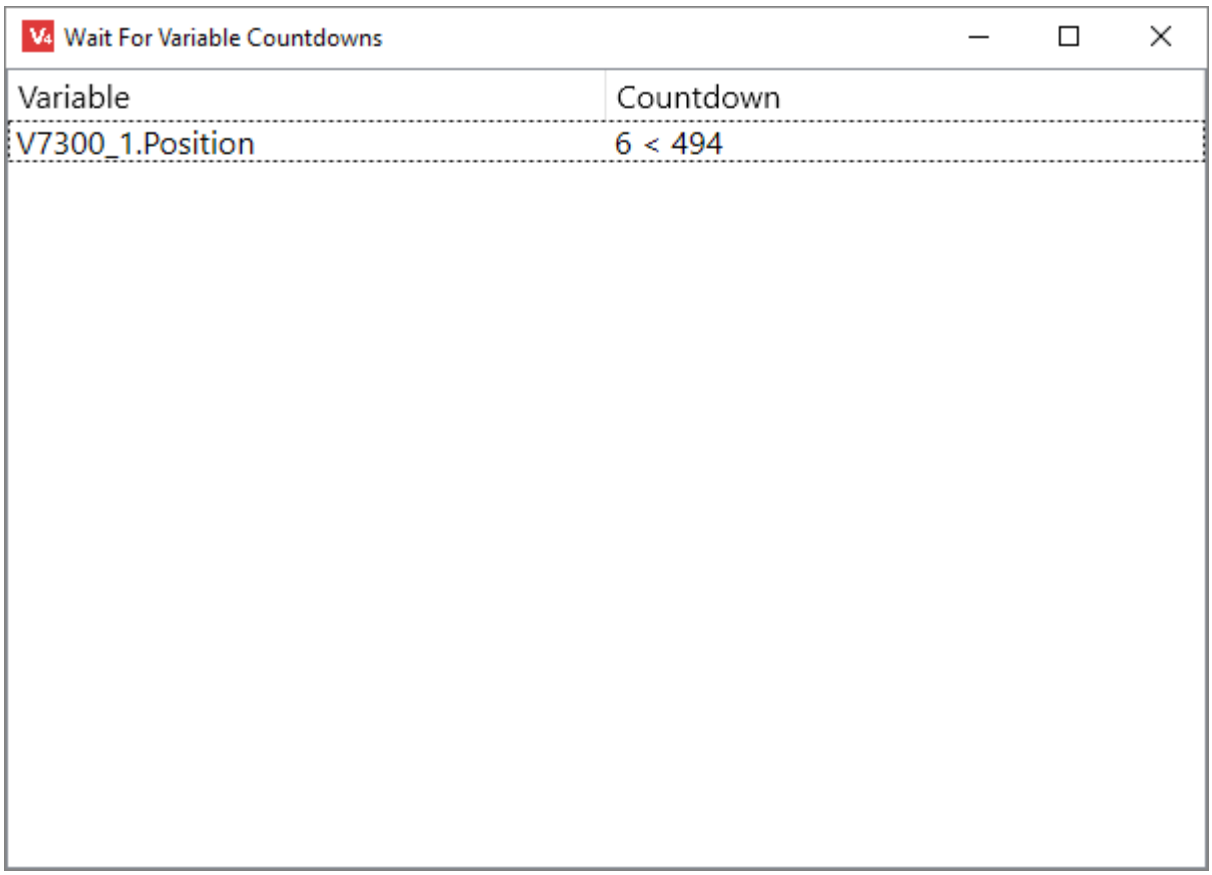
- < : Wait while less the value right
- > : Wait while more the value right
- <= : Wait while less or equal the value right
- >= : Wait while bigger or equal the value right
- <> : Wait while not equal the value right
- = : Wait while equal the value right



enter a value in the field right and click OK



If *Show WaitForVariable Countdowns* is set in the Options->System dialog, the a Countdown window appears showing how long the cue will wait.

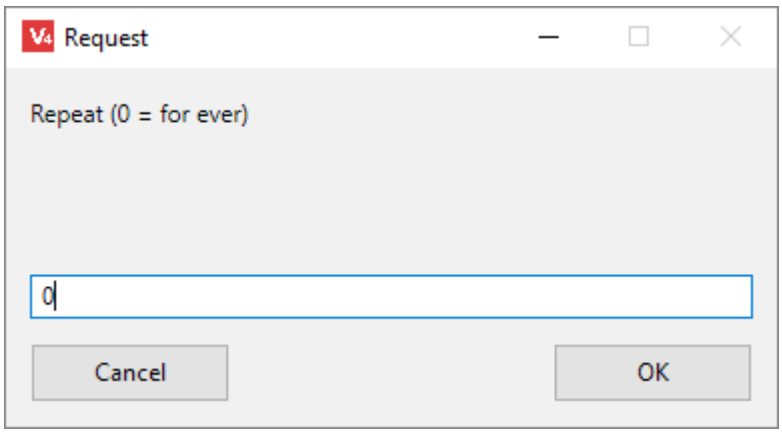


Now lets complete the task and stop the player and switch the input of the projector.

ID	Device	Channels	Command	Parameter
0	PJLink	pjlink	SetInputVideo	"","","1","","",""
1	V7300_1	v7300_1	CueUp	"","","0","","",""
2	V7300_1	v7300_1	Play	"","","","","",""
3			WaitForVariable	V7300_1.Position < 500
4	V7300_1	v7300_1	Pause	"","","","","",""
5	PJLink	pjlink	SetInputDigital	"","","2","","",""
6				

Repeat

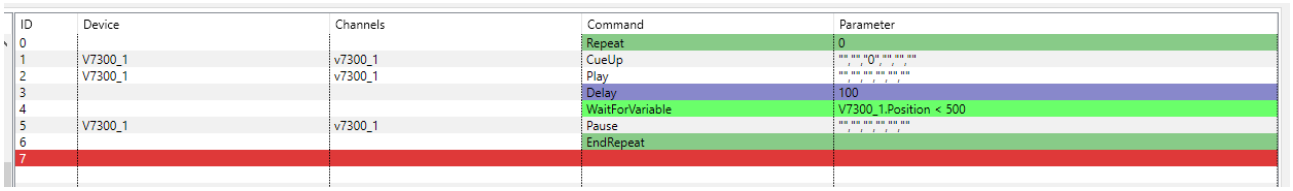
The Repeat command is used to build loops. The instructions between Repeat and EndRepeat are looped as many times as entered. The Repeat command is available via Cuelist context menu (right click -> *Insert* -> *Repeat*) or main menu (*Insert* -> *Repeat*). A dialog box asks for the number of repeats. To create endless loops, enter 0 as value.



Next, fill the part between Repeat and Endrepeat with instructions. It is possible to move the Repeat or EndRepeat command via Cut (*Ctrl+X*) and Paste (*Ctrl+V*).

In the example below we created an endless loop that plays a video from position 0 to position 500. The *Delay* between the *Play* command and the *WaitForVariable* command is necessary for the following reason: The player receives the two commands CueUp and Play. During this sequence, the device driver can not ask for the position, because the player is busy with the two commands. That means that the device variable *Position* still has the last value (500 or more). In that case the WaitForVariable Condition is true and the cue is skipped. One solution to avoid this is to set a small Delay, so the driver has time ti update the correct position. Another solution is to modify the driver. With the CueUp command we can set the Device Variable position to the value of the CueUp command. In this case we don't need the delay.

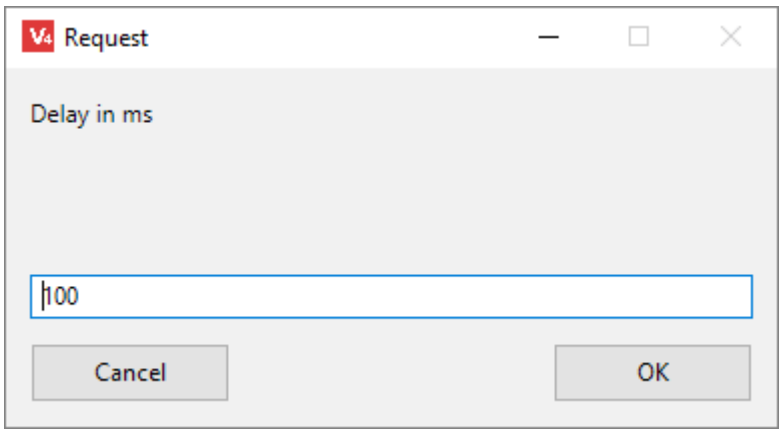
The loop sequence now looks like this:



ID	Device	Channels	Command	Parameter
0			Repeat	0
1	V7300_1	v7300_1	CueUp	"0"
2	V7300_1	v7300_1	Play	
3			Delay	100
4			WaitForVariable	V7300_1.Position < 500
5	V7300_1	v7300_1	Pause	
6			EndRepeat	
7				

Delay

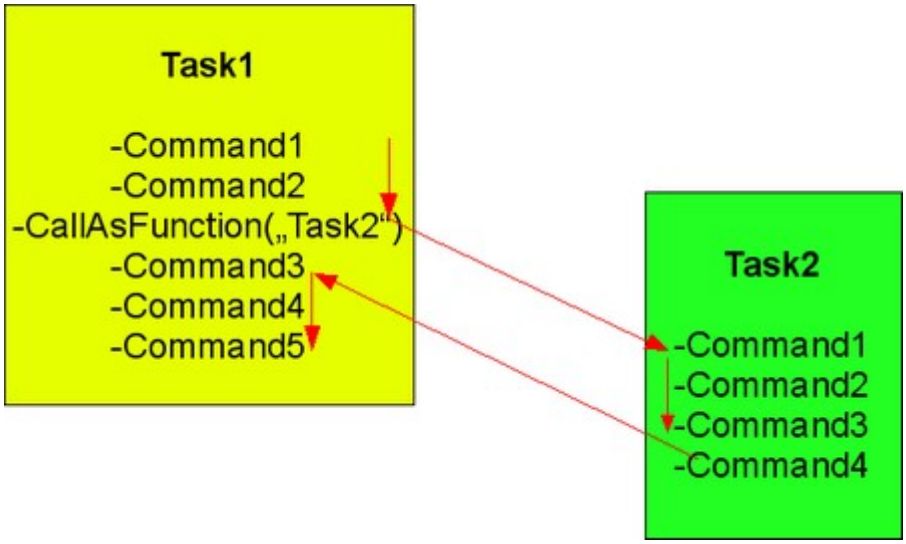
It might be necessary to have a Delay between two commands to pause the task for a given amount of time (see example above). In the main menu select *Insert -> Delay* or use the context menu. The Delay is specified by the value in milliseconds.



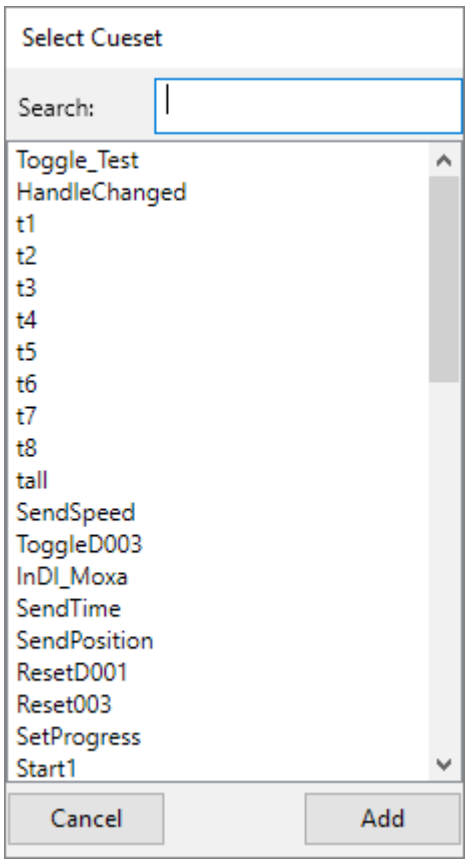
Delays displayed with a violet background in the list.

Call as Function

Even in complex projects it makes sense to split the whole project into several subroutines. This makes maintenance easier and gives a better overview over the projects structure. Every Task, whether a Cuelist or Script, can be a function. If a task is executed, it is usually processed from top to bottom. A *CallAsFunction* branches to the task specified, finishes the called task and returns to the next command in the calling task.



To call a task as function, select *Insert -> CallAsFunction* in main menu or Cuelist context menu (right click *Insert -> CallAsFunction*). That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be called.

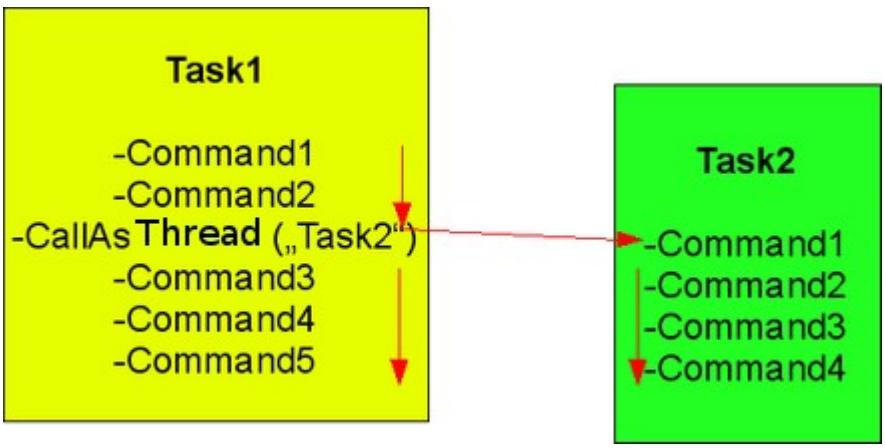


Finally, the new CallAsFunction command in´s inserted in the Cuelist.

ID	Device	Channels	Command	Parameter
0			CallAsFunction	SetProjectorInput
1			Repeat	0
2	V7300_1	v7300_1	CueUp	
3	V7300_1	v7300_1	Play	
4			Delay	100
5			WaitForVariable	V7300_1.Position < 500
6	V7300_1	v7300_1	Pause	
7			EndRepeat	
8				

Call as Thread

In contrast to CallAsFunction, a CallAsThread instruction launches the new task parallel to the calling task. The calling task is not paused during the time, the called task is executed. The programmer / operator has to take care that the two parallel running tasks don't use the same resources (devices / channels).

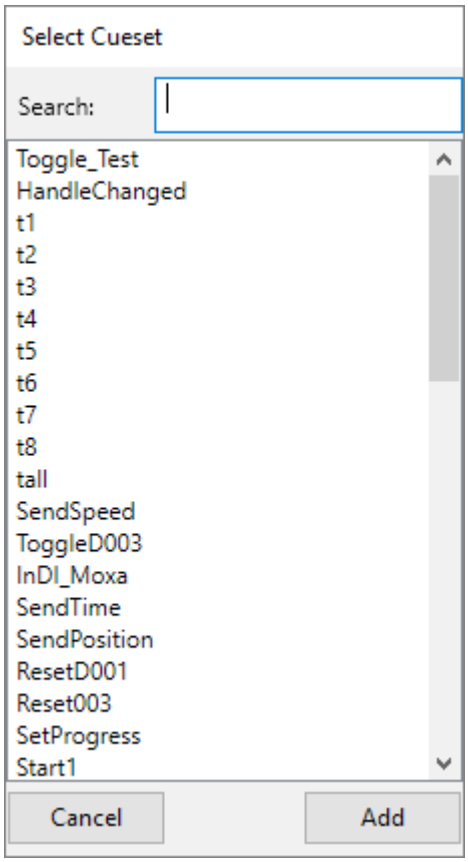


To create a CallAsThread instruction select *Insert -> CallAsThread* in main menu or use the Cuelists context menu (right click *Insert -> CallAsThread*). That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be called.

Stop Task

It might be necessary to stop a running task before it terminates regularly. This can be done manually via *Stop Task* tool button or as instruction from a running task. Assuming there is an endless loop working with two Players. Now the Players needed for some other action, but it is impossible to use them as long as the endless loop is running. With the StopTask command it is possible to terminate the endless loop and then use the Players in another task.

The command is available via main menu *Insert -> Stop Task* or the Cuelists context menu. That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be stoped.

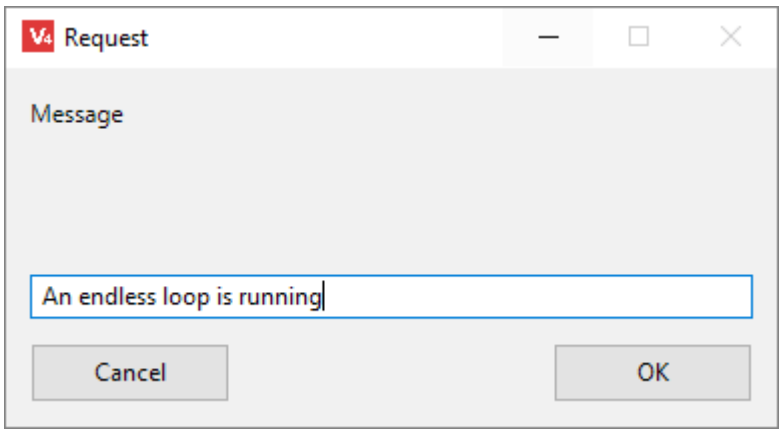


Stop All Tasks

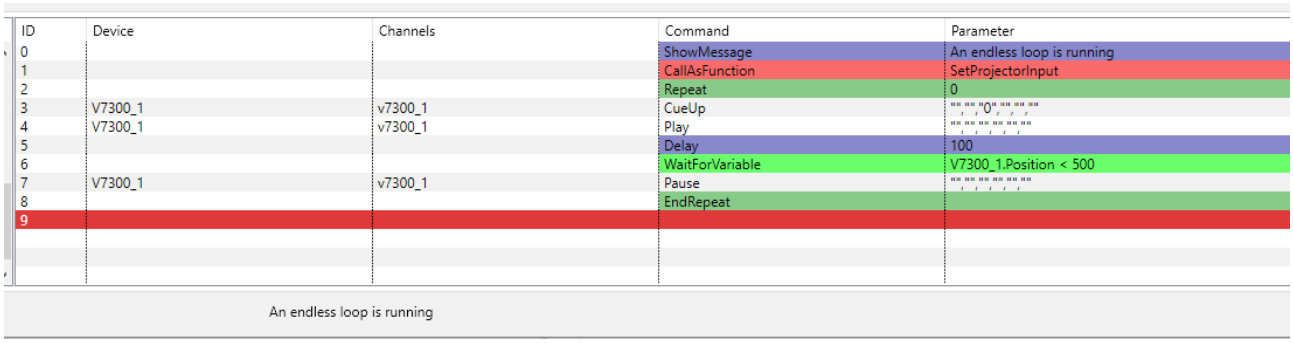
same as StopTask, but stops all tasks except the one that is calling this command.

Show Message

The ShowMessage command is used to display a message in the status bar. Via main menu *Insert -> Show Message* or Cuelist context menu this command is available.

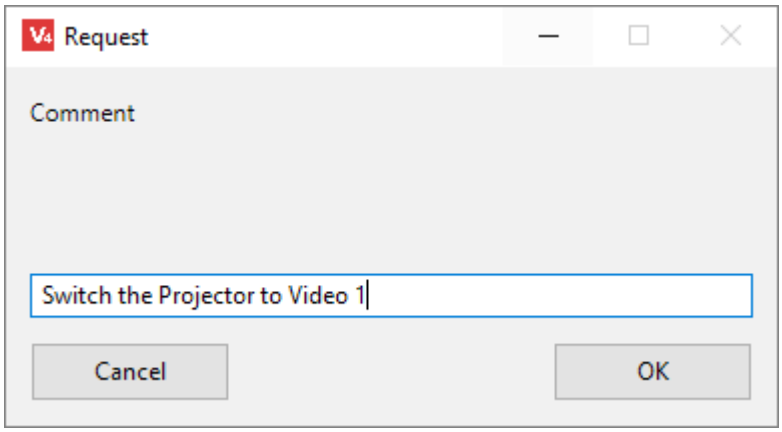


If the instruction is executed, the message appears in the status bar.

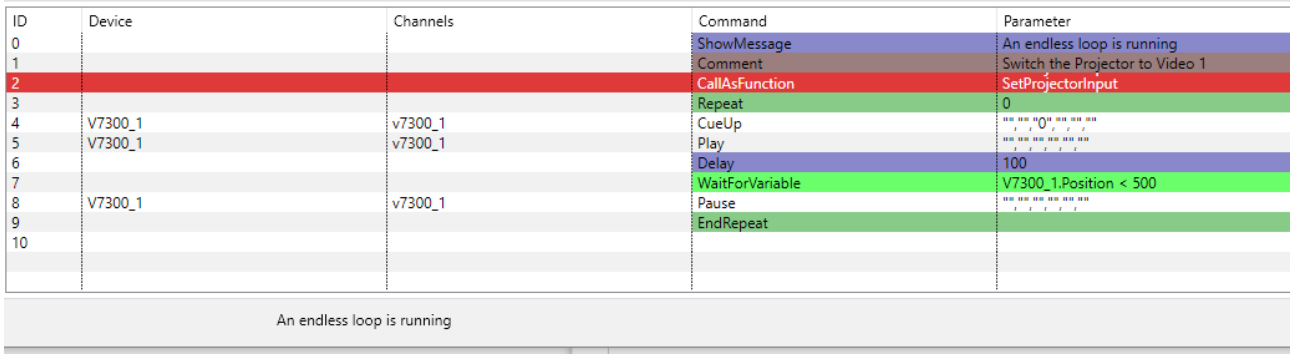


Comment

Insert -> Comment allows to insert a comment to the list. A comment is only for documentation purpose and should describe what happens in this part of the Cuelist.



A comment appears in the Cuelist with a dark red background.

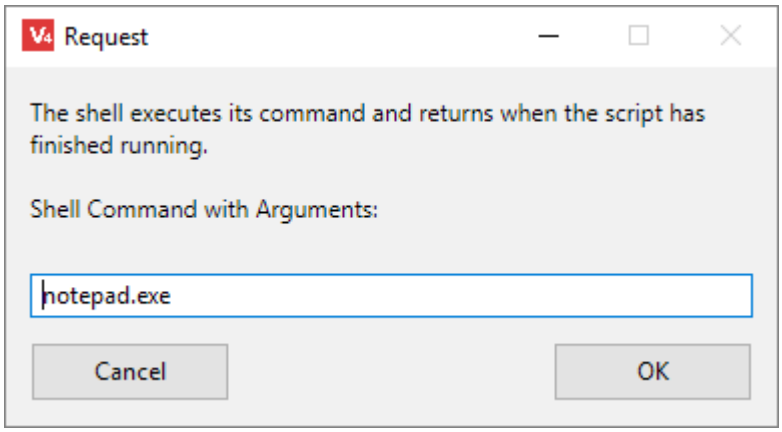


A comment has no functionality for the script. During task execution it is ignored. The only purpose is to make notes for the user to have a documentation for the script.

Shell Execute as Function

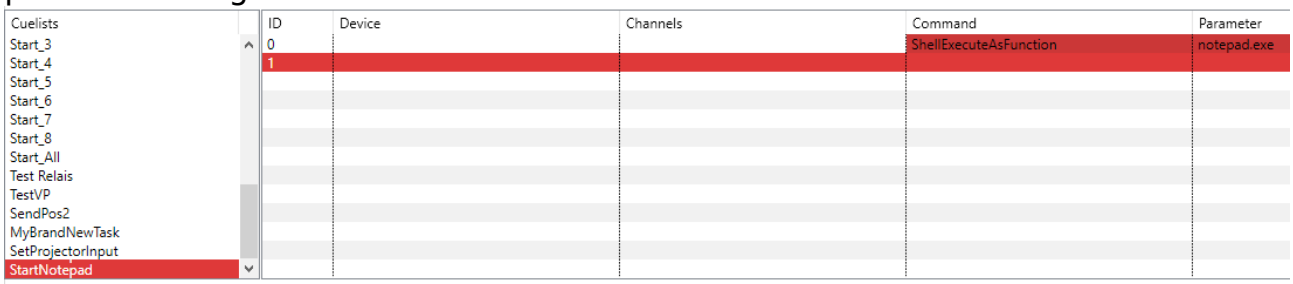
The ShellExecuteAsFunction command is used to launch shell commands / scripts. It works similar to CallAsFunction, and is available via main menu *Insert -> Shell Execute As Function* or the cuelist context menu.

A dialog asks for the shell command and optional parameters:



ShellExecute commands are displayed with red background in the cuelist.

In this example notepad.exe is called. If programs or scrips needs to be executed that are not in the operating systems path variable, the the complete path must be given here.



Shell Execute as Thread

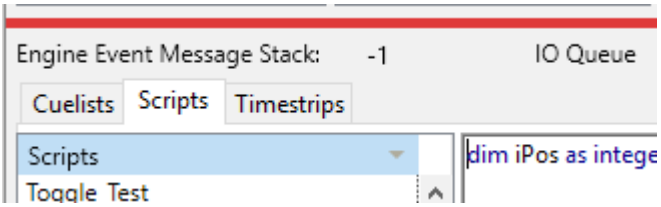
ShellExecuteAsThread launches programs / scripts as thread (see Shell Execute as Function). V-Control launch the shell command and returns immediately, without waiting for the shell command to finish.

Scripts

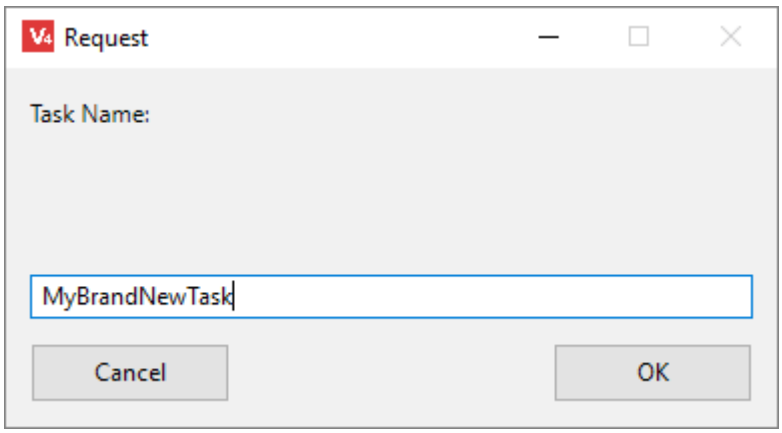
These kind of tasks are much more powerful than Cuelists. Scripts offer access to the integrated Basic language. The modern compiler compiles native code for every supported platform. V-Control Basic enhances the standard Basic by commands giving access to the Device Driver and Tasks as well as to globally shared variables. In the Compiler section, there is a list of all available commands and functions.

Create Scripts

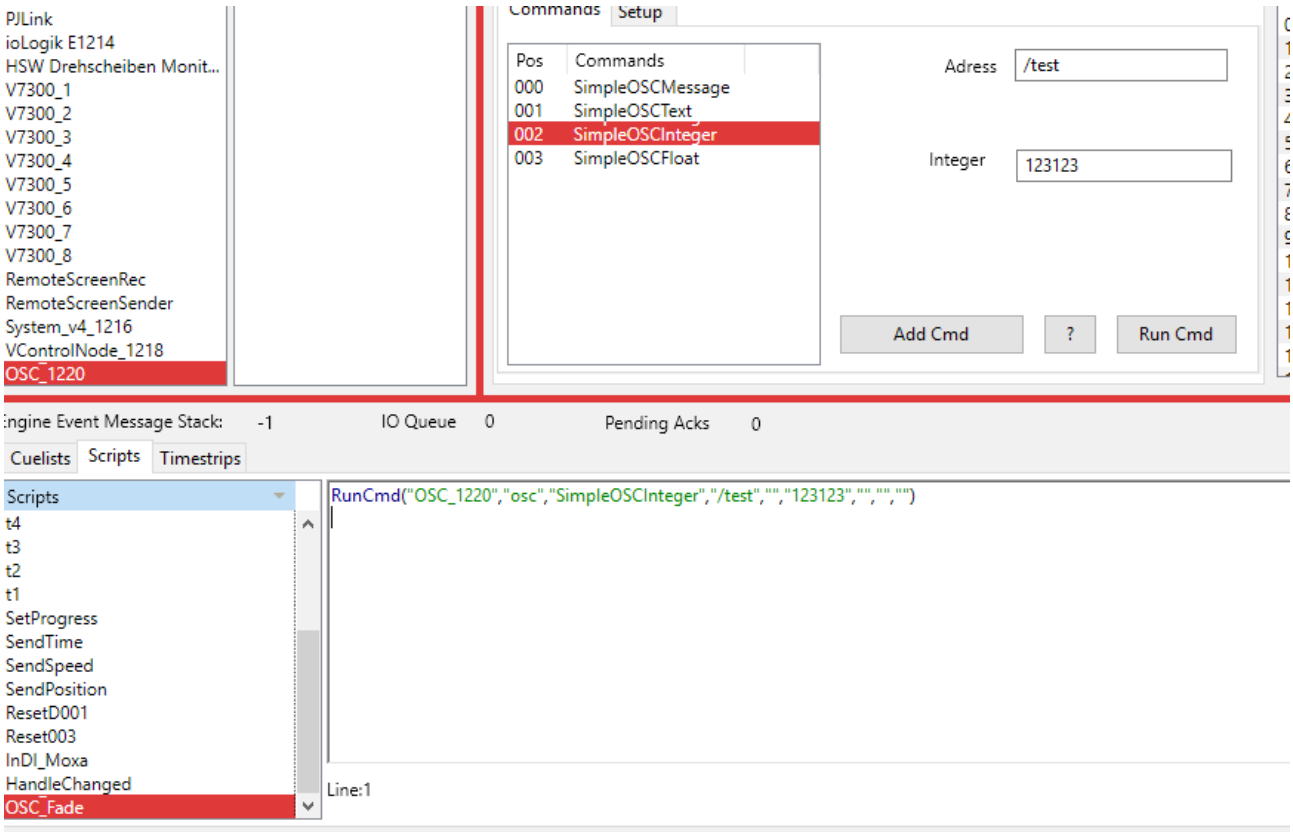
To create a Script switch to Script View.



Then select *Tasks -> Add* in main menu or via Cueset list context menu (right click) select Add Task. Alternatively the click the *Add Task* tool button.



Now the task has to be filled with instructions. Generally this is the same procedure than programming Cuelists. In the following example we send an OSC (Open Sound Control) message to a device. Select the device in the *Device List*, select the command in the *Command List* and enter the parameter. Then click the *Add Cmd* button.



The command generated has this structure:

```
RunCmd("Device Name","Channel Name","Command Name","Device
Parameter 1","Device Parameter 2",
"Command Parameter 1","Command Parameter 2","Command Parameter
3","Command Parameter 4")
```

Parameters that are empty shown as “”

So the resulting entry is

```
RunCmd("OSC_1220","osc","SimpleOSCInteger","/
test","","123123","","","")
```

Next step is to modify this command. Assume that the *SimpleOSCInteger* command controls a fader value, and the fader should move from 0 to 100 in 2 seconds. The script can be something like this:

Dim v as Variant creates a local variable v. The type Variant is very flexible. It behaves as an integer if used so. But the content of the variable is converted to string if a string is needed. The **For** loop accepts only Integers as loop iterator, but the **RunCmd** Function accepts only string parameters. In both cases we can use a variant variable.

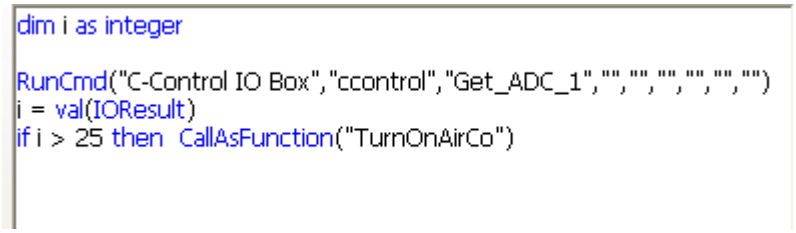
Here is an example with Integer and String Variables:

```
Dim i as Integer
Dim s as String

for i = 0 to 100
     s = str(i)

RunCmd("OSC_1220","osc","SimpleOSCInteger","/test","",s,"","","")
     Delay("20")
Next
```

The **str** Function converts an Integer to String

IOResult

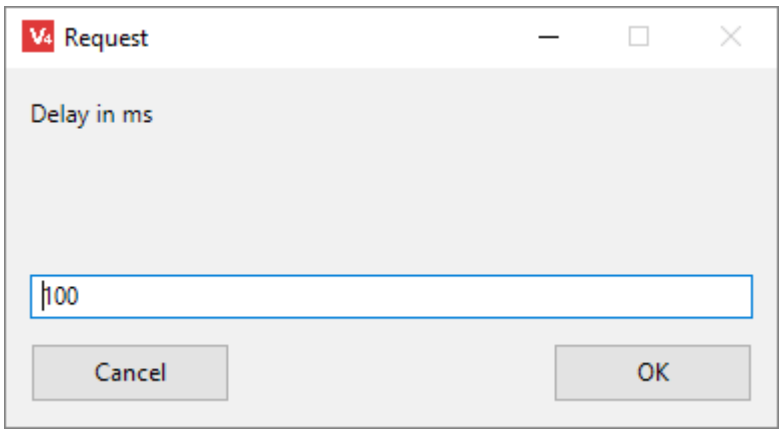
Every RunCmd instruction changes the variable IOResult. IOResult does not have to be declared, it is a (string) variable that is part of every Script and contains the result of the last RunCmd instruction.



In the example above a `Get_ADC_1` command is sent to a C-Control I/O interface. The result of the command is stored in the variable IOResult. The value of IOResult is (in this case) the measured value of analog digital converter (ADC) 1 of the C-Control interface. Because the type of IOResult is string, but the value is needed (for comparison) as integer, the line `i = Val(IOResult)` converts the string to an integer. If the measured value is bigger than 25, an air condition is switched on.

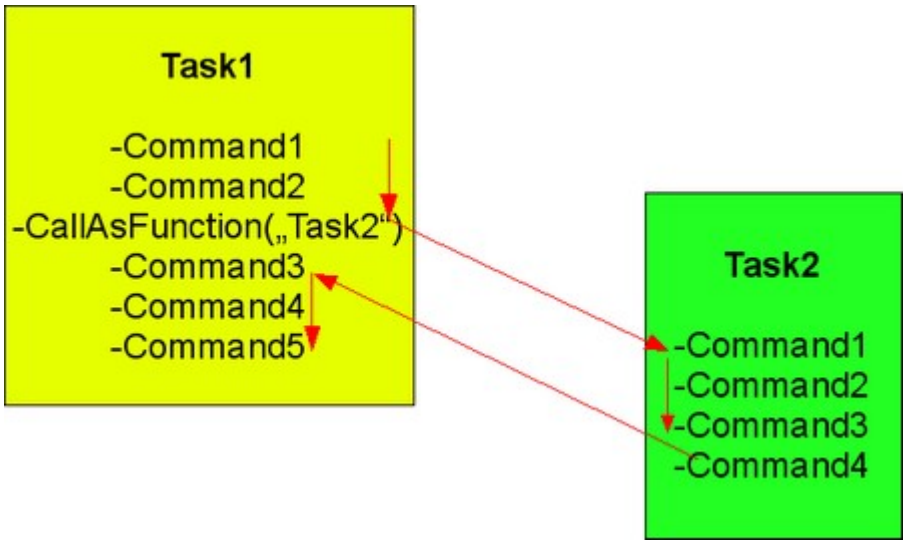
Delay

It might be necessary to have a Delay between two commands to pause the task for a given amount of time. In the main menu select *Insert -> Delay* or use the context menu. The Delay is specified by the value in milliseconds.

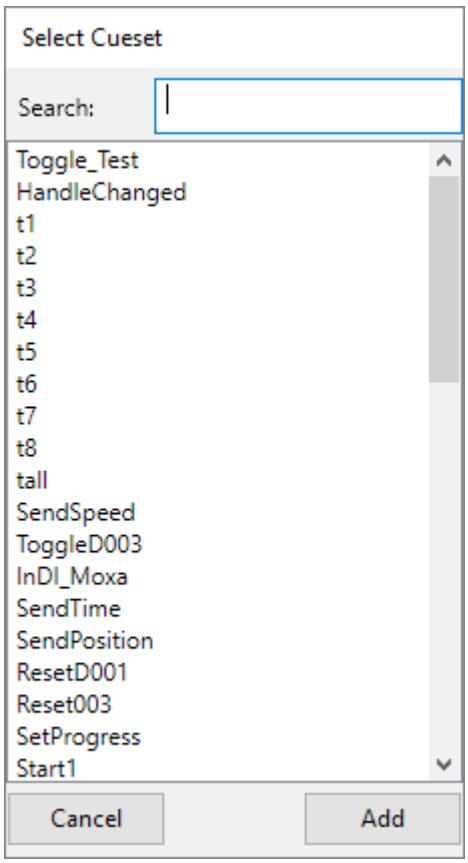


Call As Function

Even in complex projects it makes sense to split the whole project into several subroutines. This makes maintenance easier and gives a better overview over the projects structure. Every Task, whether a Cuelist or Script, can be a function. If a task is executed, it is usually processed from top to bottom. A CallAsFunction branches to the task specified, finishes the called task and returns to the next command in the calling task.

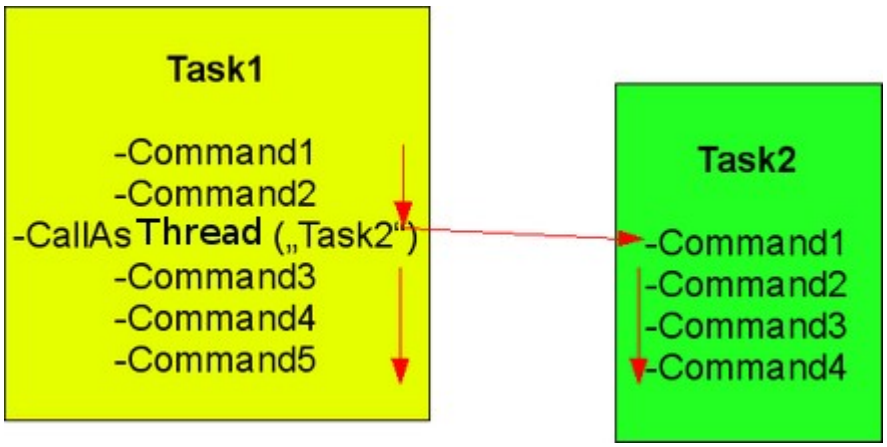


To call a task as function, select *Insert -> CallAsFunction* in main menu or Script Editor context menu (right click *Insert -> CallAsFunction*). That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be called.

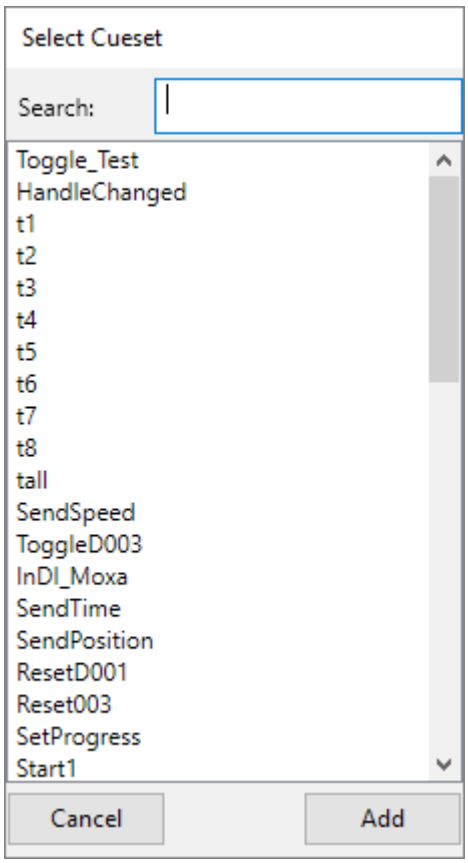


Call As Thread

In contrast to CallAsFunction, a CallAsThread instruction launches the new task parallel to the calling task. The calling task is not paused during the time, the called task is executed. The programmer / operator has to take care that the two parallel running tasks don't use the same resources (devices / channels).



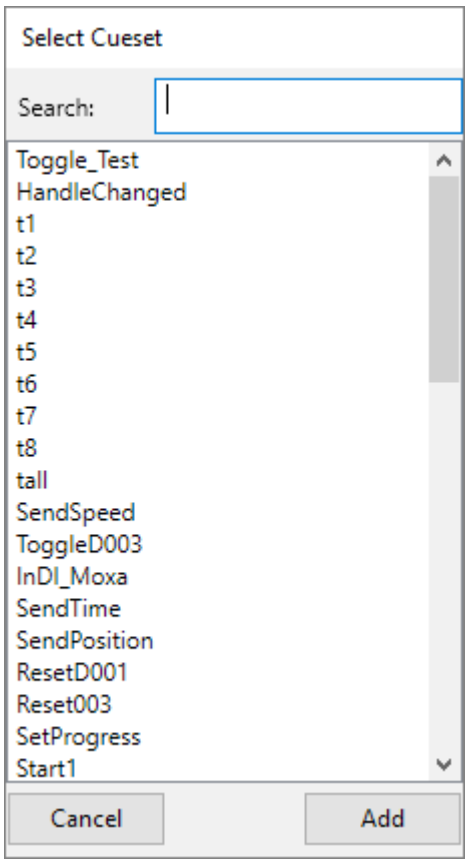
To create a CallAsThread instruction select *Insert -> CallAsThread* in main menu or use the Script Editor's context menu (right click *Insert -> CallAsThread*). That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be called.



Stop Task

It might be necessary to stop a running task before it terminates regularly. This can be done manually via Icon / Toolbar) or as instruction from a running task. Assuming there is an endless loop working with two DVD Players. Now the DVD Player are needed for some other action, but it is impossible to use them as long as the endless loop is running, because they are blocked. With the StopTask command it is possible to terminate the endless loop and then use the DVD Players in another task.

The command is available via main menu *Insert -> Stop Task* or the Script Editor's context menu. That opens the Task Selector presenting all tasks (Cuelists and Scripts). Choose the task that has to be stoped.



Show Message

The ShowMessage command is used to display a message in the status bar. Via main menu *Insert -> Show Message* or Script Editor's context menu this command is available.

Local Variables

Local variables are only available in the task where they have been declared. While declaring a variable, its type is set. The following types are available:

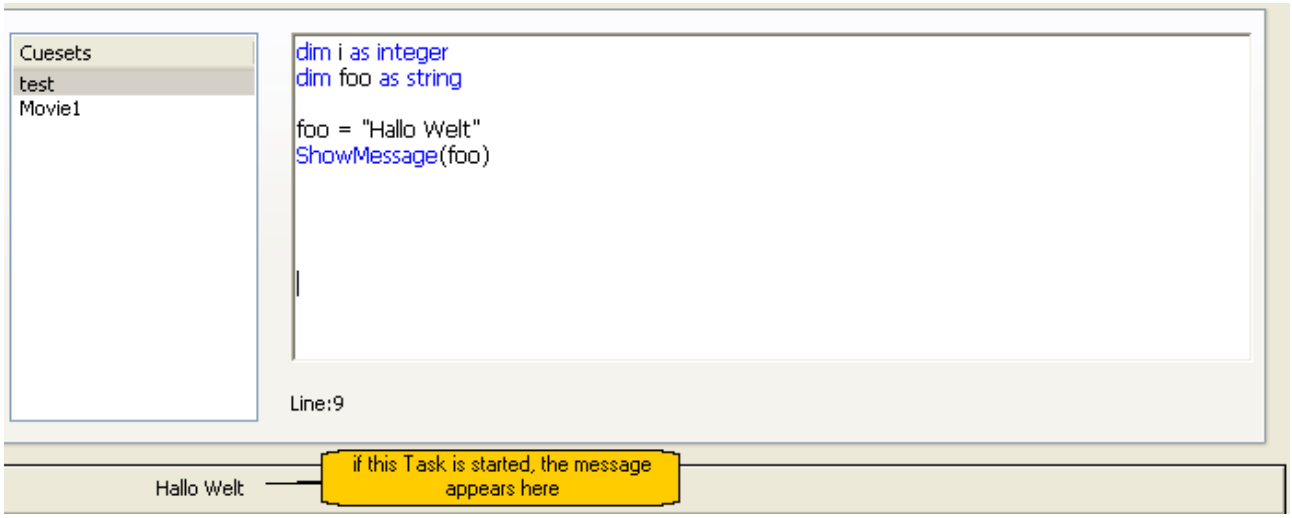
- Integer (32 Bit)
- Single (16 Bit integer)
- Double (64 Bit Floating Point)
- Boolean (True or False)
- String (e.g. “Hello World”)
- Variant (any)

It is possible to build arrays from all those types.

A variable is declared by the Dim statement. Variable names must start with a letter and must not contain special characters. Here some examples:

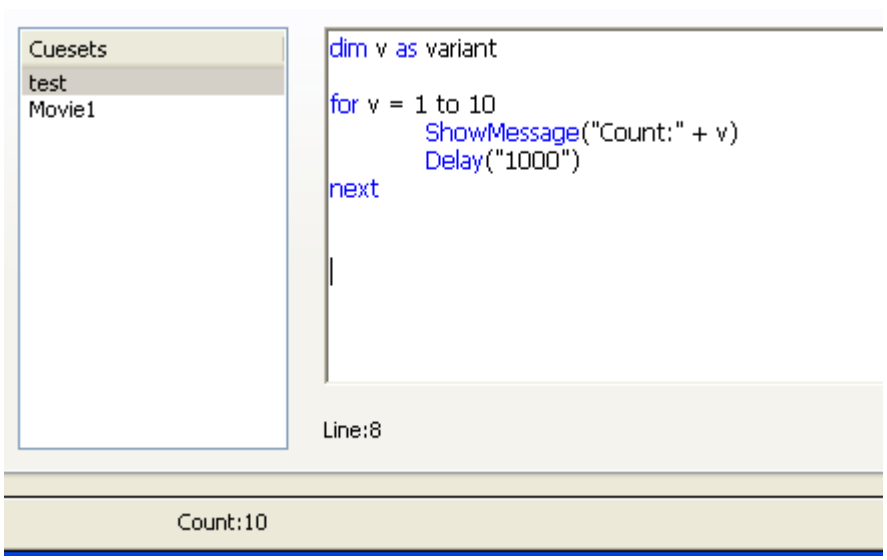
- `Dim A1 As Integer` correct
- `Dim Ä1 As Integer` incorrect
- `Dim 1A As Integer` incorrect

- `Dim A 1 As Integer` incorrect, space not allowed
- `Dim A_1 As Integer` correct

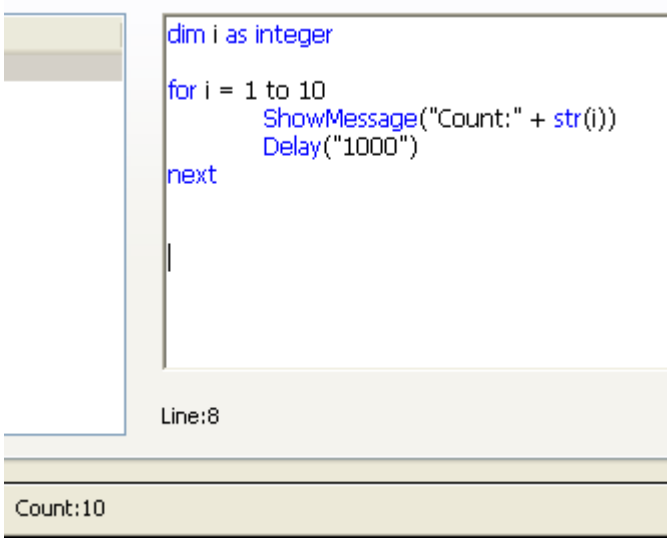


In the screenshot above, the Dim statement declares a String variable named foo. Then, the variable foo gets the value “Hallo Welt”. The content of foo is displayed by the ShowMessage command with foo as parameter. In this example, a line ShowMessage(“Hallo Welt”) would have the same result.

Variant variables are very special. Because they have no (or any) type, they are very flexible. A variant stores a value in different formats and “knows” which one is needed if there is an access to. In the example below a variant variable named v is declared:



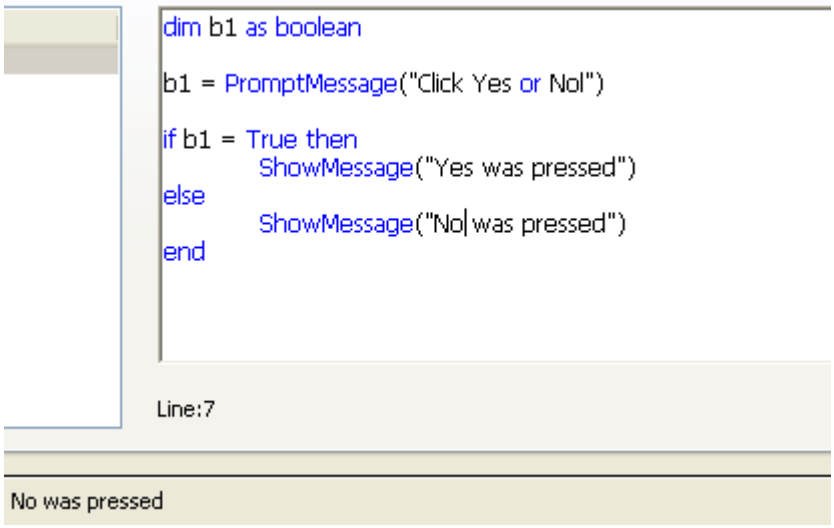
In the `For..Next` loop, an integer value is assigned to v. As discussed above, the ShowMessage command need a string variable as parameter, but the line ShowMessage(“Count:” + v) is working correct, because v knows that only strings are valid and present its content as string. So v is used as integer and as string in the same subroutine. A more structured solution for the same job is the example below:



At first, an integer variable i is declared. The beginning of the `For..Next` loop is unchanged, because the For..Next instruction expect an integer variable anyway. The command `ShowMessage("Count:"+ i)` will not work, because it is impossible to add an integer to a string. But V-Control Basic knows a function that converts an integer to string, the `Str(i)` command. The result of str(i) is "10", if i has the value 10. The string "10" comprising from ASCII code 49 ("1") and ASCII code 48 ("0").

An Overview of available basic functions is available in the Basic Language Reference Chapter.

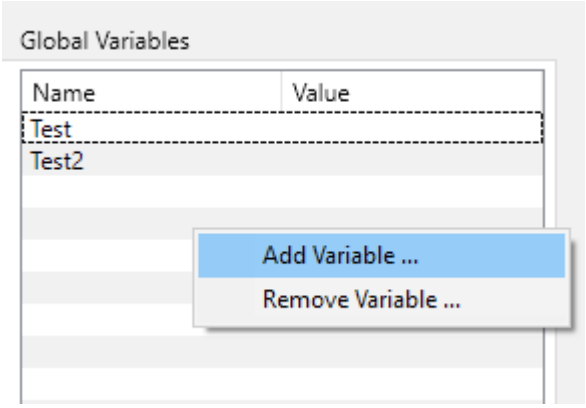
Another often used variable type is boolean. Boolean variables know to states, True or False (1/0).



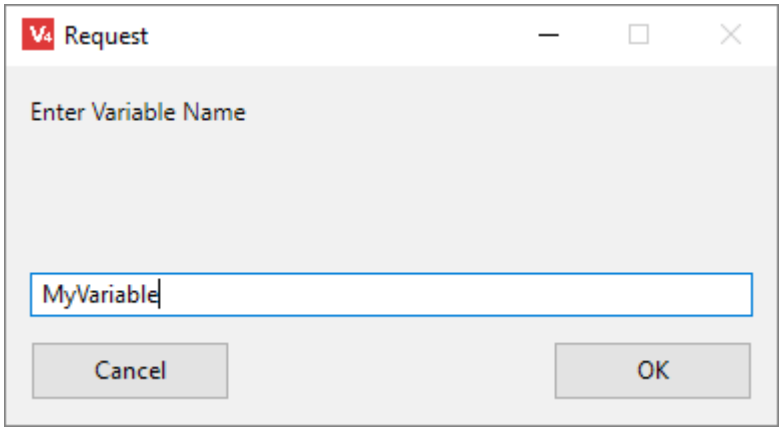
In this example, the variable b1 is declared as Boolean. The PromptMessage command need a boolean variable to return the result of the PromptMessage dialog. If the result is True, the OK button was pressed, if False, the user clicked the Cancel button. The if..then instruction check b1 and branch to the correct ShowMessage command.

Global Variables

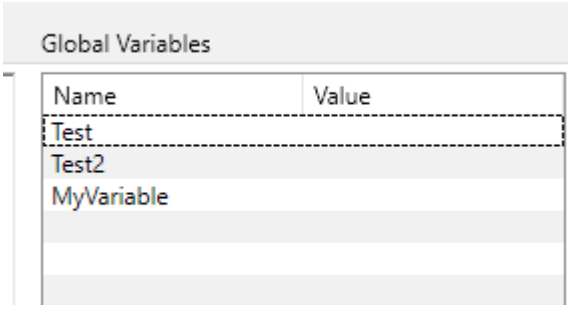
Global variables are accessible from every Script. It is possible e.g. to assign a global variable a value in task A, and check this value in task B. Global variables are from type string only. Because every Script has access to global variables, it is not declared in a Script. To declare a global variable, select *Cues -> Add Global Variable...* in the main menu or use the context menu of the Global Variables list.



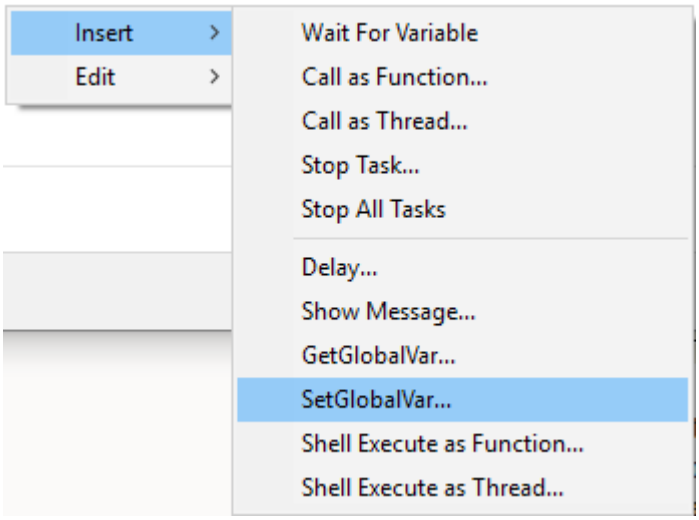
Enter the name of the variable in the dialog box:



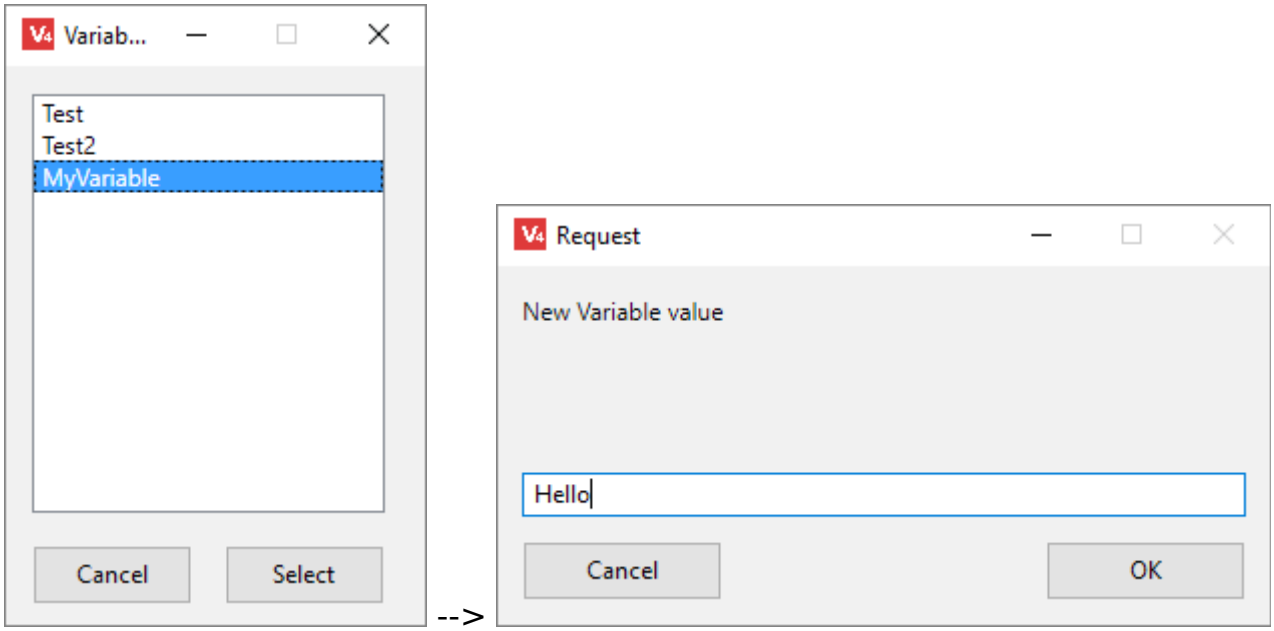
The new variable appears in the Global Variables list. At this point, the content of the new variable is an empty string “”.



To access the global variables, the commands *GetGlobalVar* and *SetGlobalVar* are used (available via main menu *Insert -> GetGlobalVar*, *Insert -> SetGlobalVar* or the Script Editor's context menu).



To assign a value to a global var, use the function *SetGlobalVar*. As shown in the picture above, it is available via Script Editor's context menu. In the following dialog (see below) select a variable from the list and enter a value for that variable:



The result is `SetGlobalVar("MyVariable","Hello"`

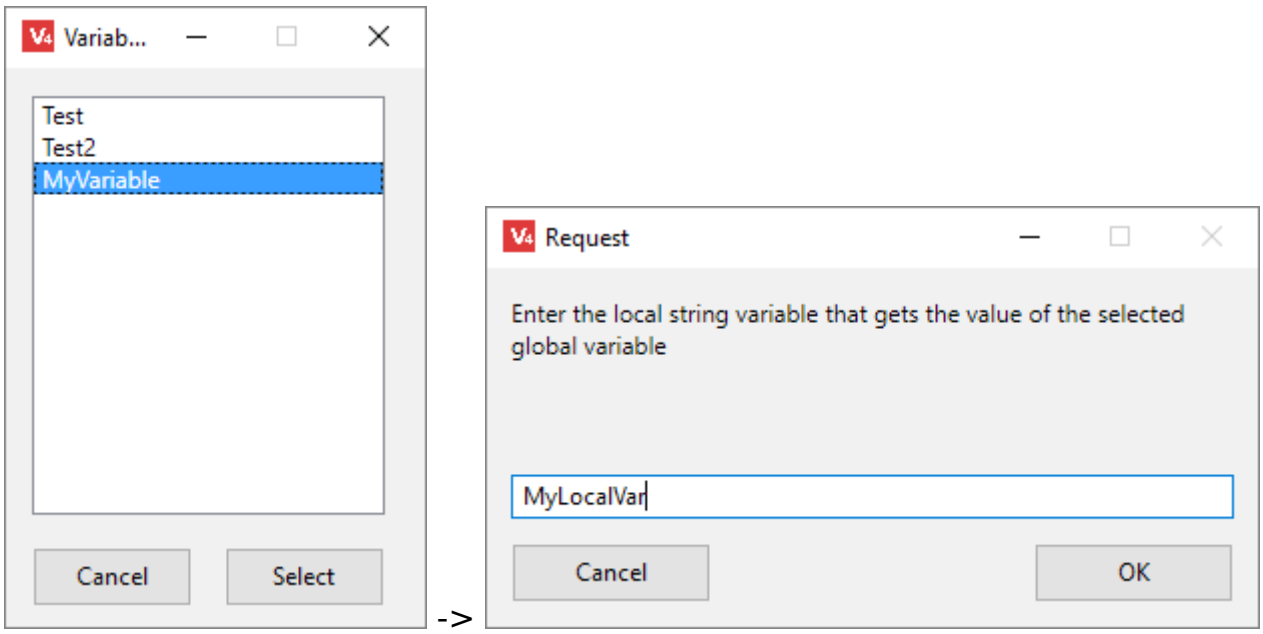
If this task is launched, the variable MyVariable gets the string “Hello” as content. To check this, run the task.

To read a global variable use *GetGlobalVar*. Instructions like “MyVariable” = “6” are invalid, because “MyVariable” is a string and not a variable. MyVariable = “6” is also invalid, because MyVariable is not declared in the Script and only (locally) declared variables are known. The solution is to declare a local variable of type string and assign the content of the global variable to that local one.

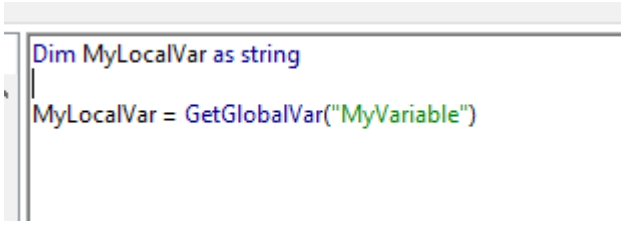
```
Dim MyLocalVar as string
```

Via main menu *Insert -> GetGlobalVar*, or the Script Editor's context menu a dialogue box with all available global variables pops up. If a variable is selected

a new dialogue box asks for the local variable, that gets the content of the global one.



The result looks like this:

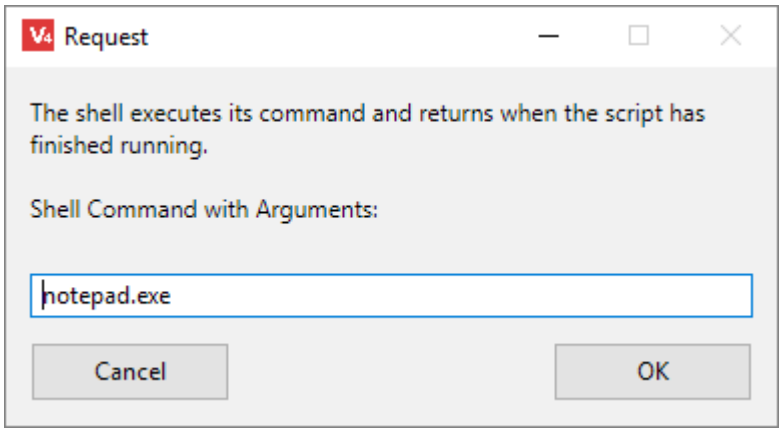


now, the content of the global variable is available in the Script and we can work with it.

Shell Execute As Function

The ShellExecuteAsFunction command is used to launch shell commands / scripts. It works similar to CallAsFunction, and is available via main menu *Insert -> Shell Execute As Function* or the context menu.

A dialogue asks for the shell command and optional parameters:



Shell Execute As Thread

ShellExecuteAsThread launches programs / scripts as thread (Shell Execute as Function). V-Control launch the shell command and returns immediately, without waiting for the shell command to finish.

Compiler

Basic Language Reference

Abs

```
Abs (Double) As Double
```

Returns the absolute value of the number specified.

```
Dim d As Double

d=Abs(23.9) //returns 23.9
d=Abs(-23.9) //returns 23.9
```

Acos

```
Acos (Double) As Double
```

Returns the arccosine of the value specified. The arccosine is the angle whose cosine is value. The returned angle is given in radians.

```
Dim d As Double

Const PI=3.14159265358979323846264338327950
d=Acos(.5) //returns 1.0471976
d=Acos(.5)*180/PI //returns 60
```

Asc

```
Asc (String) As Integer
```

Returns as an Integer, the ASCII value for the first character of a String.

```
Dim a As Integer

a = Asc("@") //returns 64
```

The Asc function returns the code point for the first character in the String passed. Characters 0 through 127 are the standard ASCII set. They will be the same on practically every platform. Asc returns the code point for whatever encoding the string is in. If the string is in MacRoman, you get the code point specified by MacRoman, and so forth.

If you need to get the ASCII code of the first byte of the string rather than the first character, use the AscB function.

AscB

```
AscB (String) As Integer
```

Returns as an Integer, the value for the first byte of a String.

```
MsgBox Str(AscB("a")) //returns 97
MsgBox Str(AscB("A")) //returns 65
```

The AscB function returns the code for the first byte in the String passed. If you need to get the character code of the first character of the string rather than the first byte, use the Asc function.

AscB should be used instead of Asc when the string represents binary data or when your application will run on a one-byte character set (such as the US system) and you want case-sensitivity.

Asin

```
Asin (Double) As Double
```

Returns the arcsine of the value specified.

```
Dim d as Double

Const PI=3.14159265358979323846264338327950
d=Asin(.5) //returns 0.5235988
d=Asin(.5)*180/PI //returns 30
```

The arcsine is the angle whose sine is value. The Asin function returns the angle (in radians) of the sine passed to it. To express the arcsine in degrees, multiply the result by 180/PI.

Atan

```
Atan (Double) As Double
```

Returns the arctangent of the value specified. The arctangent is the angle whose tangent is value.

```
Dim d as Double

Const PI=3.14159265358979323846264338327950
d=Atan(1) //returns 0.785398 (PI/4 radians)
d=Atan(1)*180/PI // returns 45
```

Atan2

```
Atan2 (Double, Double) As Double
```

Returns the arctangent of the point whose coordinates are x and y. The

arctangent is the angle from the x-axis to a line drawn through the origin (0,0) and a point with coordinates x, y.

```
Dim d as Double

Const PI=3.14159265358979323846264338327950
d=Atan2(1,0) //returns 1.57
d=Atan2(1,0)*180/PI //returns 90
```

BitAnd

The BitAnd method returns a UInt64 that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if both bits in the same position in the integer passed are 1. Otherwise, 0 is assigned to the bit position.

```
Dim i As Integer
i = BitAnd(5, 3) // returns 1
```

BitOr

The BitOr method returns a UInt64 that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if either of the bits in the same position in the integers passed are 1. Otherwise, 0 is assigned to the bit position.

```
Dim i As Integer
i = BitOr(5, 3) // returns 7
```

BitXor

The BitXor method returns a UInt64 that is the result of comparing each bit of the two integers passed and assigning 1 to the bit position in the integer returned if both bits in the same position in the integers passed are not equal. Otherwise, 0 is assigned to the bit position.

```
Dim i As Integer
i = BitXor(5, 3) // returns 6
```

BCDToTCString

Timecode from Devices with Sony 9-pin protocol and compatible usually comes in BCD format (4 Bytes FFSSMMHH). This function converts it to a string (HH-MM-SS-FF)

```
dim sTC, sBCD as string
//assign value to sBCD
```

```
//by reading the value from a device
sTC = BCDToTCString(sBCD)
```

Ceil

`Ceil (Double) As Double`

Returns the value specified rounded up to the nearest Integer.

```
Dim d as Double

d=Ceil(1.234) //returns 2
```

ChangeCMD_MinMax

This Command is available only in Device Drivers. It is used to update a drop down list with parameter values.

`ChangeCMD_MinMax("CueSpot","P1",res)`

See Device Editor Automatic Parameters for more information.

CheckSum8Bit

`CheckSum8Bit(string) As Integer`

Returns the Check-sum of a string. Each byte of the string is treated as binary value (not as ASCII code) and is added to the Check-sum. If the Check-sum exceed 256 in “roll over” to zero.

```
Dim c as integer

c = CheckSum8Bit("ABC")   // returns 198
c = CheckSum8Bit(chr(1)+chr(2)+chr(3)) // returns 6
```

CheckSumMod255

`CheckSumMod255(string) As Integer`

Returns the Check-sum of a string. Each byte of the string is treated as binary value (not as ASCII code) and is added to the Check-sum. The result is the Sum Mod 255.

---

checksumXOR

Returns the XOR Checksum od a given string

```
dim sXor as string
```

```
sXour = checksumXOR("My String to Checksum")
```

CRC16

Returns the CRC16 Checksum of a given string

```
dim sCrc as string
sCrc = CRC16("My String to Checksum")
```

Chr

```
Chr (Double) As String
```

Returns the character whose ASCII value is passed.

```
Dim Tab,CR,MyA as String

Tab=Chr(9) //returns a tab
CR=Chr(13) //returns carriage return
MyA = Chr(65) //returns "A"
```

The Chr function will return a single byte String when running on single byte systems and return a double byte string when running on double byte systems. If you need to get a single byte string regardless of whether the system software is single or double byte, use the ChrB function.

ChrB

```
ChrB (Double) As String
```

Returns a single byte String whose value is passed.

```
Dim s as String

s=ChrB(32) //returns a space
s=ChrB(13) //returns carriage return
```

The ChrB function returns a single byte string whose value is specified. ChrB should be used rather than Chr when value represents binary data.

If you need to get a single byte string when running on single byte system software and a double byte string when running on double byte system software, use the Chr function.

If you need to specify a code point above 127, use the Chr property of the TextEncoding class. With it, you specify both the encoding and the character code in that encoding.

Const

```
Const name = value
```

Declares a value as a local constant.

```
Const Pi=3.14159265358979323846264338327950
```

The Const statement can be used in place of the Dim statement followed by an assignment statement when you are sure that the value of the variable should not change within the method. Using Const instead of Dim provides a convenient way to manage such values.

A Const statement can be placed anywhere in a method, including inside a conditional structure, such as an If statement, or a looping structure. Constants declared in this manner are local to the method.

Cos

```
Cos (Double) As Double
```

Returns the cosine of the given angle.

```
Dim d as Double

Const PI=3.14159
d=Cos(45*PI/180) //returns 0.707
```

CountFields

```
CountFields (String, String) As Integer
```

Returns the number of values (fields) in the string passed that are separated by the separator string passed. If the source string is binary data or you require case-sensitivity, use CountFieldsB instead.

The example below returns 5.

```
Dim count as Integer
Dim s as String

s="Dan*Smith*11/22/69*5125554323*Male"
count=CountFields(s, "*")
```

The following example returns three because it counts the null “field” after the (unnecessary) final field delimiter.

```
Dim count as Integer
Dim s as String

s="Dan*Smith*"
count=CountFields(s, "*")
```

CRC16

Returns a CRC16 calculated checksum

Delay

`Delay (String)`

Waits the given number in ms. The given number must be a string.

```
Delay("1000")    //Wait 1 Second
```

```
Dim i as integer

i = 500
Delay(str(i))     //Wait 500 ms
```

Exit

`Exit`

The Exit statement causes control to exit a loop and jump to another line of code without the loop conditions being satisfied. The optional keywords enable you to control where execution will resume.

- If you use Exit without any optional keywords, V-Control exits whatever loop the Exit statement is in.
- If you use Exit with the Do keyword, V-Control exits the Do loop it is in, even if it is also inside another type of loop.
- If you use Exit with the While keyword, V-Control exits the While loop it is in, even if it is also inside another type of loop.
- If you use Exit with the For keyword without passing a parameter, it exits the innermost For loop it is in, even if it is also inside another loop.
- If you have nested For statements, you can pass the For keyword the variable that controls the loop you want to exit. For example, if you are testing all the elements of a two-dimensional array with nested For statements, you can use the loop variable for the outermost loop to exit at that point.

```
For i= 0 to 255
   For j= 0 to 255

     If myArray(i,j) = 23 then
       Exit For i
     End if
   Next
Next
```

Exp

```
Exp( Double) As Double
```

Returns “e” to the power of the value specified.

```
Dim d as Double

d=Exp(10) //returns 22026.4657948
```

FloatStringToStr

Converts a string representing a double i.e. “1.204”) to a four byte float string

This is the code behind the function call

```
Sub FloatStringToStr(fString as string) As String

//converts a string representing a float value ("1.2302")  to a
four byte float string

Dim mb As MemoryBlock = New MemoryBlock(4)
dim s,stmp as string
dim FloatVal as Single

FloatVal = CDbl(fString)
mb.SingleValue(0) = FloatVal
// automatic conversion between memoryblock and strings!!   =)
s = mb
stmp = mid(s,4,1)+mid(s,3,1)+mid(s,2,1)+mid(s,1,1)
return stmp
End Sub
```

Floor

```
Floor( Double) As Double
```

Returns the value specified rounded down to the nearest Integer.

```
Dim d as Double

d=Floor(1.234) //returns 1
```

For...Next

```
For...Next
```

Executes a series of statements a specified number of times. The For Each... Next statement is a variation that executes once for each element of a one-dimensional array.

The counter variable in a For statement can be declared inside the For

statement rather than externally, as a local variable. For example, the code:

```
Dim i as Integer

For i=0 to 10
    //your code goes here
Next
```

The following example uses the DownTo keyword to decrement the counter:

```
Dim i as Integer

For i=5 DownTo 1 Step 1
   Beep
Next
```

Format

`Format (Double, String) As String`

Returns as a String a formatted version of the number passed based on the parameters specified. The Format function is similar to the way spreadsheet applications format numbers. Format will use the information based on the user's locale even if the user's locale is a Unicode-only locale.

Syntax

result = Format(number, formatSpec)

The formatSpec is a string made up of one or more special characters that control how the number will be formatted:

Character	Description
#	Placeholder that displays the digit from the value if it is present.If fewer placeholder characters are used than in the passed number, then the result is rounded.
0	Placeholder that displays the digit from the value if it is present.If no digit is present, 0 (zero) is displayed in its place.
.	Placeholder for the position of the decimal point.
,	Placeholder that indicates that the number should be formatted with thousands separators.
%	Displays the number multiplied by 100.
(	Displays an open paren.
)	Displays a closing paren.

+	Displays the plus sign to the left of the number if the number is positive or a minus sign if the number is negative.
-	Displays a minus sign to the left of the number if the number is negative. There is no effect for positive numbers.
E or e	Displays the number in scientific notation.
\character	Displays the character that follows the backslash.

The absolute value of the number is displayed. You must use the + or - signs if you want the sign displayed.

Although the special formatting characters are U.S. characters, the actual characters that will appear are based on the current operating system settings. For example, Windows uses the settings in the user's Regional and Language Options Control Panel. Formatting characters are specified in similar ways on other operating systems.

The formatSpec can be made up of up to three formats separated by semicolons. The first format is the format to be used for positive numbers. The second format is the format to be used for negative numbers and the third format is the format to be used for zero.

The following are several examples that use the various special formatting characters.

Format	Number	Formatted String
#.##	1.786	1.79
#.0000	1.3	1.3000
0000	5	0005
#%	0.25	25%
###,###.##	145678.5	145,678.5
#.##e	145678.5	1.46e+5
-#.##	-3.7	-3.7
+#.##	3.7	+3.7
#.##;(#.##);\z\e\r\o	3.7	3.7
#.##;(#.##);\z\e\r\o	-3.7	(3.7)
#.##;(#.##);\z\e\r\o		zero

The following example returns the number 3560.3 formatted as $3,560.30.

```
Dim s as String

s=Format(3560.3, "\$###,##0.00")
```

FramecodeToTimecode

returns a string that contains a Timecode in the format “hh-mm-ss-ff”. The function uses the Global Framerate (Main Menu -> Options) to determine the frames per second. The following example converts a framecode to a timecode.

```
dim tc as string
tc = FramecodeToTimecode(123456)
```

GetDayOfMonth

Returns the day of the current month (1..31)
 Result integer

GetDayOfWeek

Result integer (Sunday=0, Monday = 1…)

GetDeviceVar

Gets the string value of a device variable.

In a Device Driver, the syntax is

```
dim s as string
s = GetDeviceVar("VariableName")
```

In a Task, the syntax is

```
dim s as string
s = GetDeviceVar("DeviceName","VariableName")
```

All Parameters are strings

GetGlobalVar

Gets the String Value of a Global variable

```
Dim s as String
s = GetGlobalVar("Test")
```

This Function call is not available in Device Drivers.

GetHour

Result integer

GetMinute

Result integer

GetMonth

Result integer

GetTime

Result string

This function returns a Timecode Time string. The format is HH:MM:SS:FF (FF = Frames). To determine the Timecode format, the function need a parameter for the framerate (fps)

```
Dim res as string
dim framerate as integer

framerate = 25
res = GetTime(framerate)
```

GetSeconds

Result integer

GetYear

Result integer

Hex

```
Hex (Integer) As String
```

Returns as a String the hexadecimal version of the number passed.

If the value is not a whole number, the decimal value will be truncated.

You can specify binary, hex, or octal numbers by preceding the number with the & symbol and the letter that indicates the number base. The letter b indicates binary, h indicates hex, and o indicates octal.

```
Dim hexVersion As String

hexVersion=Hex(5) //returns "5"
hexVersion=Hex(75) //returns "4B"
```

```
hexVersion=Hex(256) //returns "100"
```

HexToStr

```
HexToStr (String) As String
```

Returns as a String hexadecimal numbers converted to string

```
dim s as string

s = HexToStr("414243")
   // Returns "ABC", because "A" has ASCII code &h41 or 65
decimal,
   //         "B" has ASCII code &h42 or 66 decimal etc.
```

If...Then...Else

```
If...Then...Else
```

Conditionally executes a group of statements, depending on the value of a Boolean expression.

When executing an If statement, the condition is tested. If condition is True, the statements associated with the If statement following the Then statement are executed. If condition is False and an Else clause follows, its statements will be executed. If condition is False and there is no Else clause or it is preceded by an ElseIf statement, the condition following the ElseIf statement is tested. After executing the statements following Then, ElseIf or Else execution continues with the statement that follows End If.

```
If error=-123 Then
 Beep
 MsgBox "Whoops! An error occured."
End If
```

```
Dim theNumber As Integer
Dim digits As Integer

theNumber=33
If theNumber<10 Then
   digits=1
ElseIf theNumber<100 Then
   digits=2
ElseIf theNumber<1000 Then
   digits=3
Else
   digits=4
End If
```

InStr

```
InStr (Integer, String, String) As Integer
```

Returns the position of the first occurrence of a String inside another String. The first character is numbered 1.

```
Dim first As Integer

first = InStr("This is a test", "t")     //returns 1
first = InStr("This is a test", "is")    //returns 3
first = InStr(4, "This is a test", "is") //returns 6
first = InStr("This is a test", "tester")//returns 0
```

InStrB

```
InStrB (Integer, String, String) As Integer
```

If the find string is not found within the source string, 0 (zero) is returned. InStrB is case-sensitive; it treats source as a series of raw bytes. It should be used instead of InStr when the string represents binary data or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

```
Dim first As Integer

first = InStrB("This is a test", "T")     //returns 1
first = InStrB("This is a test", "t")     //returns 11
first = InStrB("This is a test", "is")    //returns 3
first = InStrB(4, "This is a test", "is") //returns 6
first = InStrB("This is a test", "tester")//returns 0
first = InStrB("This Is a test", "Is")    //returns 6
```

Left

```
Left (String, Integer) As String
```

Returns the first n characters in a source String.

```
Dim s As String

s=Left("Hello World", 5) //returns "Hello"
s="Hello World"
s=s.Left(5) //returns "Hello"
```

LeftB

```
LeftB (String, Integer) As String
```

Returns the first n bytes in a source String.

The LeftB function returns bytes from the source string starting from the left side (as the name implies). If you need to read characters rather than bytes, use the Left function.

This example uses the LeftB function to return the first 5 bytes from a string.

```
Dim s As String
s=LeftB("Hello World", 5) //returns "Hello"

s="Hello World"
s=s.LeftB(5) //returns "Hello"
```

Len

```
Len (String) As Integer
```

Returns the number of characters in the specified String.

```
Dim n As Integer
n=Len("Hello world") //returns 11

Dim s as String
s="Hello World"
n=s.Len //returns 11
```

LenB

```
LenB (String) As Integer
```

Returns the number of characters in the specified String.

LenB treats string as a series of bytes, rather than a series of characters. It should be used when string represents binary data. If you need to know the number of characters in string rather than the number of bytes, use the Len function.

Log

```
Log (Double) As Double
```

Returns the natural logarithm of the value specified.

```
Dim d As Double
d=Log(10) //returns 2.3025851
```

Lowercase

```
Lowercase (String) As String
```

Converts all characters in a String to lowercase characters.

```
Dim s As String
s=Lowercase("tHe Quick fOX") //returns "the quick fox"
s=Lowercase("THE 5 LAZY DOGS") //returns "the 5 lazy dogs"

s="tHe Quick fOX"
s=s.Lowercase /returns "the quick fox"
```

LTrim

```
LTrim (String) As String
```

Returns the String passed with leading (left side) whitespaces removed.

```
Dim s as String
s=LTrim(" Hello World ")
//Returns "Hello World "
s=" Hello World "
s=s.LTrim //Returns "Hello World "
```

Max

```
Max (Double, Double) As Double
```

Returns the largest value passed to it.

```
Dim d As Double

d=Max(3.01, 4.05) //returns 4.05
d=Max(3.012, 3.011, 1.56) //returns 3.012
```

Microseconds

```
Microseconds As Double
```

Returns the number of microseconds (1,000,000th of a second) that have passed since the user's computer was started.

```
Dim minutes As Integer

minutes=Microseconds/1000000/60
MsgBox "Your computer has been on for "+ Str(minutes)+" minutes."
```

Because modern operating systems can stay running for so long, it's possible for the machine's internal counters to “roll over.” This means that if you are using this function to determine how much time has elapsed, you may

encounter a case where this time is inaccurate.

Mid

```
Mid (String, Integer, Integer) As String
```

Returns a portion of a String. The first character is numbered 1.

```
Dim s As String

s = Mid("This is a test", 6) //returns "is a test"
s = Mid("This is a test", 11, 4) //returns "test"

s="This is a test"
s=s.Mid(11,4) //returns "test"
```

MidB

```
MidB (String, Integer, Integer) As String
```

Returns a portion of a String. The first character is numbered 1.

```
Dim s As String

s=MidB("This is a test", 6) //returns "is a test"
s=MidB("This is a test", 11, 4) //returns "test"

s="This is a test"
s=s.MidB(11,4) //returns "test"
```

MidB treats source as a series of bytes, rather than a series of characters. MidB should be used when source represents binary data. If you need to extract characters rather than bytes, use the Mid function. To determine the number of bytes in a String, use the LenB function.

Min

```
Min (Double, Double) As Double
```

Returns the smallest of the numbers passed.

```
Dim d As Double

d=Min(3.01, 4.05) //returns 3.01
d=Min(3.012, 3.011) //returns 3.011
```

---

msToTimecode

Convert Milliseconds to Timecode. Returns a string that contains a Timecode in the format “hh-mm-ss-ff”. The function uses the Global Framerate (Main Menu -> Options) to determine the frames per second. The following example converts a framecode to a timecode.

```
dim tc as string
tc = msToTimecode(1000) // tc = 00-00-01-00
```

Nil

```
Nil
```

Used to determine if an object is nil (no value).

NthField

```
NthField (String, String, Integer) As String
```

Returns a field from a row of data. The first field is numbered 1. If you need to parse binary data, use NthFieldB instead.

This example returns “Smith”

```
Dim field As String
field=NthField("Dan*Smith*11/22/69*5125554323*Male","*",2)
//Field = smith
```

In the example above, the * is a field delimiter. The string “Dan*Smith*11/22/69*5125554323*Male” has 5 fields if * is the delimiter.

1. “Dan”
2. “Smith”
3. “11/22/69”
4. “5125554323”
5. “Male”

If / is choosen as delimiter, then the string has 3 fields:

1. “Dan*Smith*11”
2. “22”
3. “69*5125554323*Male”

```
Result = NTHField(StringWithFields, Delimmiter, Position)
```

NthFieldB

Returns a field from a row of data. NthFieldB is identical to NthField except that it treats the source data as binary data. The first field is numbered 1.

This example returns “Smith”

```
Dim field As String
field = NthFieldB("Dan*Smith*11/22/69*5125554323*Male", "*", 2)
```

Using the second syntax:

```
Dim s, field As String
s = "Dan*Smith*11/22/69*5125554323*Male"
field = s.NthFieldB("*", 2)
```

Oct

```
Oct (Integer) As String
```

Returns as a String, the octal version of the number passed.

```
Dim OctVersion As String

OctVersion=Oct(5) //returns "5"
OctVersion=Oct(75) //returns "113"
OctVersion=Oct(256) //returns "400"
```

Pow

```
Pow (Double, Double) As Double
```

Returns the value specified raised to the power specified.

This example uses the Pow function to return four raised to the power of seven.

```
Dim d As Double
d=Pow(4,7) //returns 16384
```

Redim

```
Redim Array(x,y)
```

Resizes the passed array.

This example reduces the aNames array to 11 elements.

```
Redim aNames(10)
```

This example adds 10 elements to the aNames array

```
Redim aNames( Ubound(aNames)+10)
```

This example reduces the aPeople array to 11 elements for the first dimension

and 6 elements for the second dimension

```
Redim aPeople(10,5)
```

The Redim method is used to increase or reduce the number of elements in the array specified. Arrays are zero-based (the first element is zero) so you resize the array using a number that is one less than the number of elements you actually want. The number of parameters passed is the number of dimensions of the array being resized.

Rem

```
Rem any comment
```

Used to add comments to your code.

Replace

```
Replace (String, String, String) As String
```

Replaces the first occurrence of a String with another String.

```
Dim result As String

result=Replace("The quick fox","fox","rabbit") //returns "The
quick rabbit"
result=Replace("The quick fox","f","b") //returns "The quick box"
result=Replace("The quick fox","quick","") //returns "The fox"
```

ReplaceB

```
ReplaceB (String, String, String) As String
```

Replaces the first occurrence of oldString in sourceString with newString. ReplaceB is the byte version of Replace.

- If newString is an empty string (“”), the ReplaceB function deletes the first occurrence of the oldString in the sourceString.
- If oldString is an empty string (“”), the ReplaceB function returns an unchanged copy of the sourceString.

ReplaceB is case-sensitive; it treats sourceString as a series of raw bytes. It should be used instead of Replace when the string represents a series of bytes or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

ReplaceAll

```
ReplaceAll (String, String, String) As String
```

Replaces all occurrences of a String with another String.

```
Dim result As String

result=ReplaceAll("xyxyxy","x","z") //returns "zyzyzy"
result=ReplaceAll("The quick fox"," ","") //returns "Thequickfox"

result="The Quick Fox"
result=result.ReplaceAll(" ",",") //returns "The,Quick,Fox"
```

ReplaceAllB

```
ReplaceAllB (String, String, String) As String
```

The ReplaceAllB function replaces all occurrences of oldString in sourceString with newString. ReplaceAllB is case-sensitive because it treats the source string as a series of raw bytes.

- If newString is an empty string (“”), the ReplaceAllB function deletes every occurrence of the oldString in the sourceString.
- If oldString is an empty string (“”), the ReplaceAllB function returns an unchanged copy of the sourceString.

ReplaceAllB is case-sensitive; it treats sourceString as a series of raw bytes. It should be used instead of ReplaceAll when the string represents a series of bytes or when your application will run in a one-byte character set (such as the US system) and you want case-sensitivity.

Right

```
Right (String, Integer) As String
```

Returns the last n characters from the String specified.

```
Dim s As String
s=Right("Hello World", 5) //returns "World"

s="Hello World"
s=s.Right(5) //returns "World"
```

RightB

```
RightB (String, Integer) As String
```

The RightB function returns bytes from the source string starting from the right

side (as the name implies). RightB treats source as a series of bytes rather than a series of characters. It should be used when source represents binary data. If you need to read characters rather than bytes, use the Right function.

Rnd

```
Rnd As Double
```

Returns a randomly generated number in the range 0 <= Rnd < 1. The equivalent functionality is provided by the Random class as a special case. The Random class also provides additional options, such as a random number selected from a Normal distribution.

Round

```
Round (Double) As Double
```

Returns the value specified rounded to the nearest Integer.

```
Dim d as Double

d=Round(1.499) //returns 1
d=Round(1.500) //returns 2
```

RTrim

```
RTrim (String) As String
```

Returns the String data type passed with trailing (right side) whitespaces removed.

```
Dim s as String
s=RTrim(" Hello World ") //Returns " Hello World"

s=" Hello World "
s=s.RTrim //Returns " Hello World"
```

Rtrim uses the list of unicode “whitespace” characters at http://www.unicode.org/Public/UNIDATA/PropList.txt.

Select Case

```
Select Case
```

Executes one of several groups of statements, depending on the value of an expression.

The Select Case statement is useful when there are several possible conditions

that must be checked. Unlike an If statement, the Select Case statement will exit as soon as it finds a matching Case expression and executes any statements that follow the Case expression up to the next Case expression. If there are no Case expressions that match, the elseStatements are executed. The expression Case Else can be used as a synonym for Else. The Case statement can accept several types of expressions. The expression can be a single value, a comma-delimited list of values, a function that returns a value, a range of values specified with the 'To“ keyword, or an expression that uses the “Is” keyword to do an equality or inequality test. You can combine types of expressions, separating them by commas Here are some examples:

```
Case 2, 4, 6, 8 //several values
Case 2 To 5 //range of values using To
Case 2 To 5, 7,9,11 //Both separate values and range
Case myFunction(x) // a Function
Case Is >= 42 // greater than/equal to operator
Case Is <19 //less than operator
```

```
Dim d as New MessageDialog //declare the MessageDialog object
Dim b as MessageDialogButton //for handling the result

d.icon= MessageDialog.GraphicCaution //display warning icon
d.ActionButton.Caption="Save"
d.CancelButton.Visible= True //show the Cancel button
d.CancelButton.Cancel= True//esc key works for Cancel
d.AlternateActionButton.Visible= True //show the "Don't Save"
button
d.Message="Save changes before closing?"
d.Explanation="If you don't save your changes, you will lose " _
+"all that important work you did since your last coffee break."
b=d.ShowModal //display the dialog
Select Case b //b is a MessageDialogButton
Case d.ActionButton //determine which button was pressed.
//user pressed Save
Case d.AlternateActionButton
//user pressed Don't Save
Case d.CancelButton
//user pressed Cancel
End Select
```

SetDeviceVar

Sets a Device Variable to a new Value.

In a Device Driver, the syntax is

```
SetDeviceVar(VarName,NewValue)
```

In a Script, the syntax is

```
SetDeviceVar(DeviceName,VarName,NewValue)
```

All Parameters are Strings

SetGlobalVar

```
// All Parameters are Strings
SetGlobalVar(VarName,NewValue)
```

This Function call is not available in Device Drivers and Event Condition scripts, only in Tasks

---

Sin

```
Sin (Double) As Double
```

Returns the sine of the value specified.

```
Dim d As Double
Const PI=3.14159265358979323846264338327950
d=Sin(0.5) //returns 0.4794255
d=Sin(30*PI/180) //returns .5
```

SingleToStr

Converts a Single to a four byte single string

This is the code behind the function call

```
Sub SingleToStr(FloatVal as single) as String

//converts a single (i.e. 0.023) to a four byte float string

Dim mb As MemoryBlock = New MemoryBlock(4)
dim s,stmp as string

mb.SingleValue(0) = FloatVal
// automatic conversion between memoryblock and strings!!   =)
s = mb
stmp = mid(s,4,1)+mid(s,3,1)+mid(s,2,1)+mid(s,1,1)
return stmp

End Sub
```

Sqrt

```
Sqrt (Double) As Double
```

Returns the square root of the value specified.

```
Dim d As Double
d=Sqrt(16) //returns 4
```

Str

```
Str (Double) As String
```

Returns the String form of the value passed.

```
Dim s As String

s=Str(123) //returns "123"
s=Str(-123.44) //returns "-123.44"
s=Str(123.0045) //returns "123.0045"
Const Pi=3.14159265358979323846264338327950
s= Str(pi) // returns "3.141593"
s= Str(3141592653589012345) // returns "3141592653589012345"
```

StrComp

```
StrComp (String, String, Integer) As Integer
```

Makes a binary (case-sensitive) or text (lexicographic) comparison of the two strings passed and returns the result.

The following example returns -1 because the two strings are the same in every way except in case.

```
StrComp(“Spam”, “spam”, 1)
```

The following example returns -1 because in a text comparison of the two strings, string2 is greater than string1. The ASCII value of “s” is greater than the ASCII value of “S”.

```
StrComp(“Spam”, “spam”, 0)
```

StrToHex

```
StrToHex (String) As String
```

Returns a String of hexadecimal numbers

```
dim s as string

s = HexToStr("ABC")
// Returns "414243", because "A" has ASCII code &h41 or 65
decimal,
//         "B" has ASCII code &h42 or 66 decimal etc.
```

Tan

```
Tan (Double) As Double
```

Returns the tangent of the angle specified.

```
Dim d As Double
Const PI=3.14159265358979323846264338327950
d=Tan(45*PI/180) //returns 1.0
```

Ticks

```
Ticks as Integer
```

Returns the number of ticks (60th of a second) that have passed since the user's computer was started.

```
Dim minutes As Integer
minutes=Ticks/60/60
MsgBox "Your computer has been on for"+ Str(minutes)+" minutes."
```

Because modern operating systems can stay running for so long, it's possible for the machine's internal counters to “roll over.” This means that if you are using this function to determine how much time has elapsed between two events, you may encounter a case where it appears that the stop time is prior to the start time.

TimecodeToFramecode

returns a integer that contains the number of frames in a given timecode. The function uses the Global Framerate (Main Menu -> Options) to determine the frames per second. The following example converts a timecodeto a framecode.

```
dim fc as integer
fc = TimecodeToFramecode("10-05-12-21")
```

Titlecase

```
Titlecase (String) As String
```

Returns the String passed to it with all alphabetic characters in Titlecase.

```
Dim s As String
s=Titlecase("tHe Quick fOX") //returns "The Quick Fox"
s=Titlecase("THE LAZY DOG") //returns "The Lazy Dog"
```

Trim

```
Trim (String) As String
```

Returns the String passed with leading and trailing whitespaces removed.

```
Dim s as String
s=Trim(" Hello World ") //Returns "Hello World"
```

Ubound

```
Ubound (array) As Integer
```

Returns the index of the last element in an array.

The Ubound function can be used to determine the last element of an array, but it can also be used to determine the size of an array. It may appear at first that the last element number and the size of the array are the same but in fact they are not. All arrays have a zero element. In some cases element zero is used and in other cases it is not. You will need to keep this in mind when using the Ubound function to determine the number of values you have in the array. If the array is zero-based, then element zero is used to store a value and you will have to add one to the value returned by the Ubound function to make up for it.

This example replaces each occurrence of X in an array with Y.

```
Dim i As Integer
For i=0 to Ubound(Names)
 If Names(i)="X" Then
    Names(i)="Y"
 End If
Next
```

Uppercase

```
Uppercase (String) As String
```

Converts all characters in a String to uppercase characters.

```
Dim s As String
s=Uppercase("tHe Quick fOX") //returns "THE QUICK FOX"
s=Uppercase("the 5 lazy dogs") //returns "THE 5 LAZY DOGS"
```

Val

```
Val (String) As Double
```

Returns the numeric form of a String.

```
Dim n As Integer
n = Val("12345") //returns 12345
n = Val(" 12345") //returns 12345
n = Val("123 45") //returns 123
```

```
n = Val(" &hFFF") //returns 4095
n = Val(" &b1111") //returns 15
```

The Val function stops reading the String at the first character it doesn't recognize as part of a number. All other characters are automatically stripped. It does recognize prefixes &o (octal), &b (binary), and &h (hexadecimal). However, spaces are not allowed in front of the ampersand. That is, ” &hFF“ returns 0, but ” &hFF“ returns 255. The CDbl function is the same as the Val function but is used when you need to pass a String that uses a character other than the period (.) as the decimal separator. It uses the decimal character specified by the operating system. For example, on Windows XP, it is set in the Regional and Language Options Control Panel. Val should generally be used to convert internal data, but not data entered by the user. Val is not international-savvy, but CDbl is. The CStr function is the same as the Str function but is used when you need to pass a number that uses a character other than the period (.) as the decimal separator. It uses the decimal character specified by the operating system. Val returns zero if string contains no numbers.

Compiler Errors

Error Codes Compiler error numbers returned in errorNumber are shown below:

Error Number	Description
1	Syntax does not make sense.
2	Type mismatch.
3	Select Case does not support that type of expression.
4	The compiler is not implemented (obsolete).
5	The parser's internal stack has overflowed.
6	Too many parameters for this function.
7	Not enough parameters for this function call.
8	Wrong number of parameters for this function call.
9	Parameters are incompatible with this function.
10	Assignment of an incompatible data type.
11	Undefined identifier.
12	Undefined operator.
13	Logic operations require Boolean operands.
14	Array bounds must be integers.
15	Can't call a non-function.
16	Can't get an element from something

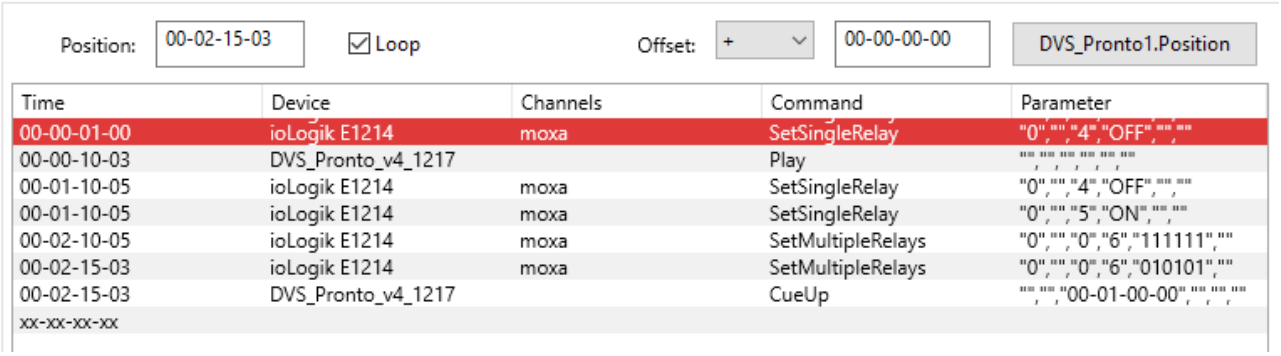
	that isn't an array.
17	Not enough subscripts for this array's dimensions.
18	Too many subscripts for this array's dimensions.
19	Can't assign an entire array.
20	Can't use an entire array in an expression.
21	Can't pass an expression as a ByRef parameter.
22	Duplicate identifier.
23	The backend code generator failed.
24	Ambiguous call to overloaded method.
25	Multiple inheritance is not allowed.
26	Cannot create an instance of an interface.
27	Cannot implement a class as though it were an interface.
28	Cannot inherit from something that is not a class.
29	This class does not fully implement the specified interface.
30	Event handlers cannot live outside of a class.
31	It is not legal to ignore the result of a function call.
32	Can't use “Self” keyword outside of a class.
33	Can't use “Me” keyword outside of a class.
34	Can't return a value from a Sub.
35	An exception object required here.
36-39	Obsolete.
40	Destructors can't have parameters.
41	Can't use “Super” keyword outside of a class.
42	Can't use “Super” keyword in a class that has no parent.

Timelines

Timelines are vertical arranged in V-Control. It is possible to run more than one Timeline at a time.

Timelines are controlled by timecode. To tell the system were to get the timecode from, a Timecode Variable has to be configured. This can be the Position variable of a device such as a timecode reader or a player, as well as the internal clock of the V-Control machine.

To add a cue to a Timeline, enter a value in the *Position* field, select a command and click *Add Cmd*. Now a new row is added to the timeline table. The table is automatically sorted by time code.



Timeline user interface

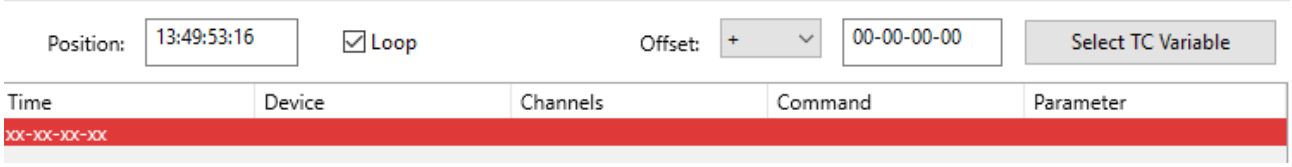
Position: If a timeline is running, this field is updated with the current time(code) for this timeline. If a Timeline is edited, this field determines the position of the new cue. Each Time a cue is added to the timeline, the value of this field ist used as trigger value.

Loop: By default *Loop* is checked. If a Timeline is running, and the current Timecode is > then the last Cue's Timecode, the Timeline will not stop. It keeps running, and as soon as the selected Timecode variables content comes into a range of the Timecode values of the Timeline, it jumps to the corect position.

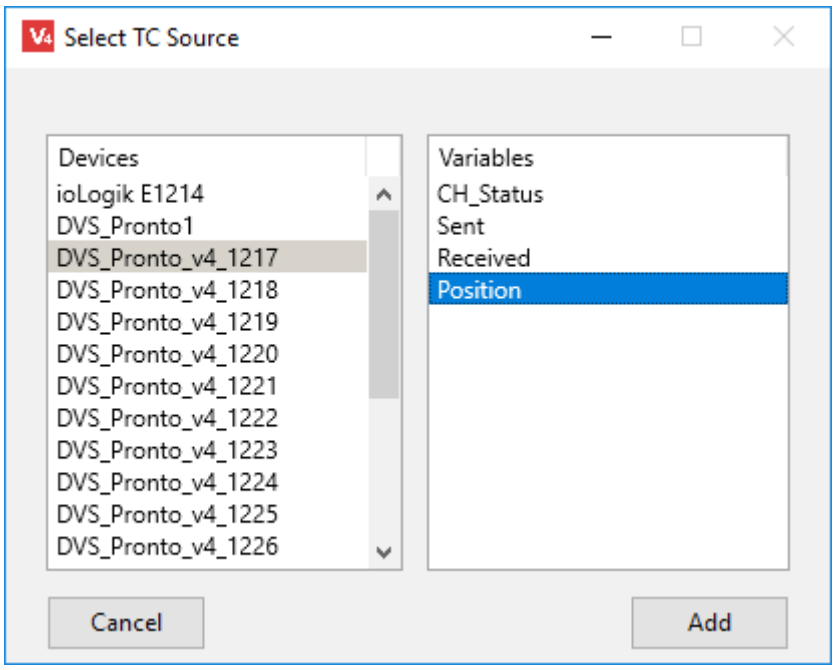
If *Loop* is unchecked, the task will stop if the last cue was executed.

Offset: Determines an offset value to the current Timecode. Could be positive or negative.

Select TC Variable:



In a new Timeline, there is a button labeled *Select TC Variable*. Click this button to select a device variable that contains a Time or Timecode that triggers this timeline.

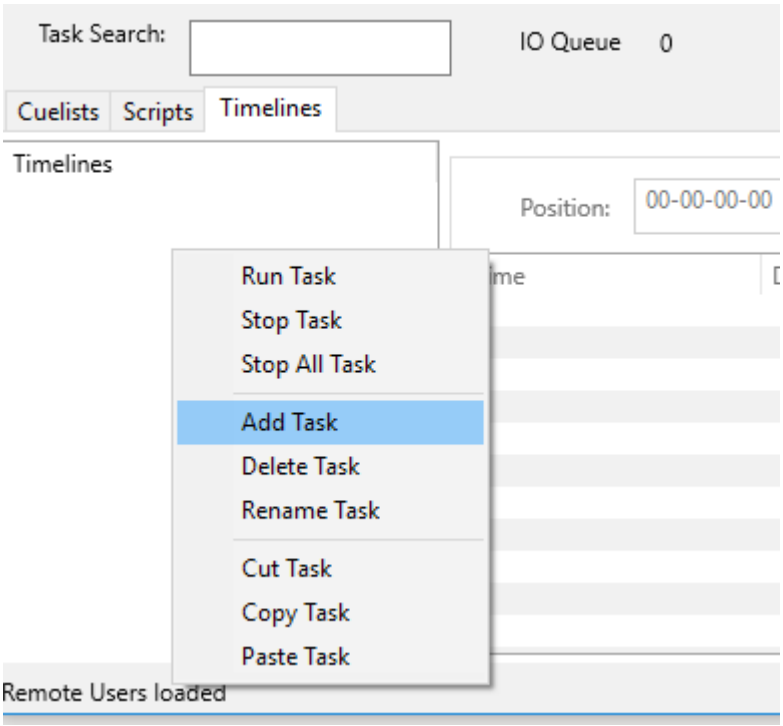


Click the *Add* button if a variable is selected. The caption of the button changes to “Devicename.Variablename”.

Create Timelines

A new Timeline can be created by the Timelinelist's context menu.

Alternatively use the  Icon to do this.



Once created, the work with timelines is very similar then with cuelists. The

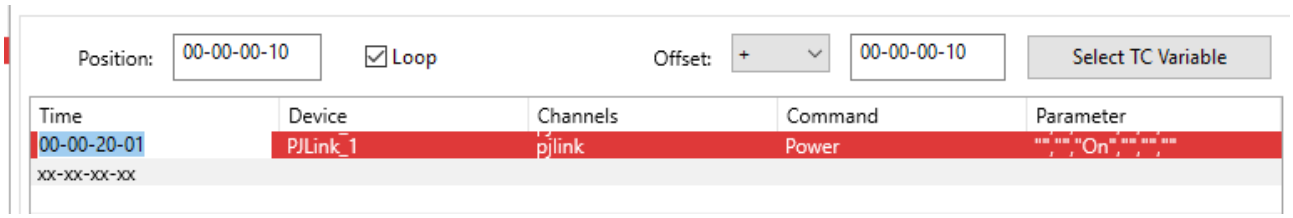
only difference is that the first column has a timecode value instead of an ID. This timecode is used to trigger the cue.

Enter Timecode Values

Once a timecode variable is selected as timecode master, enter a time value in the *Position* field. This field is optimized for timecode values and takes care that the timecode is allways in a correct format. There is no need to enter the exact format by yourself. For example, if you want to have a timecode trigger value “00-01-12-23” it is OK to enter “11223”. After pressing the <Enter> key on your keyboard, the value is automatically formated to “00-01-12-23”.

The algorithm always looks from right to left, filling the missing digits with “0”. The *Offset* field works in the same way.

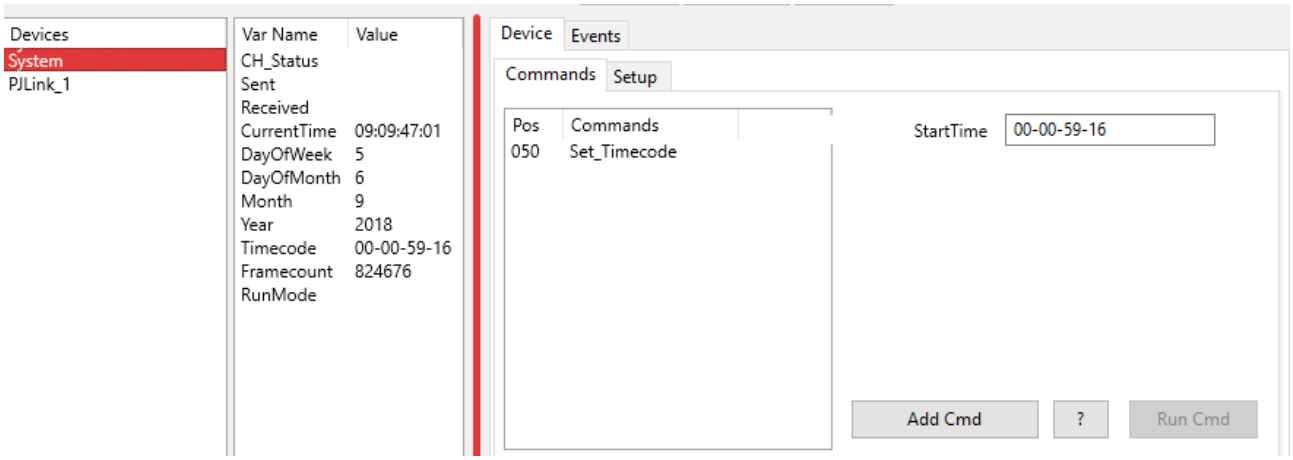
If you want to edit an existing value in the timeline after the cue was created, double click on the timecode value to edit it.



After editing, it might happen that the cue jumps to another row in the timeline. This is because the timeline is always sorted by timecode.

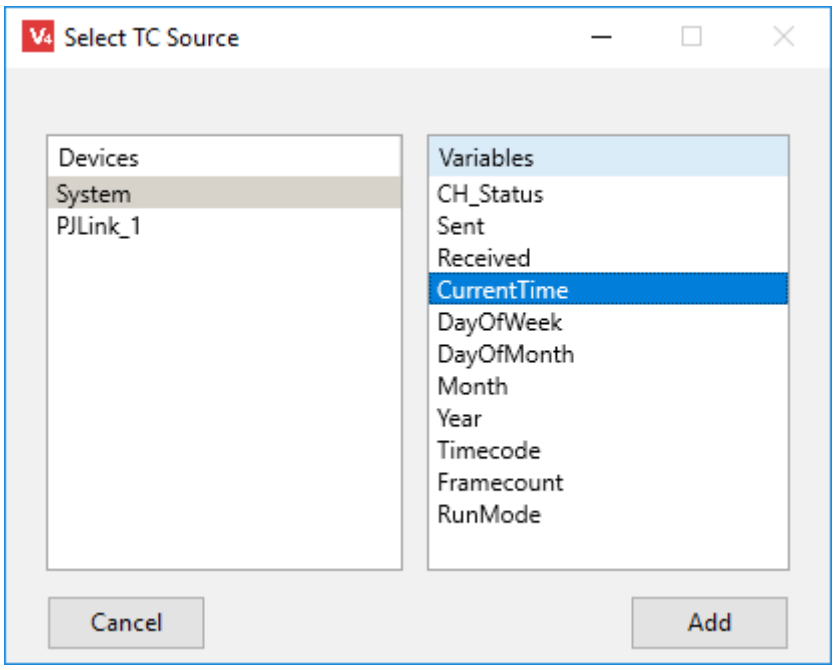
Using Timelines for daily tasks

If there are daily tasks such as power on and off a system, the timeline is one solution (among others) to do this. A *System* device is needed to get the real time of the V-Control system.

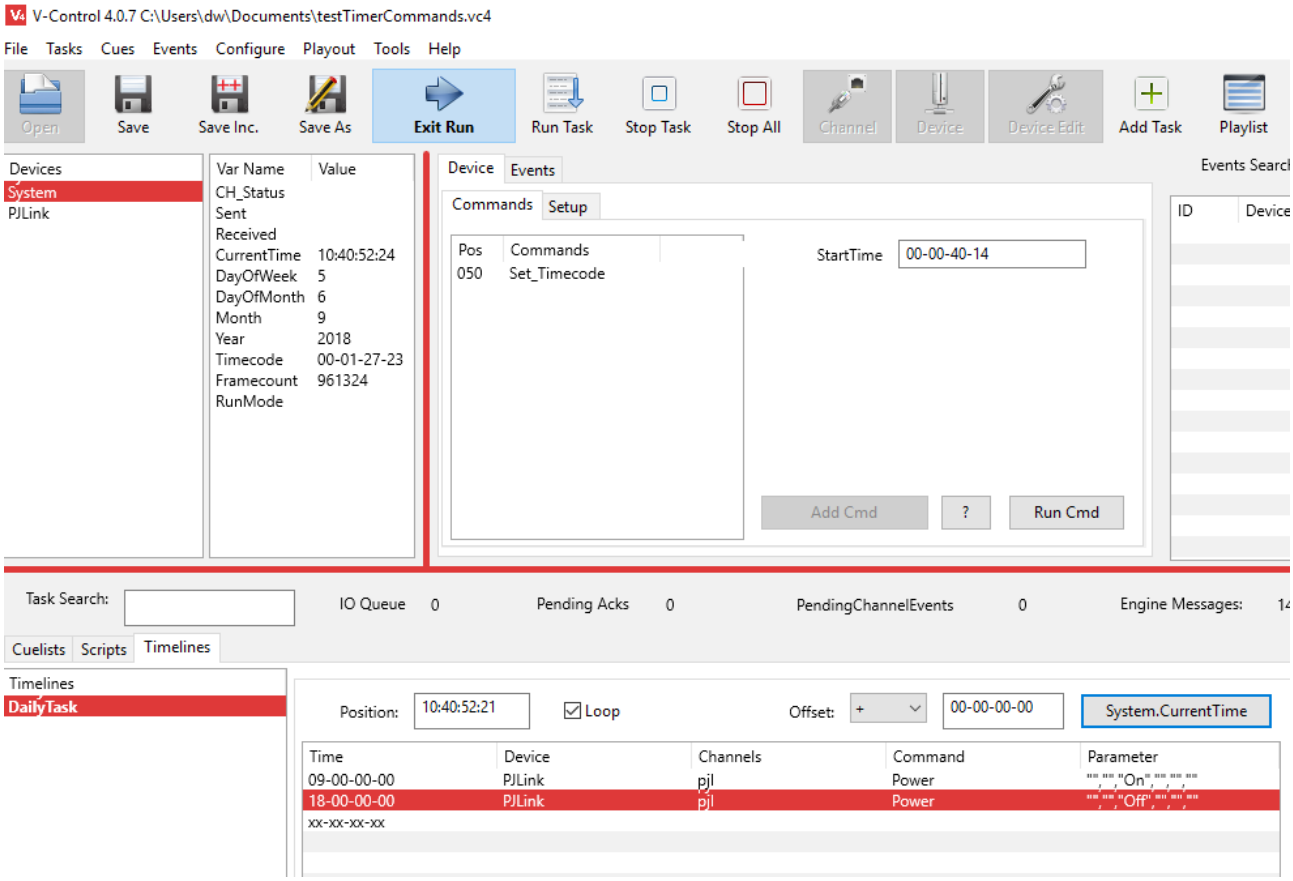


The *System Device* has the device variable *CurrentTime*, which can be used as timecode master for this task.

Click the *Select TC Variable* button to select the *CurrentTime* variable as master timecode.



In the example below, a projector is switched on at 09:00:00 in the morning and off at 18:00:00 in the evening.



Events

An event in V-Control works as a trigger. If the event occurs, a defined task is executed as event handler. All events are triggered by a variable change. This could be a device variable, a system variable or a global variable.

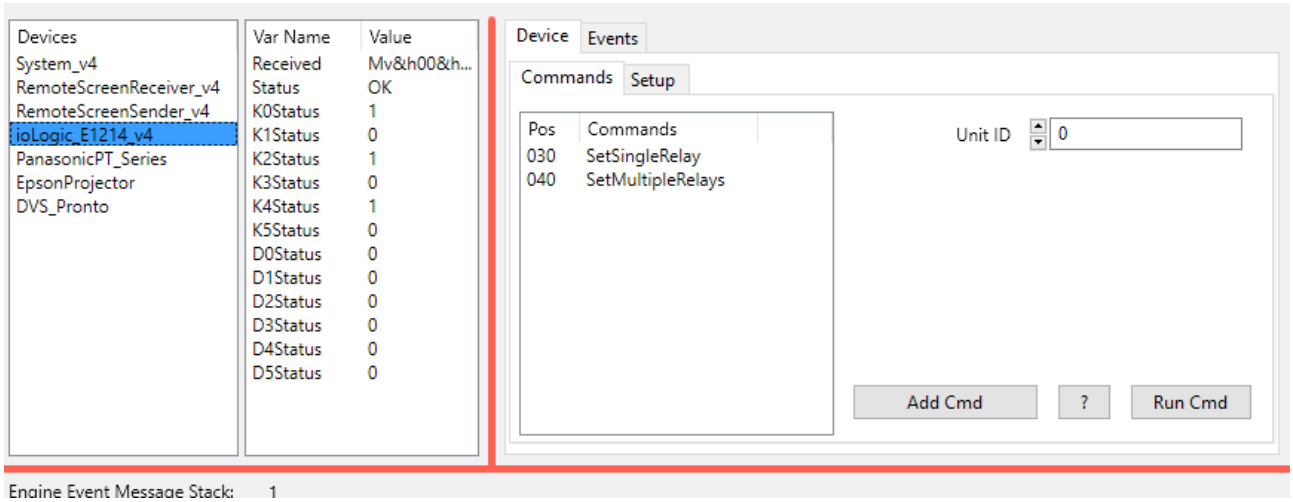
The device drivers collect information about the connected device and represent the status of the device in device variables. Each time a variable changes an event is fired. If there is an event handler for this event then the task that handles the event is fired. Device Events allow V-Control to respond on external events triggered by a connected device. If, for example, a temperature sensor send a message that a limit is reached, or a motion sensor detects a person and notify this, or a players position has changed, V-Control can launch a task and react on this event.

To trigger an event, a condition script determines if the event handling task is fired or not. Lets say we have a temperature sensor and we want to start a fan if the temperature is over 35°C. The fan should stop if the temperature is less then 30°C. So we have two conditions to check and determine which task to start.

Condition 1 is: if Temperature > 35 then start fan Condition 2 is: if Temperature < 30 then stop fan

Device Variable Change Events

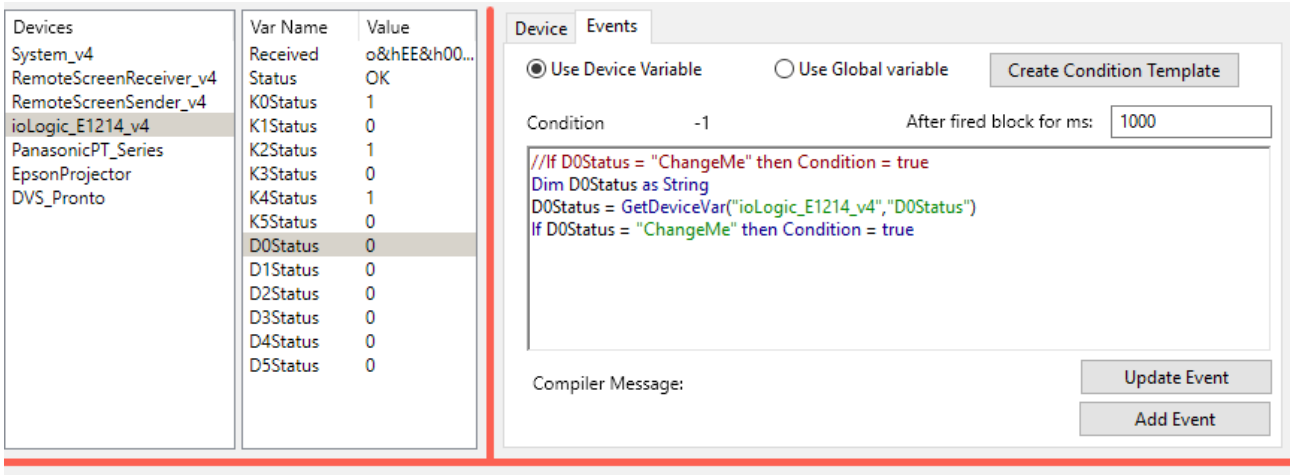
These events initiated by the external attached devices, e.g. a liquid level sensor reports a new level or a light barrier detects a break. The sensor send a message via the channel (or the device driver requests the status periodically), and V-Control looks if there is an event handler. If an event handler is found, it checks the condition and fire the task if the condition is true.



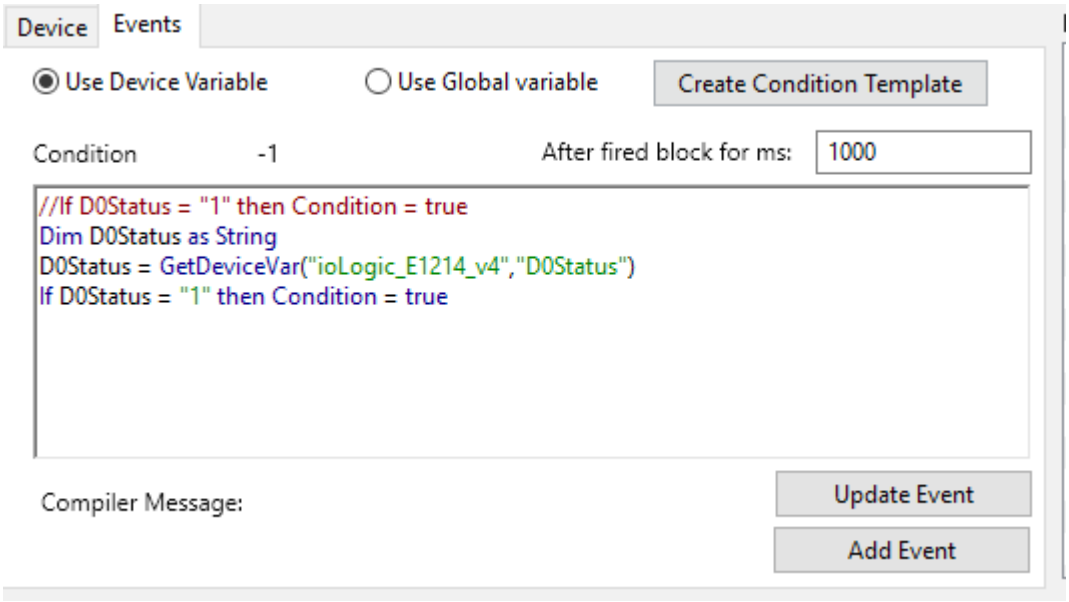
Lets do an example:

The ioLogic E1214 is a TCP controlled box from Moxa. It uses the Modbus TCP protocol. It has 6 Relays and 6 digital inputs.Now we want to run a task if digital input 1 gets High. Select the device and the device variable D0Status. Then click the *Create Condition Template* button. V-Control creates a condition script

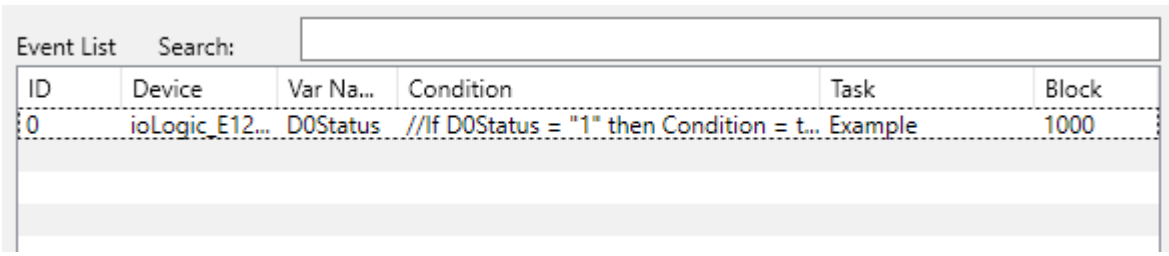
that has to be edited so that the condition is true if D0Status = 1.



The first line of the condition script is a comment. It is shown in the *Eventlist* and is used as a hint for the programmer. Change the condition script as shown below.



Click the *Add Event* button. Now there is a new entry in the Eventlist. The next step is to assign a task to this event. In the context menu (click right) of the event list select Assign Task. The list now looks like this:



Now, if the variable D0Status of the device changes from 0 to 1, the task assigned to the event is fired.

The Condition Script

The first row of the condition script is a comment and shown in the Eventlist. Write whatever is helpful to have a good hint what the script is doing.

The second row

```
Dim D0Status as String
```

creates the local variable D0Status. This is necessary because we don't have direct access to a device variable.

in

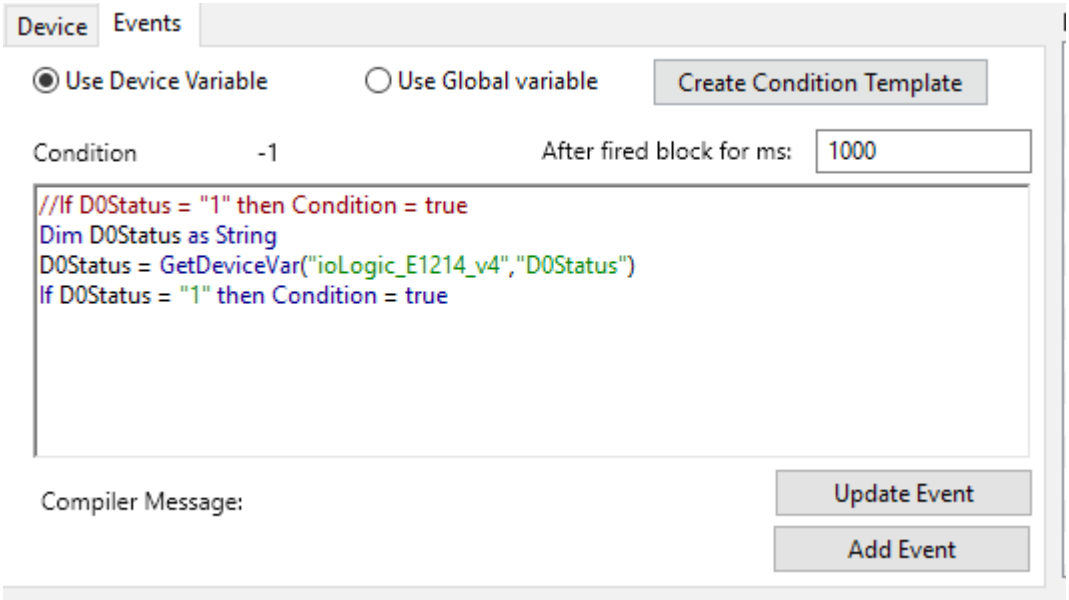
```
D0Status = GetDeviceVar("ioLogic_E1214_v4","D0Status")
```

the content of the device variable D0Status is assigned to the local variable D0Status. The GetDeviceVar function accept only strings as parameters, so the device name and the variable name have to be surrounded by “.

The last row checks if the condition is true. The condition is always false by default and changes only to true if explicit set.

```
If D0Status = "1" then Condition = true
```

The Condition Editor



The Condition Editor tries to support the programmer in creating the script. As explained above, the *Create Condition Template* button creates a script that is used as first step.

Variable change events can occur from device variables and global variables. If then *Use Device Variable* option is set, a device variable is used. For global variables use the *Use Global Variable* option.

Device variables can change very frequently. A variable containing a timecode changes every 33 ms (30 fps). If you have a condition such as

```
Dim Positionas string
```

```
Dim iPosition,iTarget as Integer

Position = GetDeviceVar("DVS_Pronto","Position"

iTarget = TimecodeToFramecode("10-12-21-10")
iPosition = TimecodeToFramecode(Position)
if (iPosition >= iTarget) and (iPosition < Target +10) then
Condition = true
```

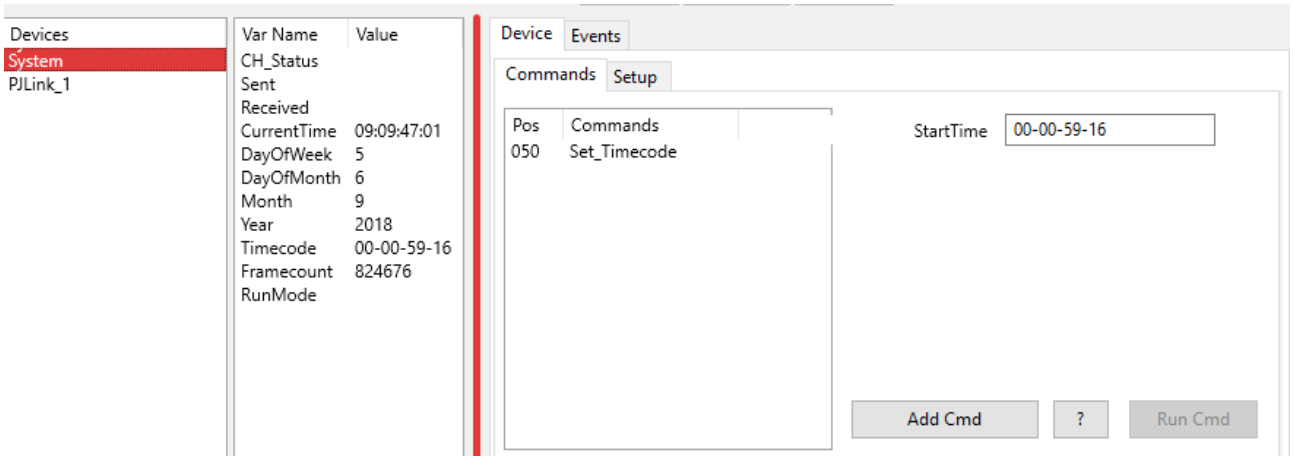
This script returns true if the Position is between 10-12-21-10 and 10-12-21-20. It might be that V-Control does not detect all of the 10 valid variable changes, but for sure more then one. In this case the Task that is linked to the event is fired frequently during this period. It could happen that V-Control tries to fire the task again while it is still running. For that reason we have the *After fired block for ms*. Here you can enter the time in ms the event should not fire again. After the time it will fire again. This value is 1000ms (1 Second) by default.

If your script causes compile errors the line with the error is marked in the script and an error message is shown (*Compiler Message*).

A double click in the eventlist loads the event again. To change the event click the *Update Event* button, to create a new click the *Add Event* button.

Calendar Events

To create events that depend on a Date, Time or Time interval the System device is used. To add a System device open the *Device Setup* dialog by clicking the *Device* button in the toolbar. Search for System_v4.

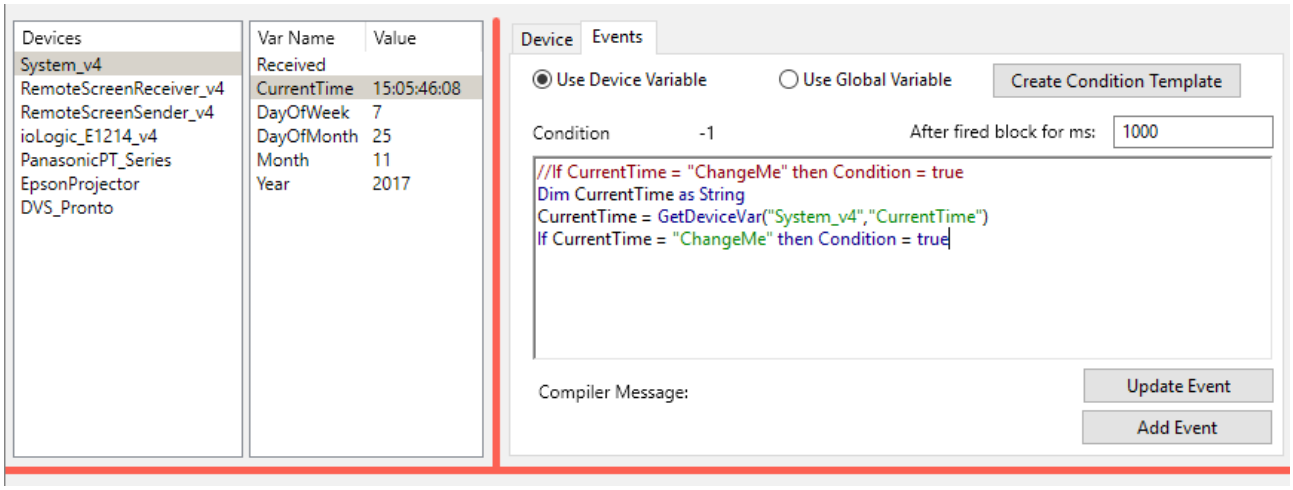


The System Device is the only device that does not need a channel. It has no commands, only status variables representing Date and Time. The Time is in the format HH:MM:SS:FF (FF = Frames). The framerate is determined in the Options Dialog System page.

DayOfWeek is the current weekday (1=Sunday..7=Saturday)

To create an event that fires weekdays at 9:00 AM we have to check the current Time and the DayOfWeek. Because the DayOfWeekVariable changes only every 24 hours, we use the *CurrentTime* variable as trigger.

Select the *CurrentTime* variable, select the *Events* tab and pres the *Create Condition Template* button.



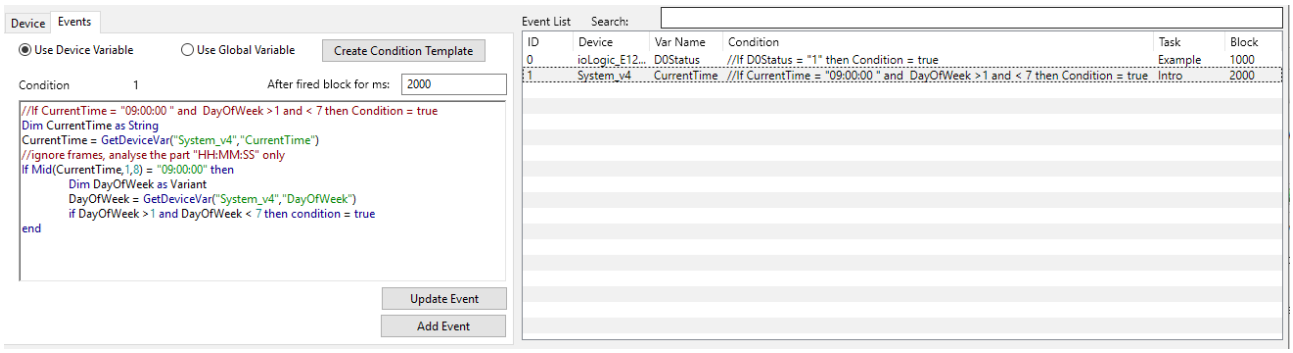
Change the Template to

```
//If CurrentTime = "09:00:00 " and  DayOfWeek >1 and < 7 then
Condition = true
Dim CurrentTime as String
CurrentTime = GetDeviceVar("System_v4","CurrentTime")
//ignore frames, analyse the part "HH:MM:SS" only
If Mid(CurrentTime,1,8) = "09:00:00" then
     Dim DayOfWeek as Variant
     DayOfWeek = GetDeviceVar("System_v4","DayOfWeek")
     if DayOfWeek >1 and DayOfWeek < 7 then condition = true
end
```

Maybe one is wondering about *Dim DayOfWeek as Variant*. If DayOfWeek was declared as string, then an operation such as *if DayOfWeek > 1* would couse a compile error, because a string cannot be compared with an integer. A variant behaves as string if the content is a string, or as integer if the content can be converted to integer. So the string “1” can be converted to the integer 1. This happens automatically.

Now the event is fired 25 Times (from “09:00:00:00” - “09:00:00:24” at 25 fps). Because we want to ensure that the event is only fired ocne, set the *After fired block for ms* to 2000 ms. Now press the *Add Event* Button

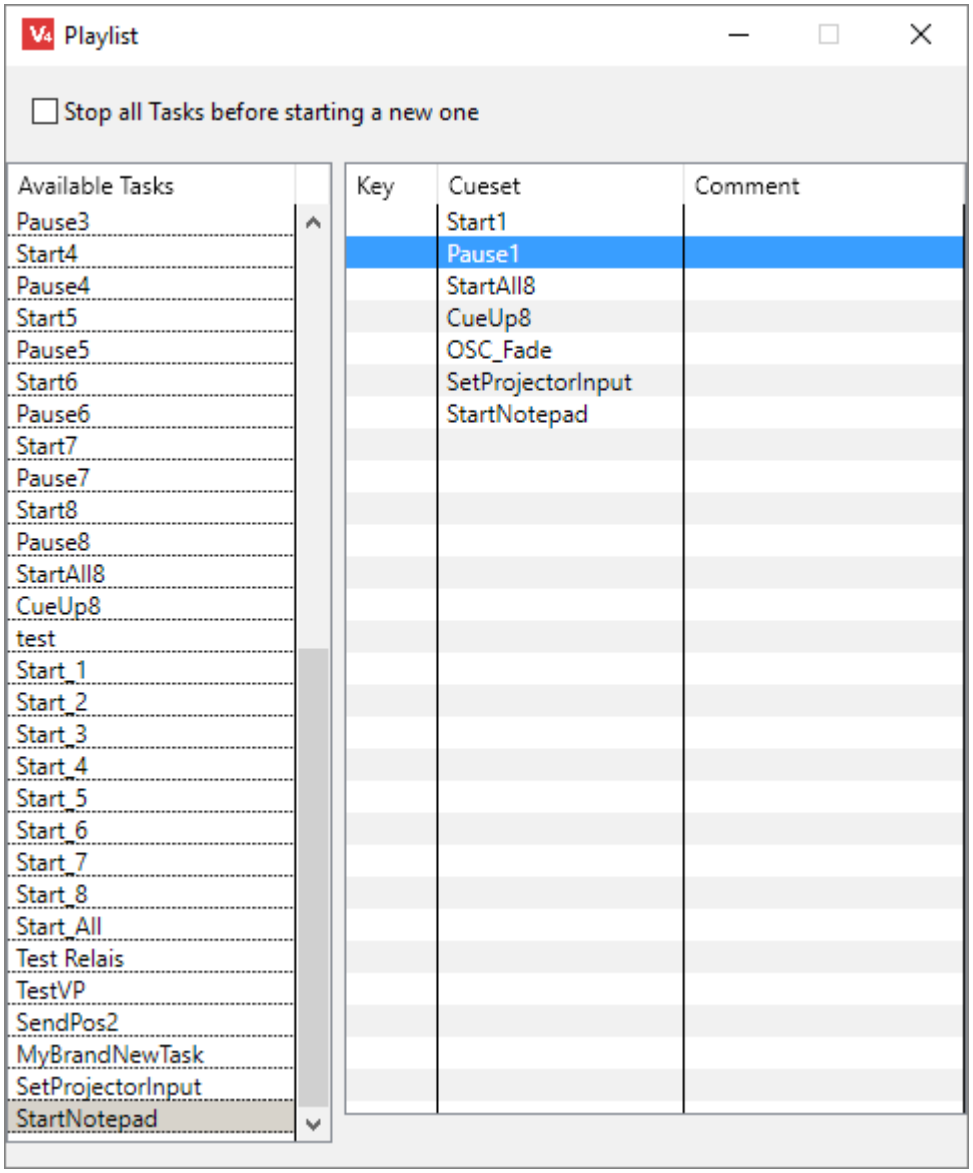
Finally, after assigning a task to the event, the result is



Simple GUI

Playlist

Via *Playlist* tool button or main menu *Playout -> Playlist* the Playlist window pops up.

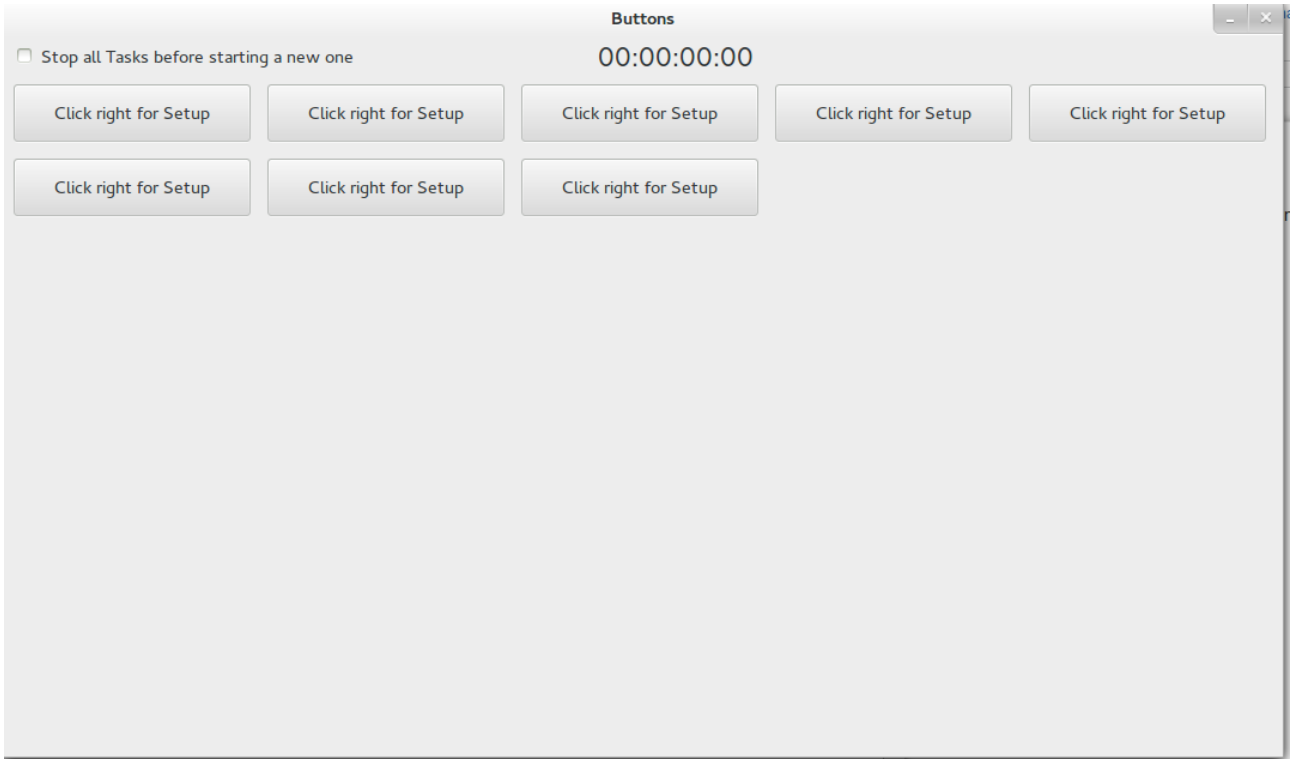


The left table lists all available tasks (Scripts, Cuelists and Timestrips). To fill the Playlist (right table) move the tasks in the left list via drag and drop to the right table. To launch a task, double click on it. Alternatively, a selected task can be launched by pressing the space bar. A shortcut key is assignable via playlist context menu as well as a comment.

If the button Stop all Tasks before playing a new one is pressed, then all running tasks are stoped before the new one launches.

CallButtons

CallButtons provide another way to have a GUI for end users. Click *Playout->CallButtons* to open the CallButton window.



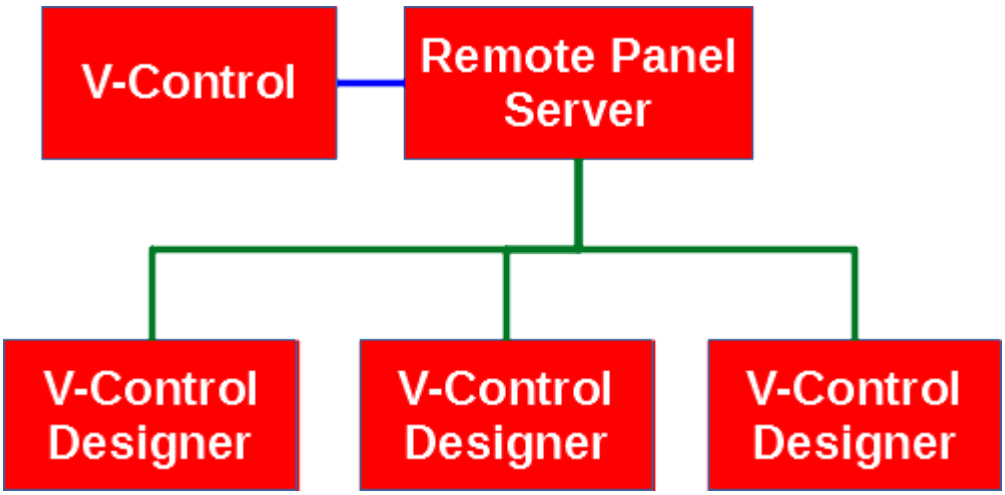
To add a new button click right on a free space and select the *Add Button* context menu. Each time the *Add Button* menu is clicked a new button occurs on the CallButton surface.

To assign a task to a button right click on the button and select the context menu *Select Task*. This opens a window with all available tasks. Select one and click *Add*. By default, the taskname is used as caption for the button. If you want another caption then the taskname you can click right on the button and select *Change Caption*

V-Control Designer

V-Control Designer is software that is used to create Graphical (touch) User Interfaces for V-Control. It communicates with the Remote Panel Server via TCP, and up to 5 V-Control Designer instances can have simultaneous access to a Remote Panel Server. The Remote Panel Server has two connections to V-Control Devices, the RemotePanelReceiver and the RemotePanelSender.

Since the Remote Panel Server, V-Control Designer and V-Control communicate via TCP, all applications can run on different computers or the same.



While V-Control and the Remote Panel Server is multi-platform software (Windows, MAC OSX and Linux) V-Control Designer is Windows only at the moment. Nevertheless, V-Control Designer can operate with Remote Panel Server operating on a Linux or MAC OS and Windows.

Installation

Remote Panel Server

Unzip the downloaded file and copy it to a directory with users write permissions. Then launch the exe file.

V-Control Designer

Unzip the downloaded file and copy it to a directory with users write permissions. **Then run vcredist_msvc2015_x64.exe** that is located in the V-Control Designer directory. Then launch the exe file.

Licensing

V-Control Designer is not open source such as V-Control. However, the program is free to use. There are no license fees for V-Control Designer, but for the Remote Panel Server, and this server is mandatory. Therefore you need to buy

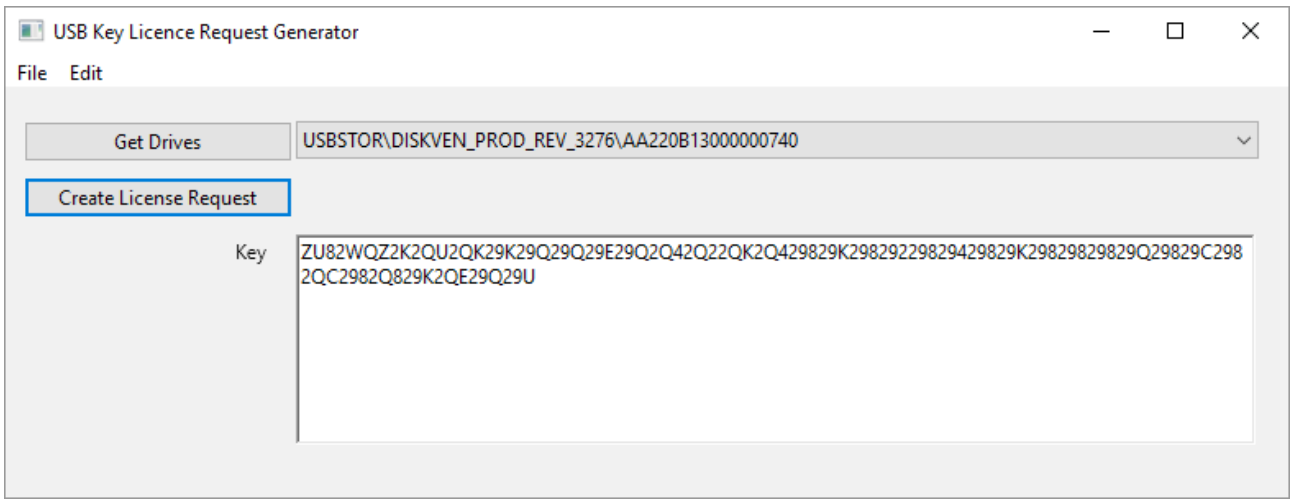
a license for the Remote Panel Server to use V-Control Designer. A license is only valid for one instance of the Remote Panel Server, and only one instance can run on a computer.

The Remote Panel server runs in Demo mode without a license. That means that every 30 minutes the network connection will be closed and the software must restart. There are no other restrictions. If purchased a license, use the hardware key (USB thumb drive) and the Server will run in licensed mode.

Hardware License Key

The Remote Panel Server needs hardware dongle as license key. The dongle will be send after purchase. Alternatively use the *File->License Request* menu to create a license request file:

- Purchase V-Control Remote Panel Server
- With the purchase, make a note that you will send a license request key later. Alternatively send the license request key with the purchase as note.
- Download and start the RemotePanelServer
- Connect a USB thumb drive to your computer
- select *File->License Request*

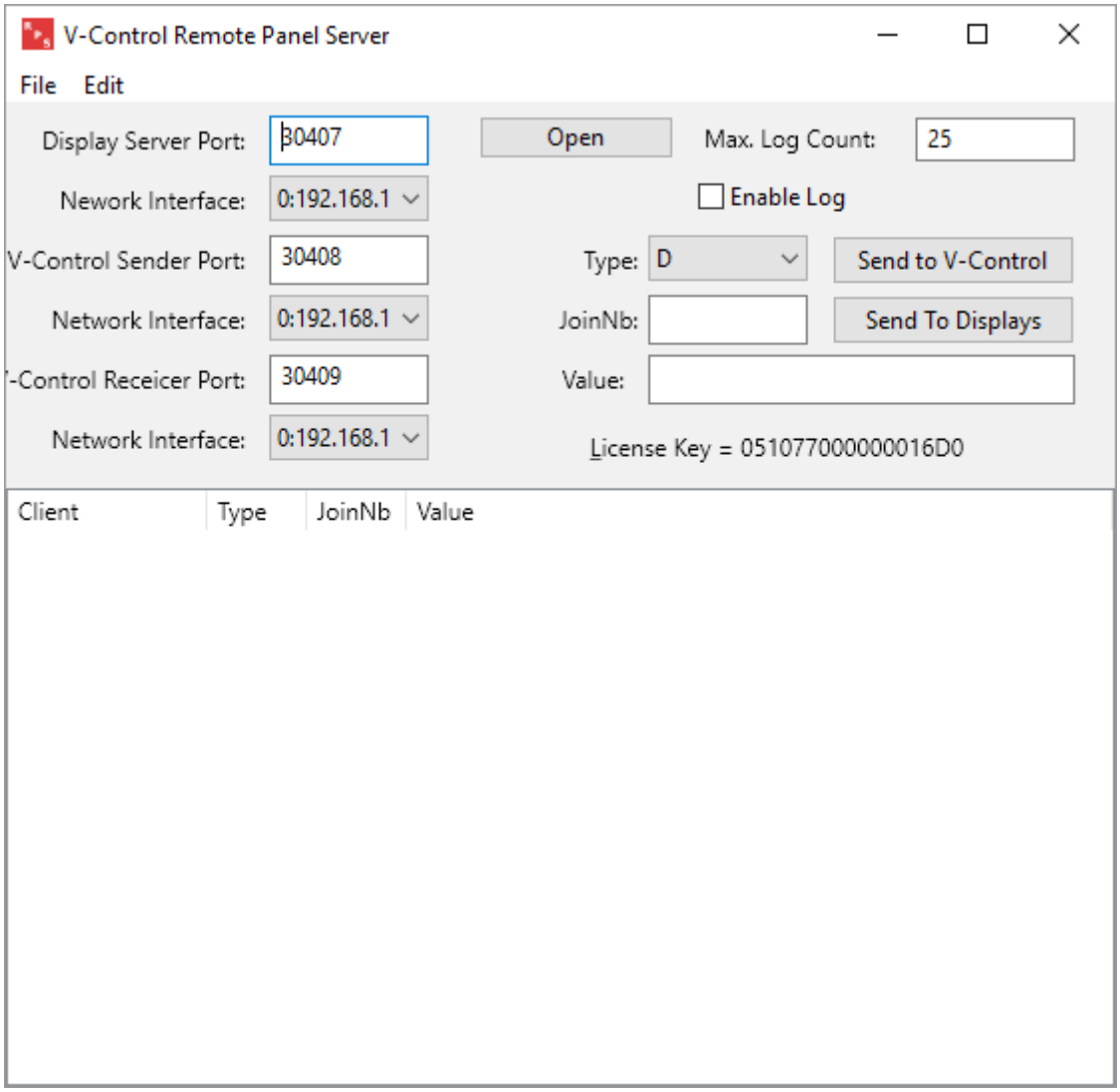


- Click the Get Drives button
- Select the USB drive in the drop down list right of that button
- Click Create License Request
- Copy the Key and send us an Email with that key
- We create a license file and send it via Email back to the purchaser
- Copy the file in the root directory of the USB drive used for the license request.

It will only work with the USB drive used for the request. Using another drive has the same result as using no key.

Remote Panel Server

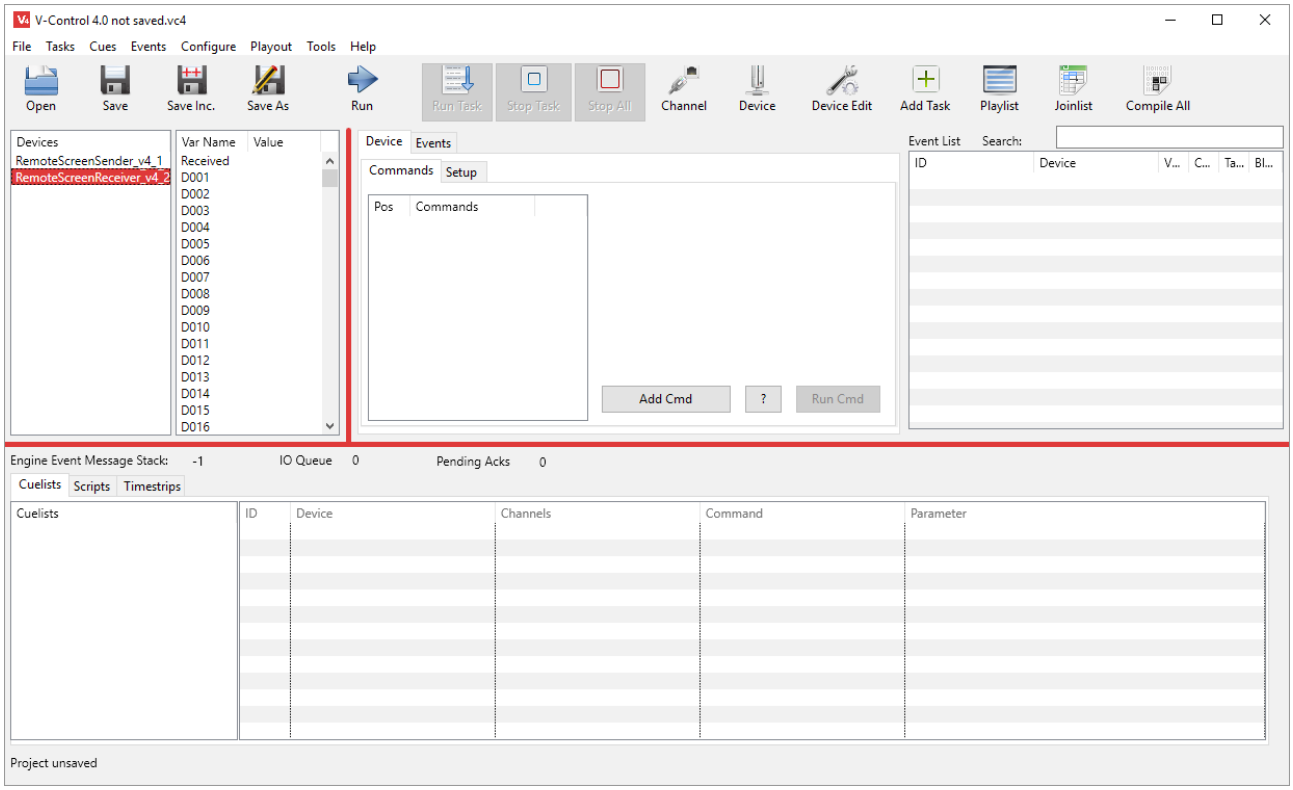
There is no direct communication between V-Control and V-Control Designer. The Remote Panel Server works as intermediary between the two programs.



First start the Remote Panel Server

The Remote Panel Server opens 3 server ports. The Display Server Port is used by V-Control Designer (up to 5 at the same time). V-Control Sender Port and V-Control Receiver Port are used by V-Control devices.

V-Control needs two new devices, The RemoteScreenSender and RemoteScreenReceiver. Create TCP-Client channels with the IP Address of the Remote Panel Server and the ports for Sender and Receiver.



Join Number Concept in Detail

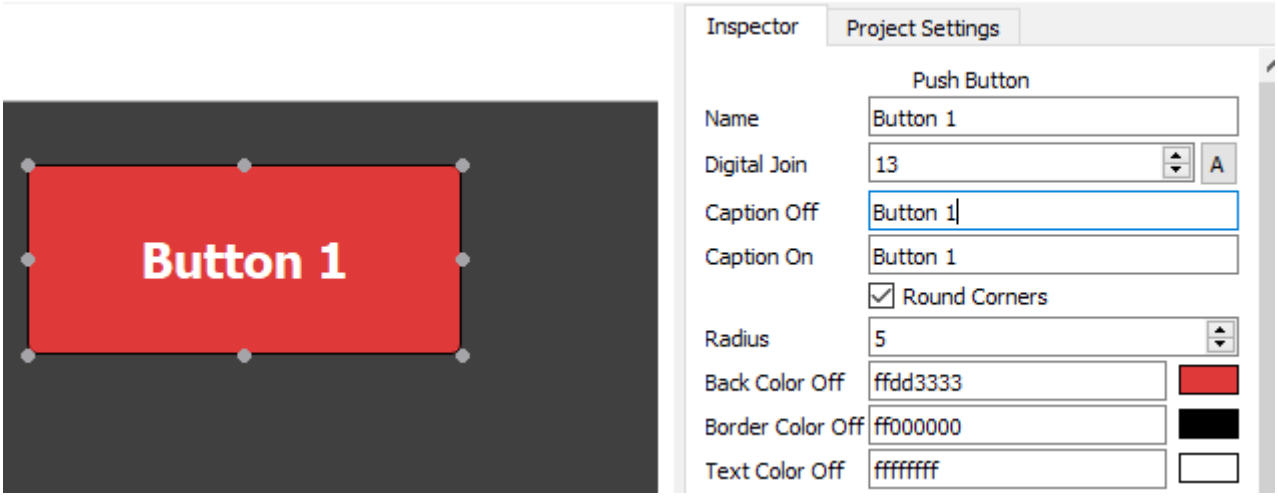
To interact with the V-Control Designer widgets, (buttons, labels, sliders etc.) a concept of join numbers is used. Join numbers define a connection to / from a widget. There are three different types of join numbers:

1. Digital Joins: They can have a value of 0 or 1.
2. Integer Joins: Any Integer value
3. String Joins: Any type of characters

Join numbers for each type can be between 1 and 999. So you have 999 digital joins, 999 integer joins and 999 string joins. All V-Control Designers connected to one Remote Panel Server share these join numbers. So if you have more then one Designer connected to the Remote Panel Server, and they show different GUI's (Graphical User Interface), take care that the join numbers don't overlap.

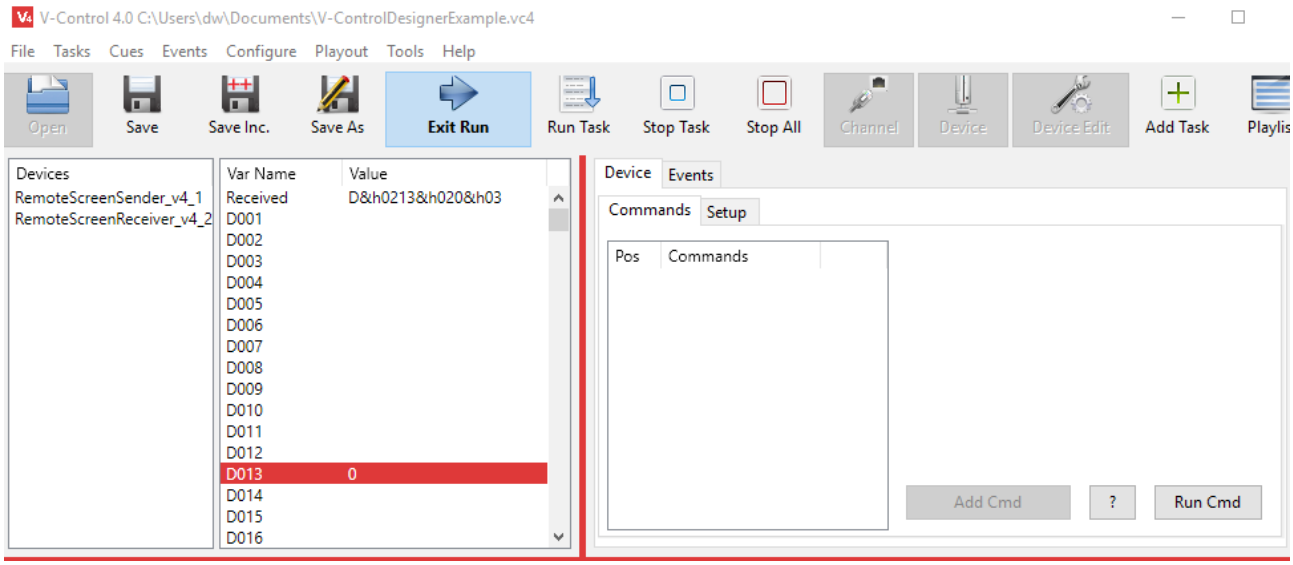
But in some cases it can be useful to share some join numbers on different GUI's. If you have information that is needed on all displays, then you can use a single string join number and have labels on all GUI's with this number. Then the information appears everywhere.

Below there is a button with some of its properties. Notice the second one, Digital Join. Because a button can only have the state On or Off, the digital Join represent its state.

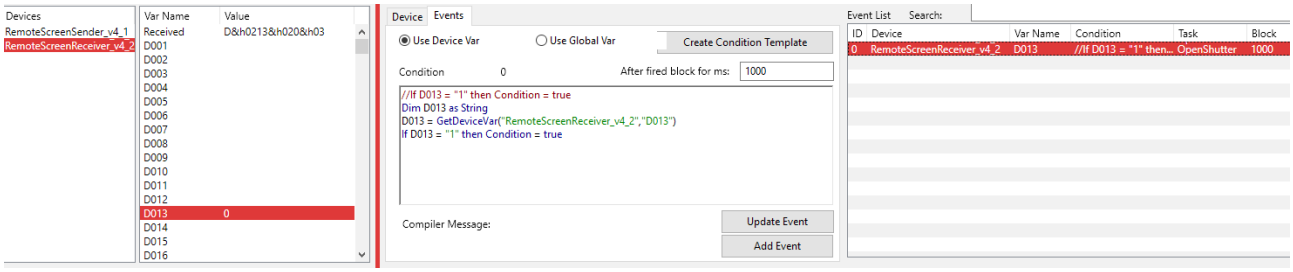


If the button is pressed, it sends a message with its type (Digital in this case), join number and 1 as value. If the button is released it sends the same message but 0 as value.

In V-Control, there is a device RemoteScreenReceiver. Per default, this device has device variables D001..D100 for digital joins, I001-I100 for integer joins and S001..S100 for string joins. If more join numbers needed add them in the Device Editor

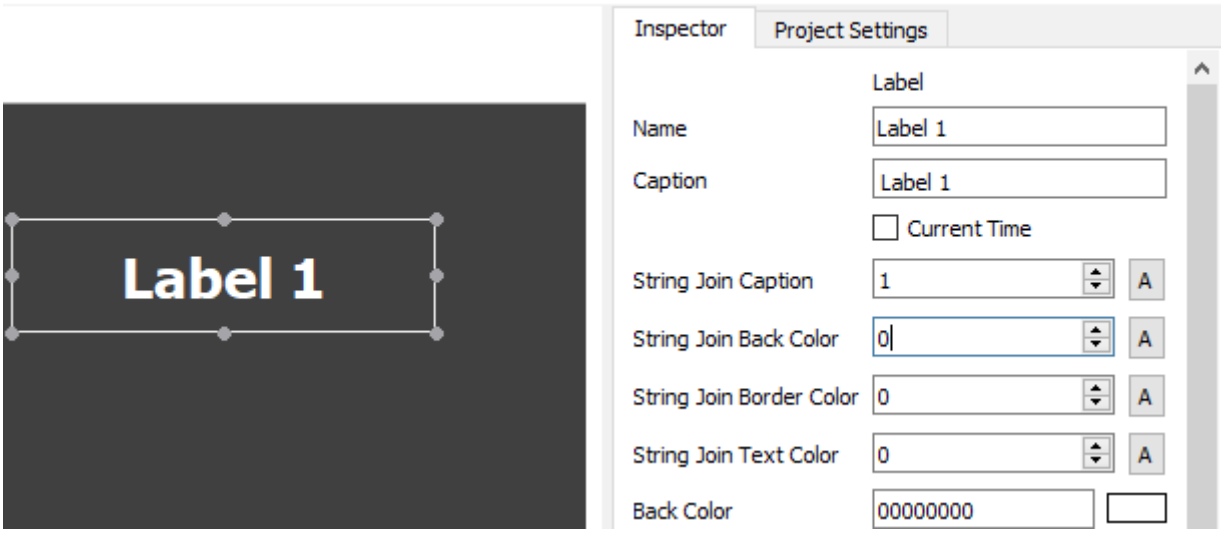


If V-Control Designer sends a message with a join number and a value, then the RemoteScreenReceiver updates a device variable (D013 in this example). This generates an event, and we can use this event to run a task.



A detailed description how events work in V-Control is at Events

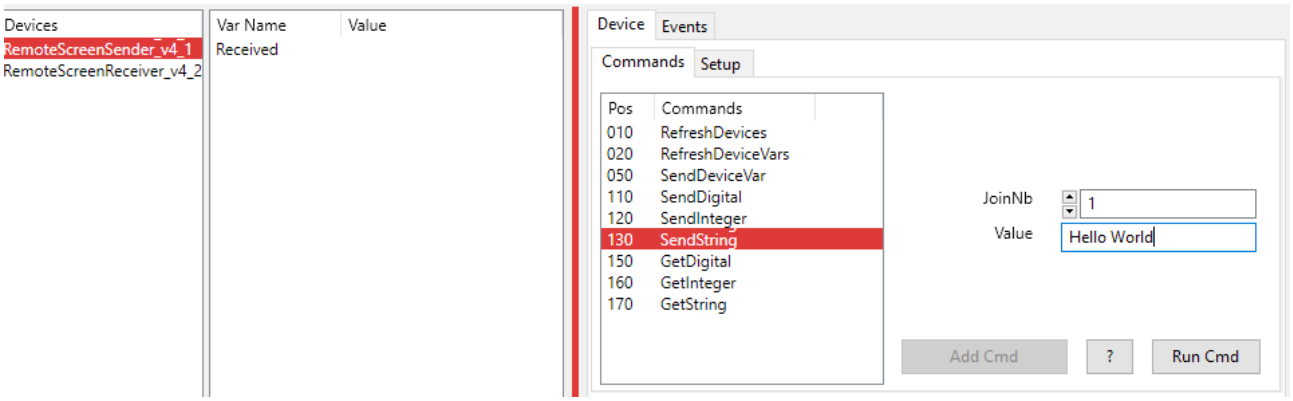
Lets Do the other way. Below is a label with the string String Join Caption 1.



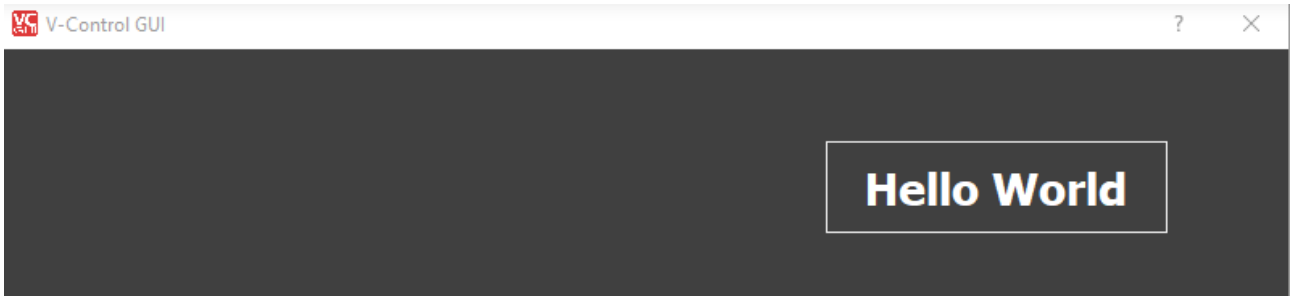
We can now use the device RemoteScreenSender to send a message to this join number. The content of this message will be shown in the label.

Maybe you notice that the label has additional string joins. String Join *Back Color*, String Join *Border Color* and String Join *Text Color* can receive messages to change the corresponding colors.

The RemoteScreenSender device has commands to send messages to the Remote Panel Server



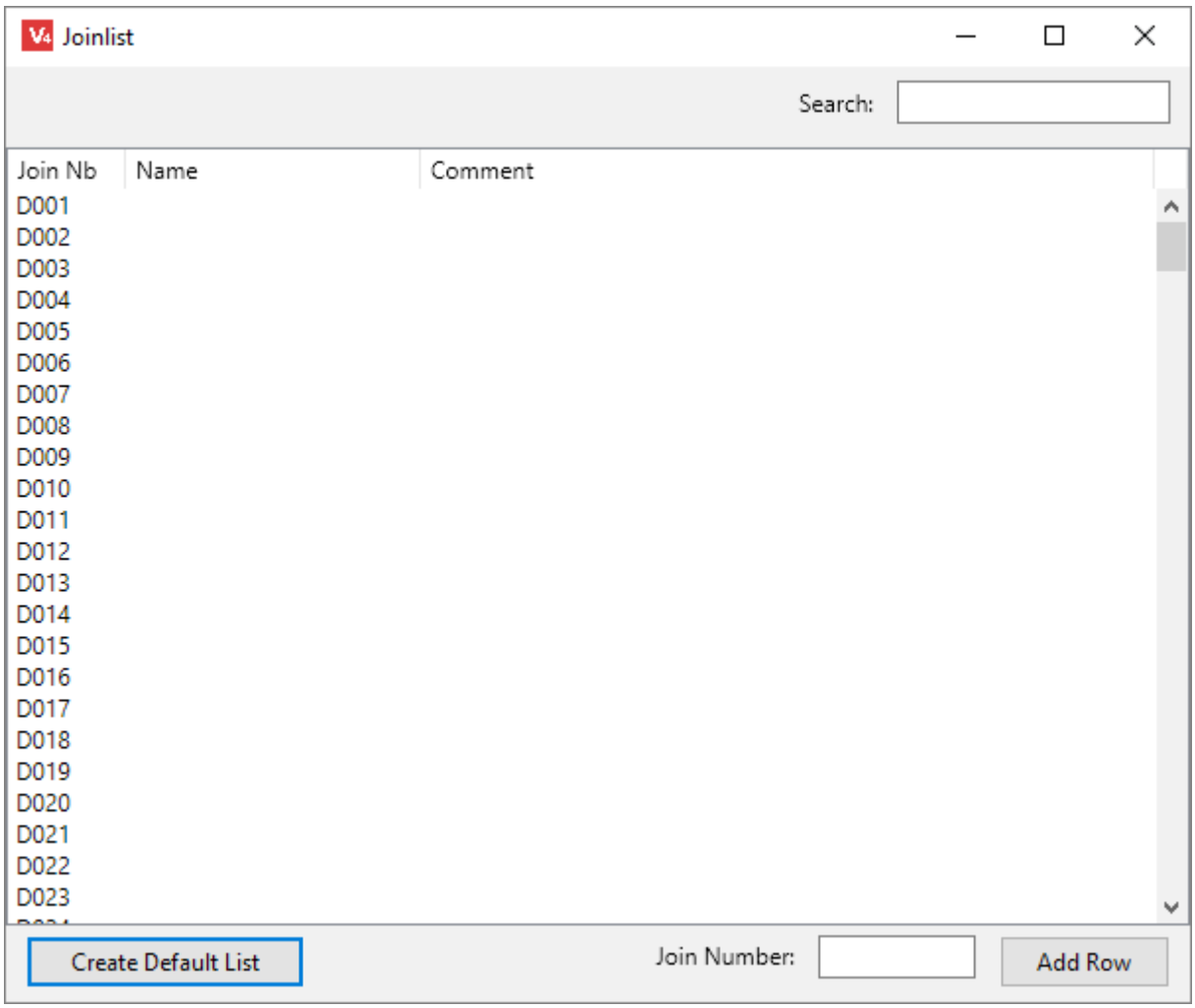
The command SendString send as message to a string join, in this case join number 1 which we used for the caption join of the label. If we run this command, the label changes its caption.



More Information about RemoteScreenReceiver and RemoteScreenSender can be found at Device Drivers.

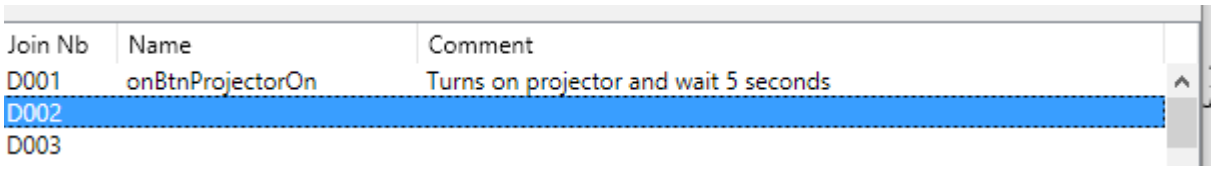
Manage Join Numbers

In large projects it is easy to lose track of the join numbers. For that reason, V-Control can help managing join numbers. In V-Control, click the Join List tool button.

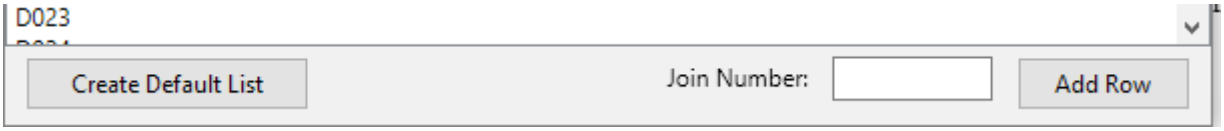


This list is used to make notes what a join number is used for. If you have a button with join number 1, and the *RemoteScreenReceivers* device variable D001 change event is used to trigger a task that turns on a projector, edit the

row with *D001* in the following way:

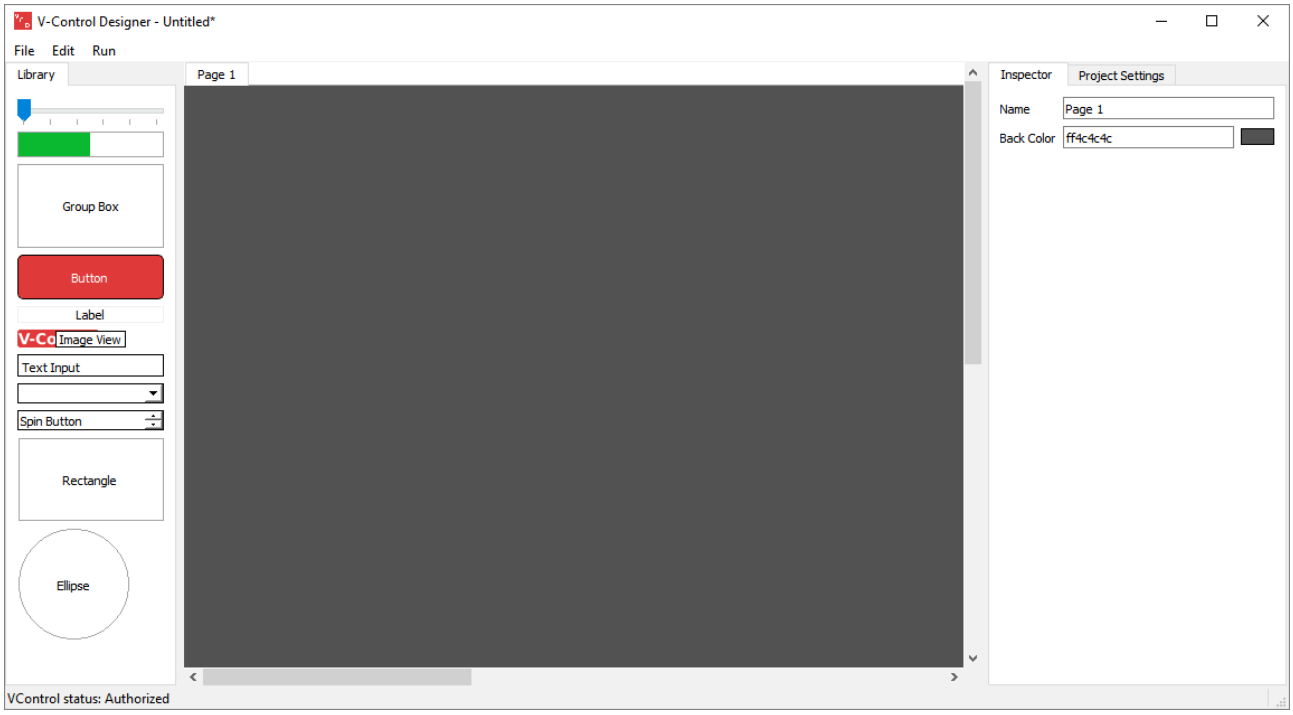


It is good practice to give the widgets meaningful names. The task that handles the change event has the same name with prefixed “on”. Use the comment column for further useful information. Because the list could be very long, the Search field helps to find a row of interest. Anything typed in here that matches any row content works as a filter. Only rows that match the search field content are shown.



At the bottom of the Join List, the Create Default List button creates a default list of join numbers (D001..D100, I001..I100 and S001..S100). If more join numbers are needed enter a new one in the Join Number field. The format is D101 for a new digital join number 101. I221 creates an integer join with number 221 and S123 a new string join with number 123. Click the Add Row button to insert the new join number.

GUI Overview

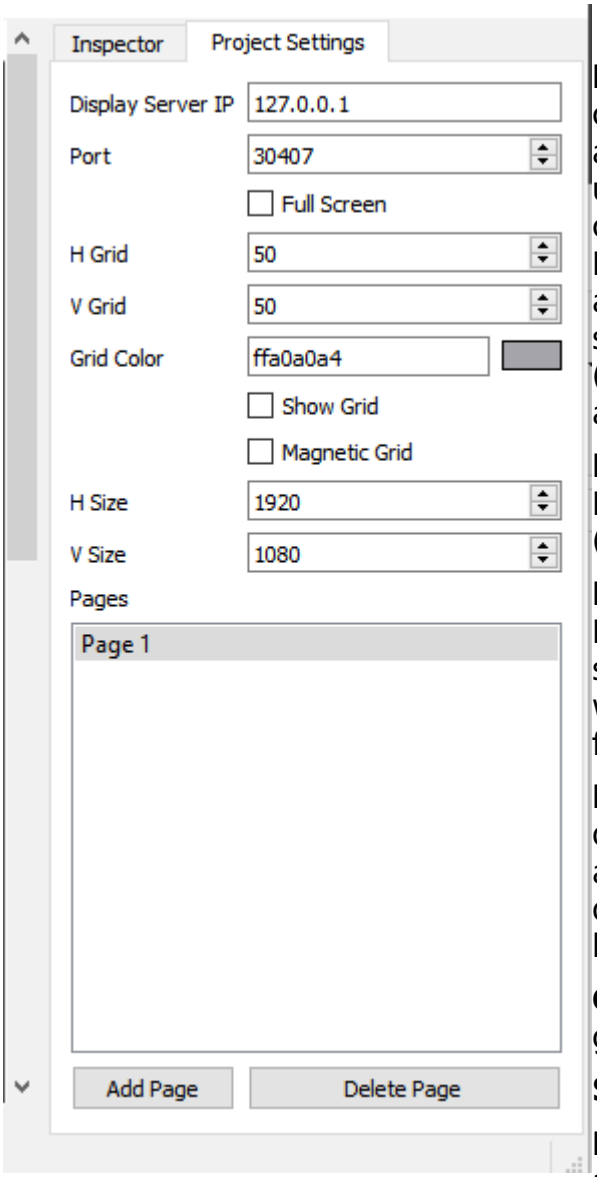


The Main Window is divided into three parts:

1. The Library (left) Here are the widgets that can be dropped to the page area. These widgets are used to build the GUI.
2. Page Area (center) This is the area where you drop and position your widgets.
3. Project Settings / Inspector (right) Here you can set up the project and customize the widgets.

Click the Run menu item to switch to run mode.

Project Settings



Display Server IP: To establish a connection between V-Control Designer and Remote Panel Server, you need to set up the TCP connection. Display Server IP contains the IP Address of the Remote Panel Server. If both, Remote Panel Server and V-Control Designer running on the same system, this should be localhost (127.0.0.1). In other cases the use the real address.

Port: The TCP port that is used by Remote Panel Server to connect to Displays (Display Server Port).

Full Screen: If checked, V-Control Designer will cover the whole screen if switched to Run mode. There will be no window decoration. To exit the run mode if full screen view, press ESC.

H Grid, V Grid: In Edit Mode, the user can display a grid that helps to position and align the widgets. H Grid and V Grid determine the distance between the grid lines.

Grid Color: Determines the color of the grid.

Show Grid: If checked, the grid is shown

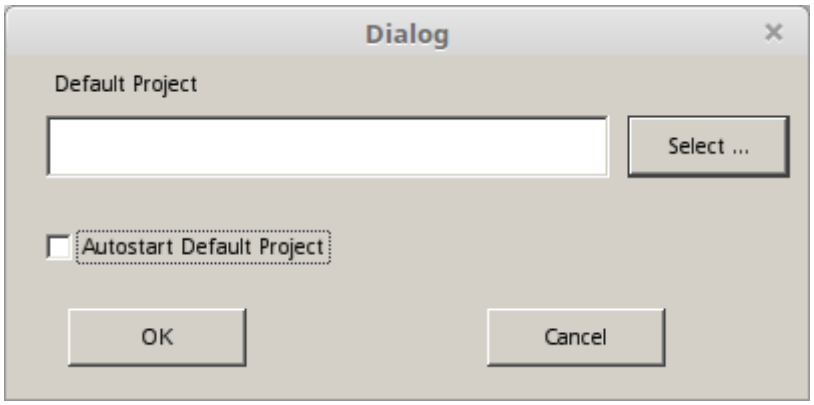
Magnetic Grid: If checked, widgets will allign to the grid if you change their position by dragging them.

H Size, V Size: Determines the size of all pages in a single project. Usually this should match the Display size if you want to run in Full Screen mode.

Add Page, Delete Page V-Control Designer can use multiple pages within one project. Here you can add new ones or delete existing ones.

Settings

Main Menu File → Settings



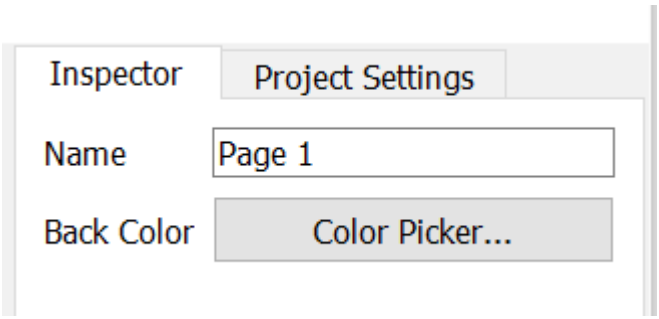
Default Project Press the Select button to choose a project that is loaded at startup.

Autostart Default Project If a valid project is loaded by default, enabling this check box will initiate run mode at startup.

Pages

A V-Control Designer project consist of one or more pages. A page act as container for widgets and can use the entire screen in run mode if Full Screen is selected.

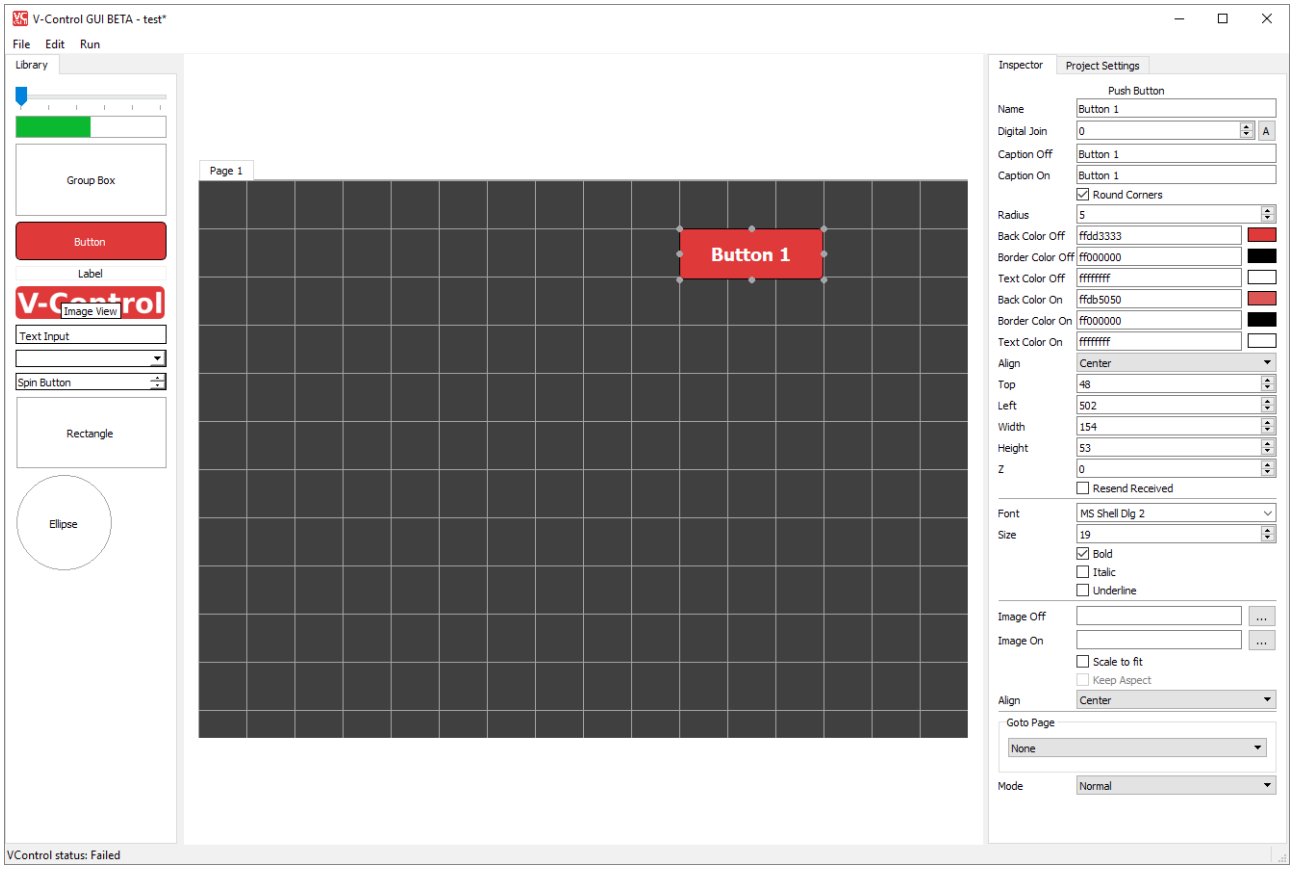
To add or delete pages switch to Project Settings.



Page Properties:

To access the page properties select the Inspector and click on an empty area in the page. Then you can change the page Name and Back Color. The name has to be unique.

Buttons



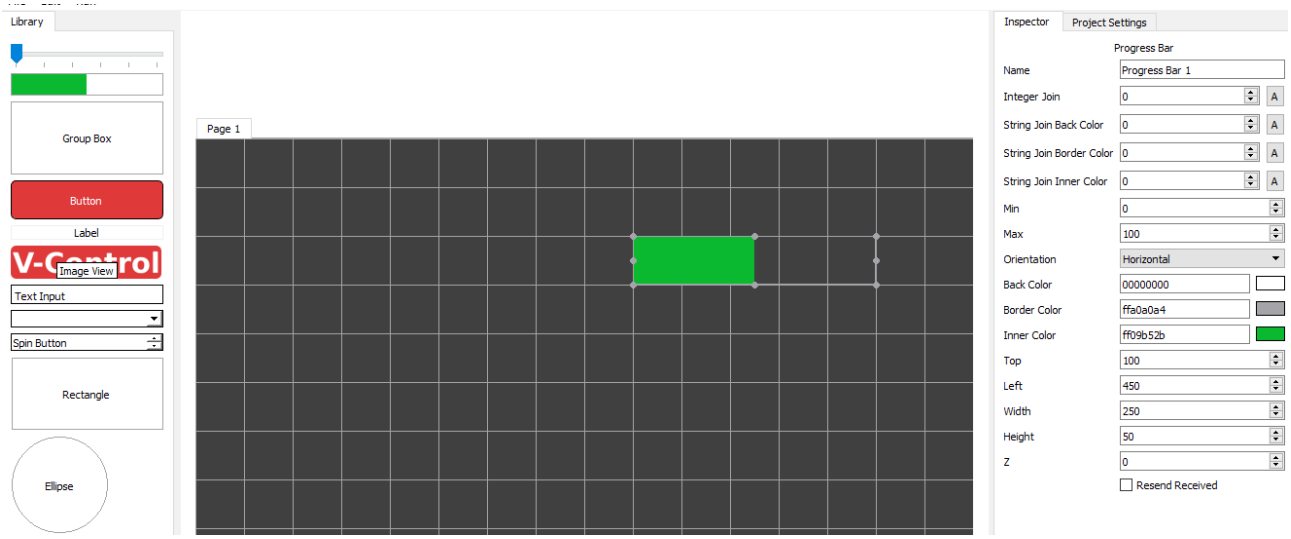
Button properties are:

- Name: The name of the button. Try to use meaningful names, this will help in complex projects to keep the overview.
- Digital Join: The join number for the button pressed and release event. If pressed, this join number will have the value 1, if released then 0. The A button automatically selects a free join number.
- Caption On: Button caption in pressed state
- Caption Off: Button caption in unpressed state
- Radius: If > 0 then the button has round corners.
- Back Color Off: The back color of the button in unpressed state
- Border Color Off: The border color of the button in unpressed state
- Text Color Off: The Text color of the button in unpressed state
- Back Color On: The back color of the button in pressed state
- Border Color On: The border color of the button in pressed state
- Text Color On: The Text color of the button in pressed state
- Align: Alignment of the caption
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the

RemoteScreenSender. Then the RemoteScreenreceiver doesn't know that the value has changed and does not update the corresponding device variable.

- Font: The font of the Caption
- Size: Font size of the caption
- Bold, Italic, Underline: Style of the Caption
- Image Off, Image On: Image to show in unpressed and pressed state
- Scale to fit: Scales the image that it fits in the widgets dimensions.
- Keep Aspect: Keep aspect ratio of the image
- Goto Page: Switch to another Page
- Mode: Normal: If pressed, it automatically gets unpressed. Toggle: If pressed, it stays pressed until it was pressed again. One Time: If pressed it stays pressed. The only way to reset is to send a message with value 0.

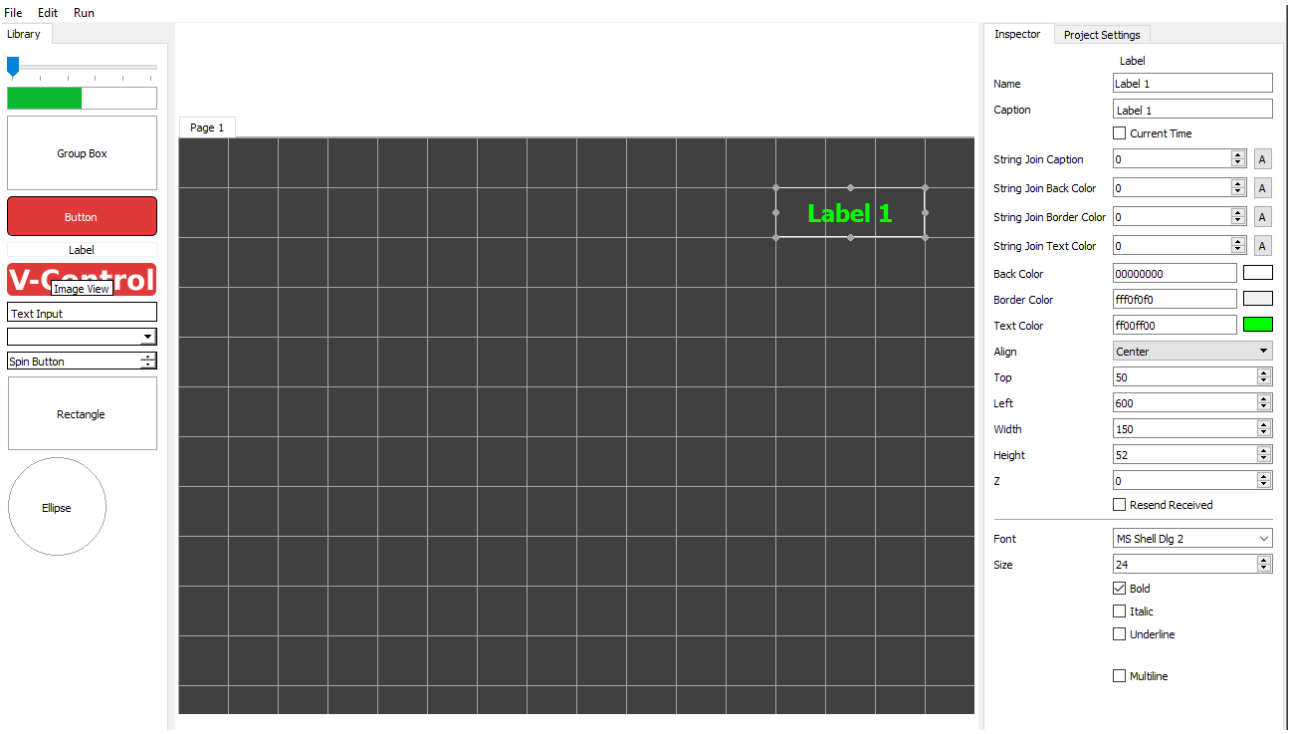
Progress Bar



- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Integer Join: The join number to set the progress. The A button automatically selects a free join number.
- String Join Back Color: The back color of the progress bar. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- String Join Border Color: The border color of the progress bar
- String Join Inner Color: The inner color of the progress bar. Use all the String Color Joins to change the color of the widget by V-Control.
- Min: The minimum integer value that the progress bar can show (no progress)
- Max: The maximum integer value that the progress bar can show (full progress)
- Top, Left, Width, Height: Position and size of the button

- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the RemoteScreenSender. Then the RemoteScreenreceiver doesn't know that the value has changed and does not update the corresponding device variable.

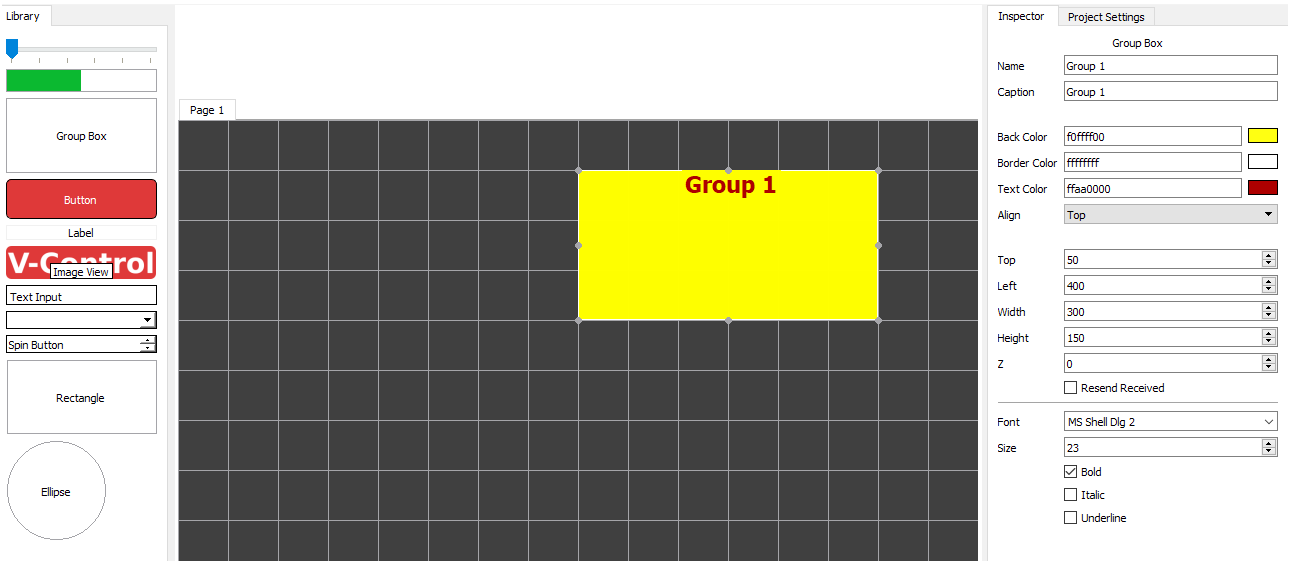
Label



- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Caption: The text that is displayed
- String Join Caption: The join number to set the Caption.
- String Join Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- String Join Border Color: The border color of the label
- String Join Text Color: The Text color of the label. Use all the String Color Joins to change the color of the widget by V-Control.
- Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the label

- Text Color: The Text color of the label.
- Align: Text alignment
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the RemoteScreenSender. Then the RemoteScreenreceiver doesn't know that the value has changed and does not update the corresponding device variable.
- Font: The font of the Caption
- Size: Font size of the caption
- Bold, Italic, Underline: Style of the Caption
- Multilie: Show multiple lines of text

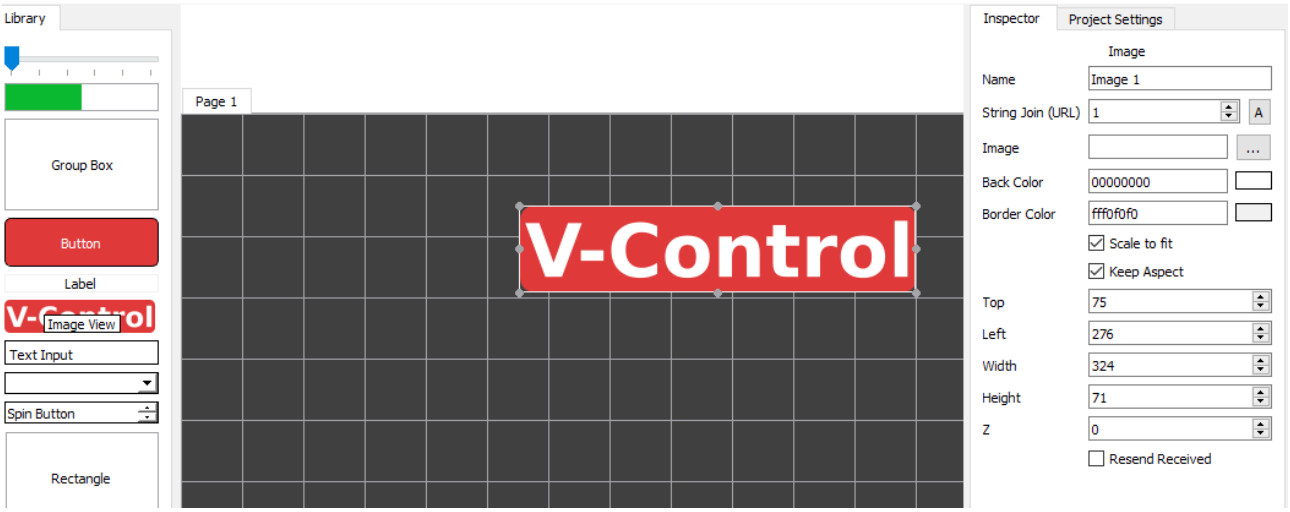
Group Box



- Name: The name of the button. Try to use meaningful names, this will help in complex projects to keep the overview.
- Caption: The text that is displayed
- Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the label
- Text Color: The Text color of the label.
- Align: Text alignment
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Font: The font of the Caption

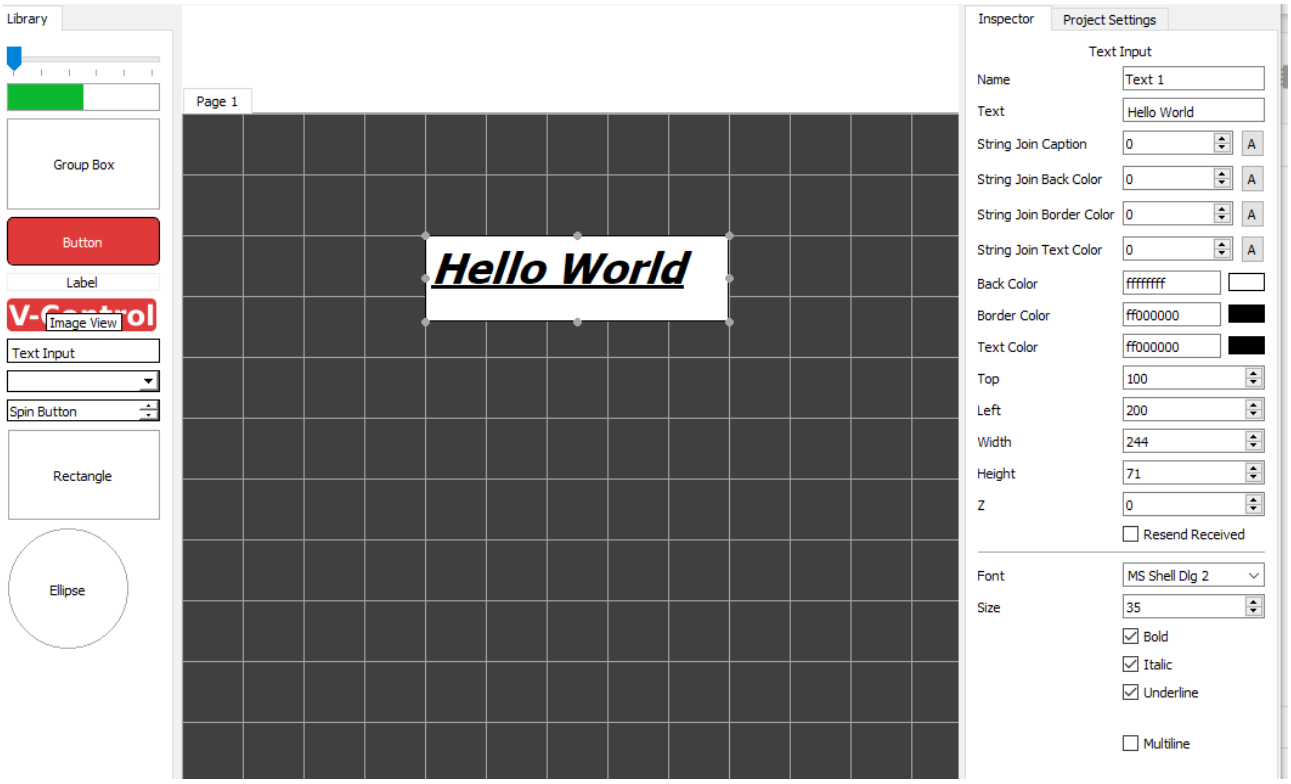
- Size: Font size of the caption
- Bold, Italic, Underline: Style of the Caption

Image



- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- String Join (URL): Send the path of an image to change the image by V-Control
- Back Color: The back color of the widger. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the widget
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.

Text Input

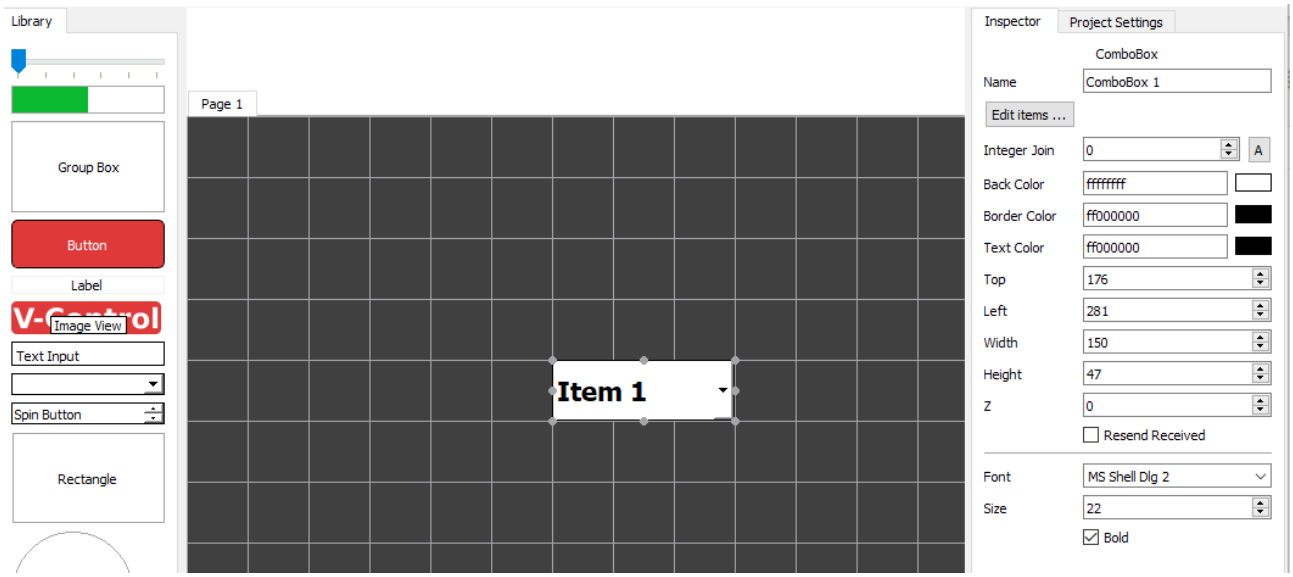


- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Text: The text that is displayed
- String Join Caption: The join number to set the Text.
- String Join Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- String Join Border Color: The border color of the label
- String Join Text Color: The Text color of the label. Use all the String Color Joins to change the color of the widget by V-Control.
- Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the label
- Text Color: The Text color of the label.
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the RemoteScreenSender. Then the RemoteScreenreceiver doesn’t know that the value has changed and does not update the corresponding device

variable.

- Font: The font of the Caption
- Size: Font size of the caption
- Bold, Italic, Underline: Style of the Caption
- Multilie: Show multiple lines of text

Combo Box



Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview. Edit Items: Opens an editor to edit the items in the combo box Integer Join: The join number to select an item. Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers. Border Color: The border color of the label Text Color: The Text color of the label. Top, Left, Width, Height: Position and size of the button Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets. Font: The font of the Caption Size: Font size of the caption Bold: Style of the Caption

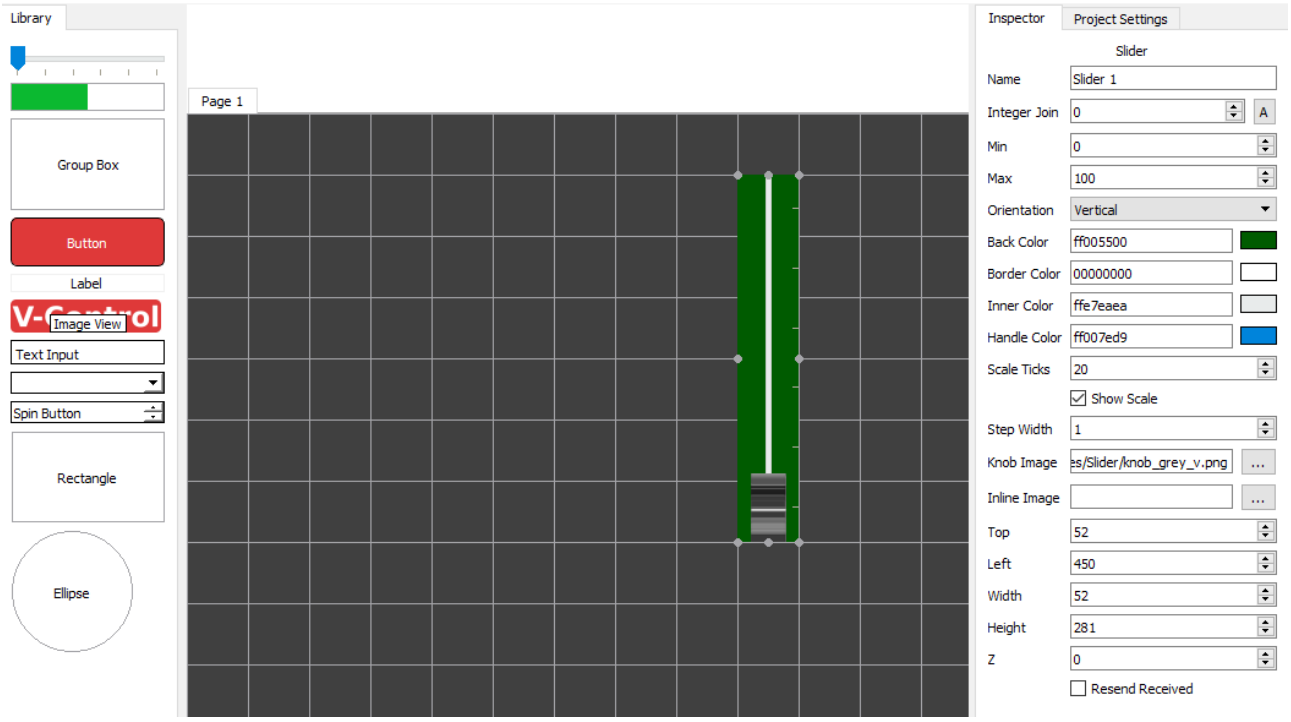
Spin Button

- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Integer Join: The join number
- Min: The minimum integer value
- Max: The maximum integer value
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the

RemoteScreenSender. Then the RemoteScreenreceiver doesn't know that the value has changed and does not update the corresponding device variable.

- Font: The font of the items
- Size: Font size of the items
- Bold: Style of the items

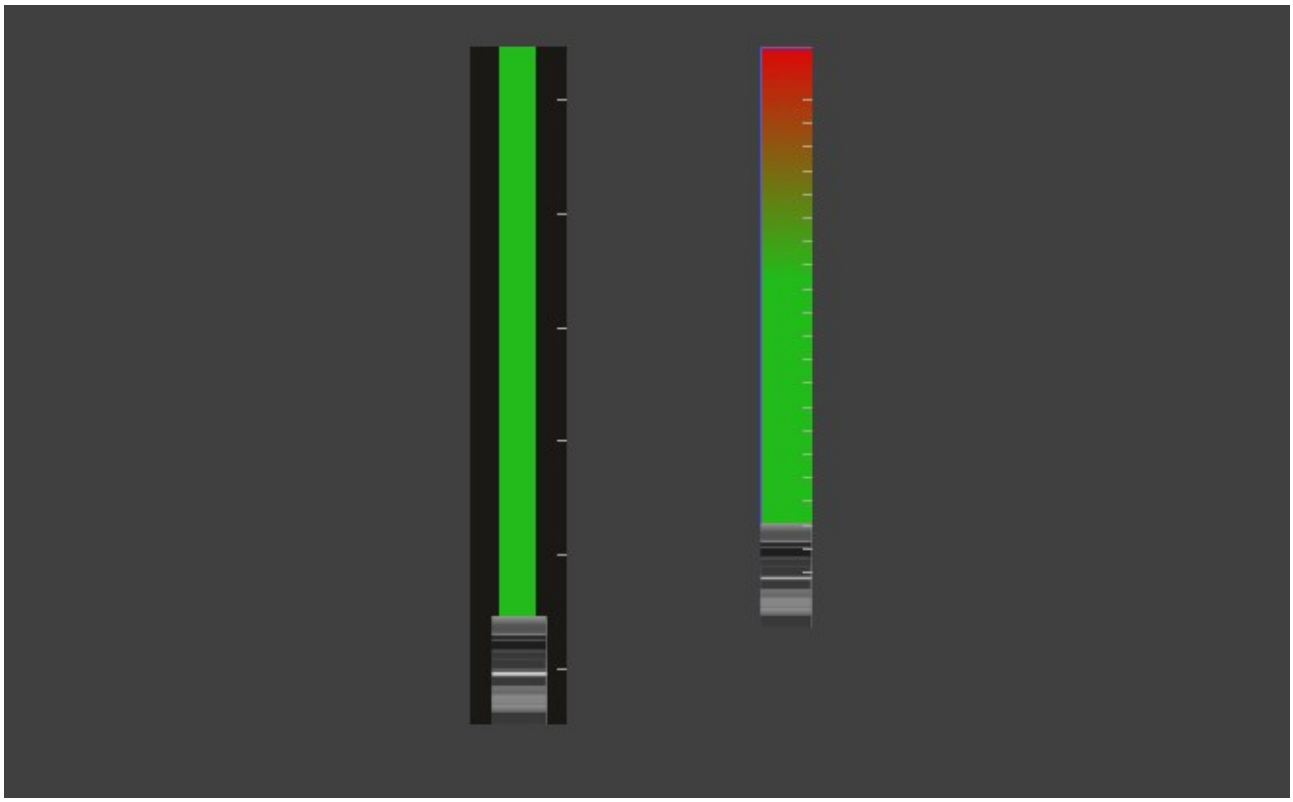
Slider



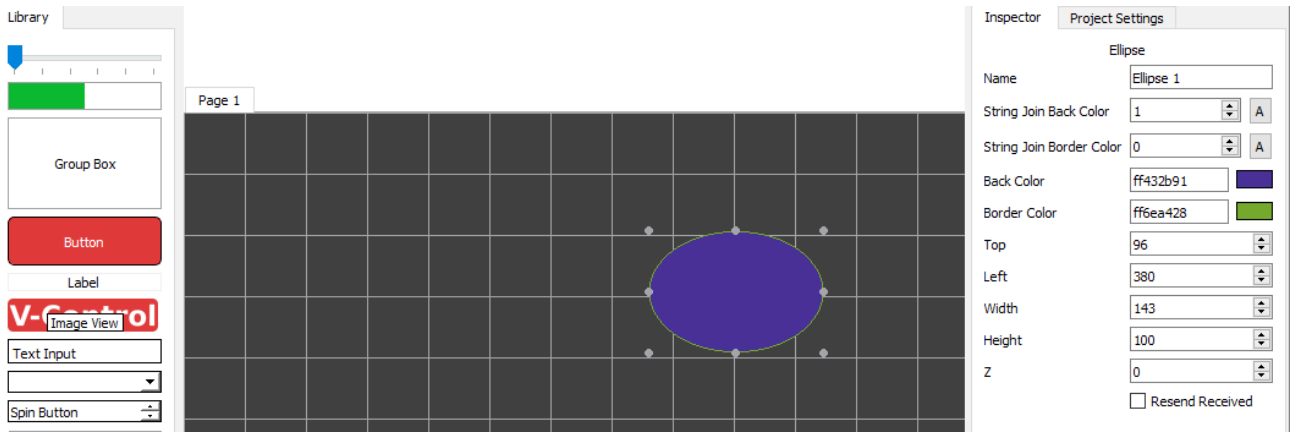
- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Integer Join: The join number
- Min: The minimum integer value that the progress bar can show (no progress)
- Max: The maximum integer value that the progress bar can show (full progress)
- Orientation: Horizontal or vertical orientation
- Back Color: The back color of the widget. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the widget
- Inner Color: The inner color of the widget.
- Handle Color: Color of the handle
- Scale Ticks: every Scale Tics pixel a marker is set
- Show Scale: Show or hide the scale
- Sep Width: Minimum width of a step
- Knob Image: Image for the handle

- Inline Image: image for the inner part
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.
- Resend Received: If a widget receive a message, then it can send the message back the message to the RemoteScreenReceiver. Imagine the button receives a message with value 0, the message was sent from the RemoteScreenSender. Then the RemoteScreenreceiver doesn't know that the value has changed and does not update the corresponding device variable.

The image properties of the slider allow custom sliders. It depends only on the graphical skills how they look.



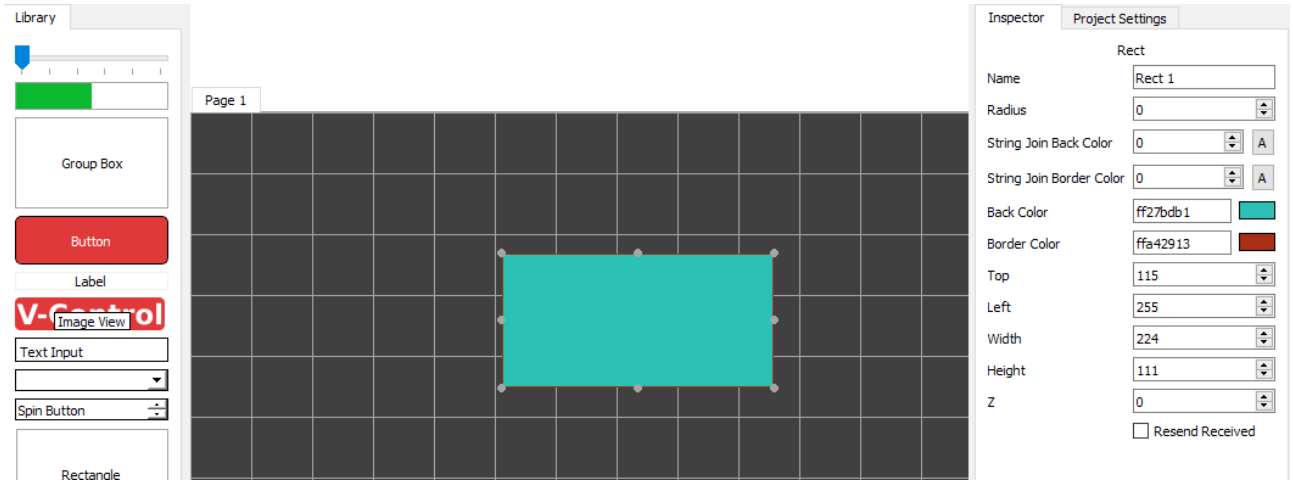
Ellipse



- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- String Join Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- String Join Border Color: The border color of the label
- Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the label
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.

The Ellipse is often used a background for other widgets to visualize a function group.

Rectangle



- Name: The name of the widget. Try to use meaningful names, this will help in complex projects to keep the overview.
- Radius: Corner radius.
- String Join Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- String Join Border Color: The border color of the label
- Back Color: The back color of the label. Color values are in the format AA,RR,GG,BB. AA = alpha (00..FF), 00 means transparent, FF full visible, RR = red (00..FF), GG= green (00..FF) and BB = blue (00..FF). All Color values are two digit hex numbers.
- Border Color: The border color of the label

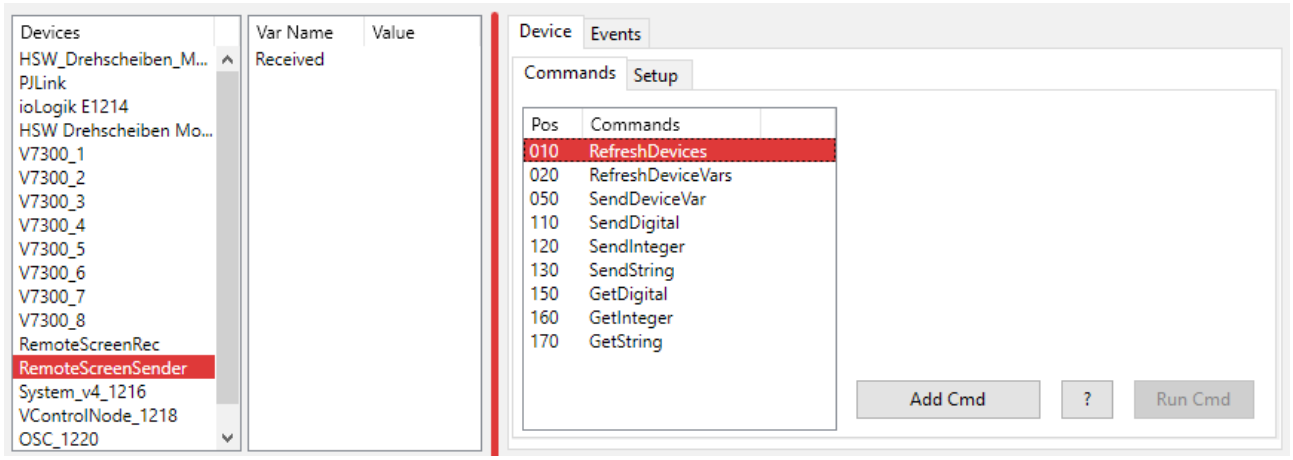
- Top, Left, Width, Height: Position and size of the button
- Z: The Z-Order of the button. The higher this value, the more on top the widget is displayed if overlapping with other widgets.

The Rectangle is often used a background for other widgets to visualize a function group.

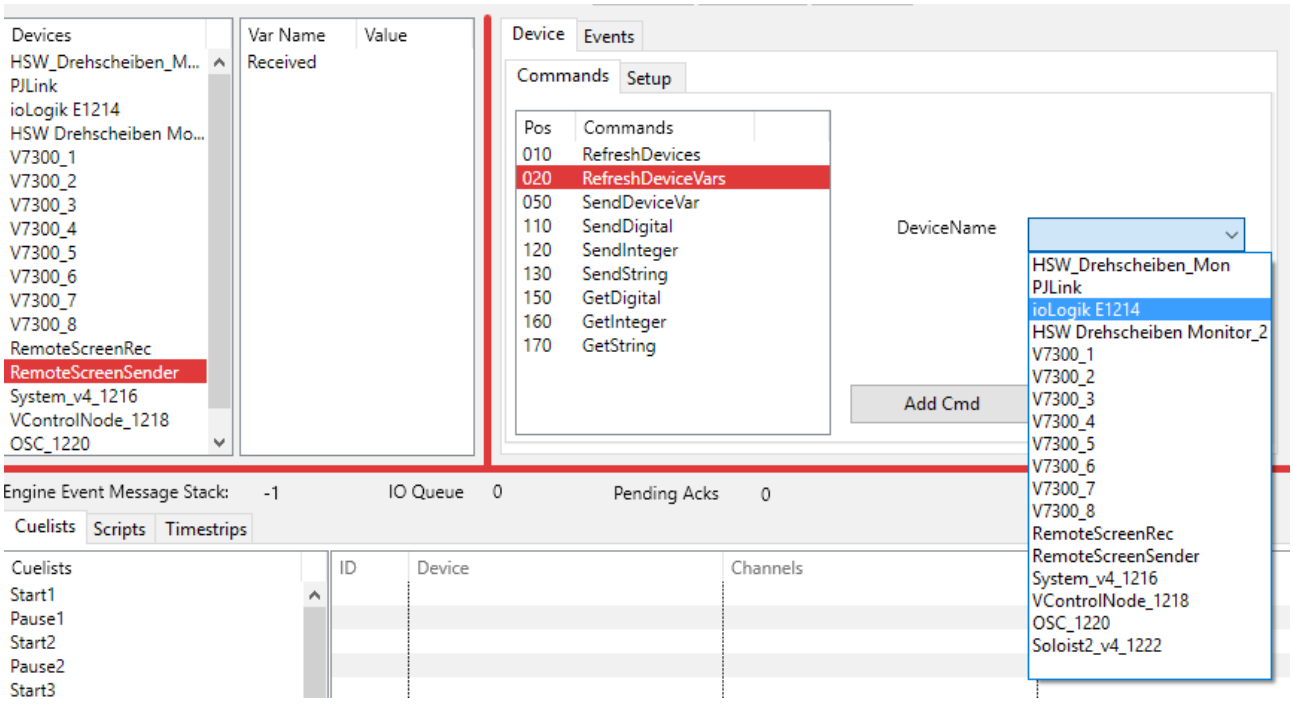
RemoteScreenSender

The RemoteScreenSender is used to send commands and / or information to a V-Control Designer instance. V-Control Designer elements (buttons, sliders, labels etc.) are referenced by join numbers. The RemoteScreenSender can send simple messages and device variables.

Sending Device Variables

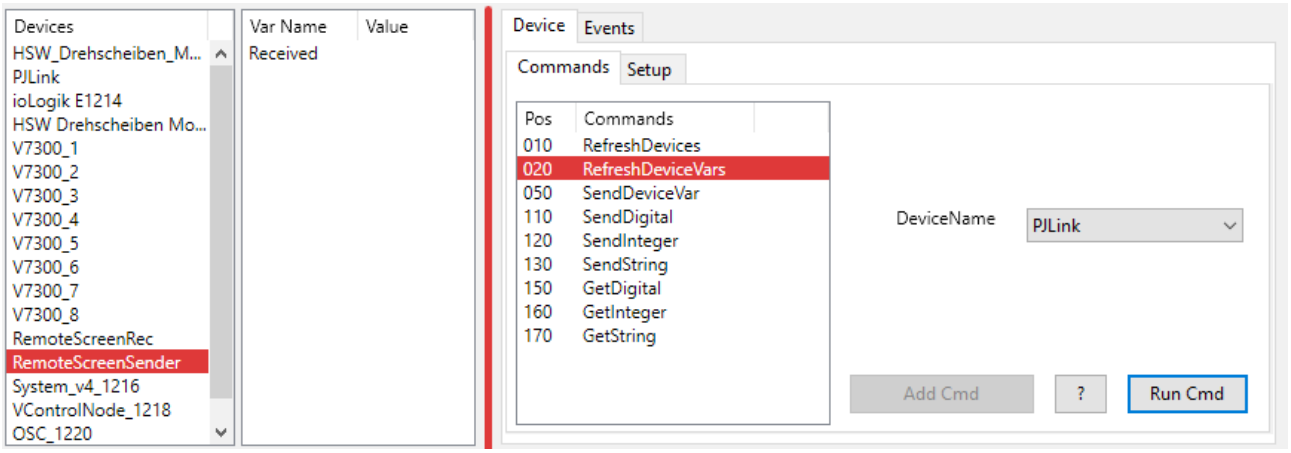


To send a device variable, the RemoteScreenSender needs to know which device and which variable. To get a list of all available devices, the *RefreshDevices* command is used. Switch to *Run* mode and press the *Run Cmd* button if the *RefreshDevices* command is selected. Nothing happens for now, but next select the *RefreshDeviceVars* command. The drop down box *DeviceName* now holds a list of all devices in the current project.

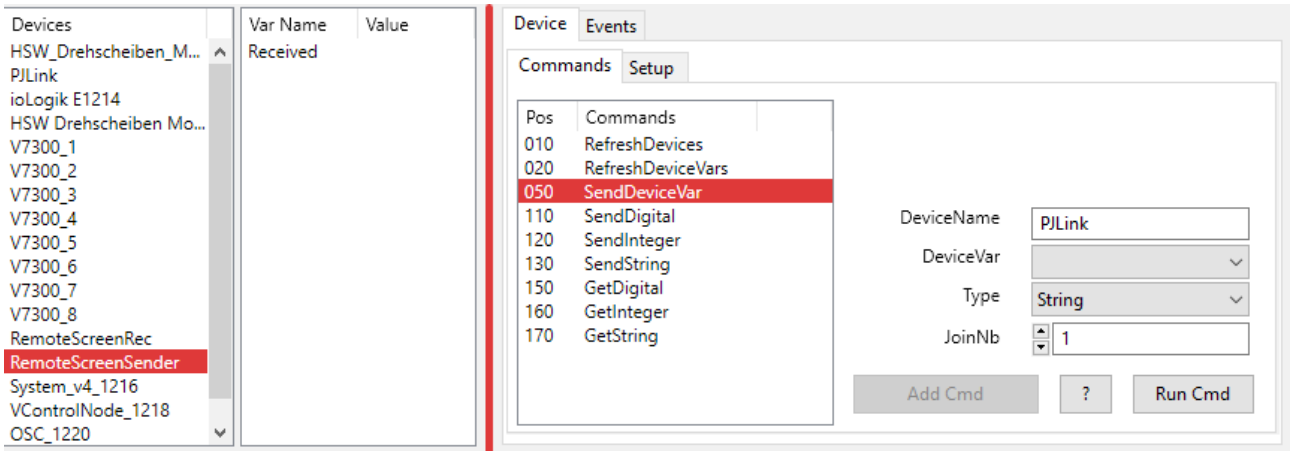


The *RefreshDevices* command has to be executed only once, or if devices were added or deleted from the current project. The list is stored with the project file.

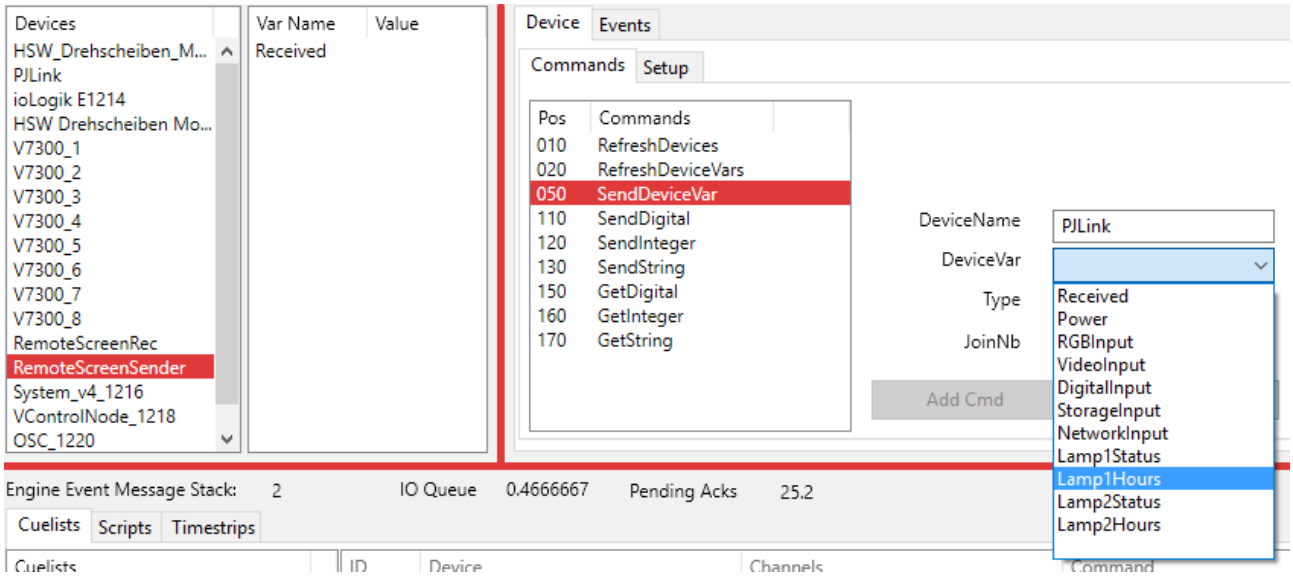
Next we need to get the device variables for a selected device. Select the *RefreshDeviceVars* command an select a device. then switch to *Run* mode and push the *Run Cmd* button.



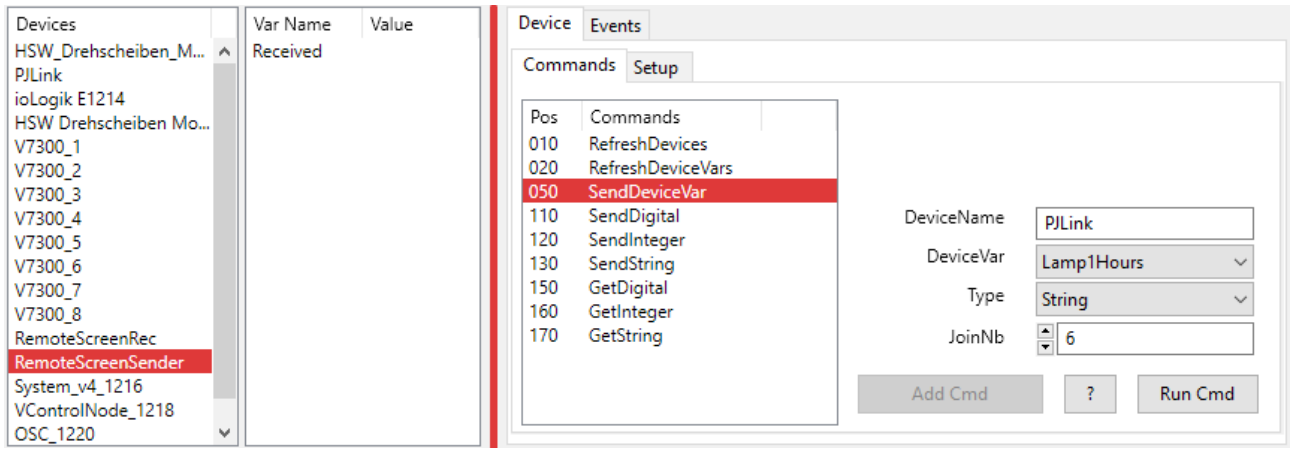
Again nothing seems to happen. But now select the *SendDeviceVar* command.



The *DeviceName* parameter is preselected with the device chossen with the *RefreshDeviceVars* command. Now select a device variable in the *DeviceVar* parameter box.



Next step is to choose to which join number in which format the content of the device variable is send.

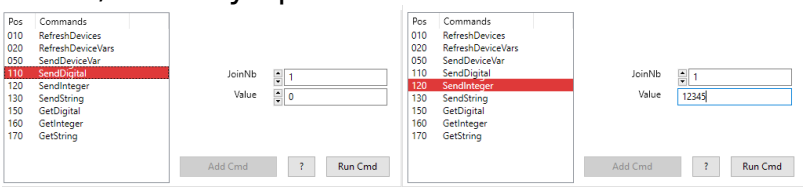


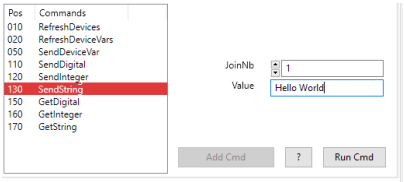
One may wonder why we choose a string join number here. If we want to show the lamp hours of a projector, we usually use a label to show the information. In this case we have a label with the string join number 6 for caption. If we send this as integer, we need a widget with an integer join such as slider or progress bar.

Sending User Data

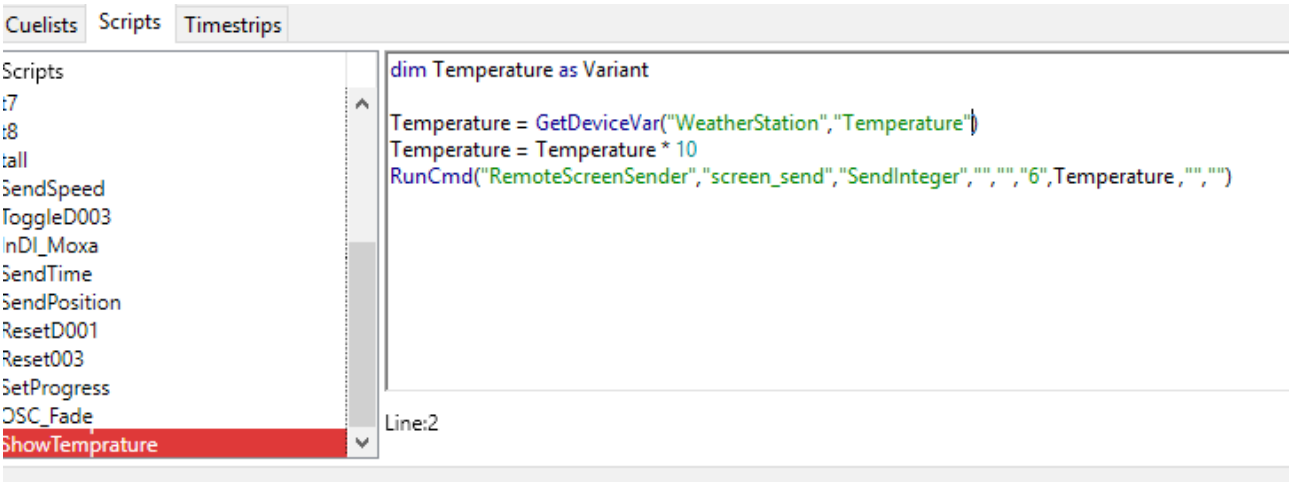
Another way to send messages to a V-Control Designer instance is to use the *SendDigital, SendInteger* and *SendString* commands.

If used in *Cuelists*, the only option is to enter the data direct as command parameter.





If used in *Scrips*, we can process the data that is send.

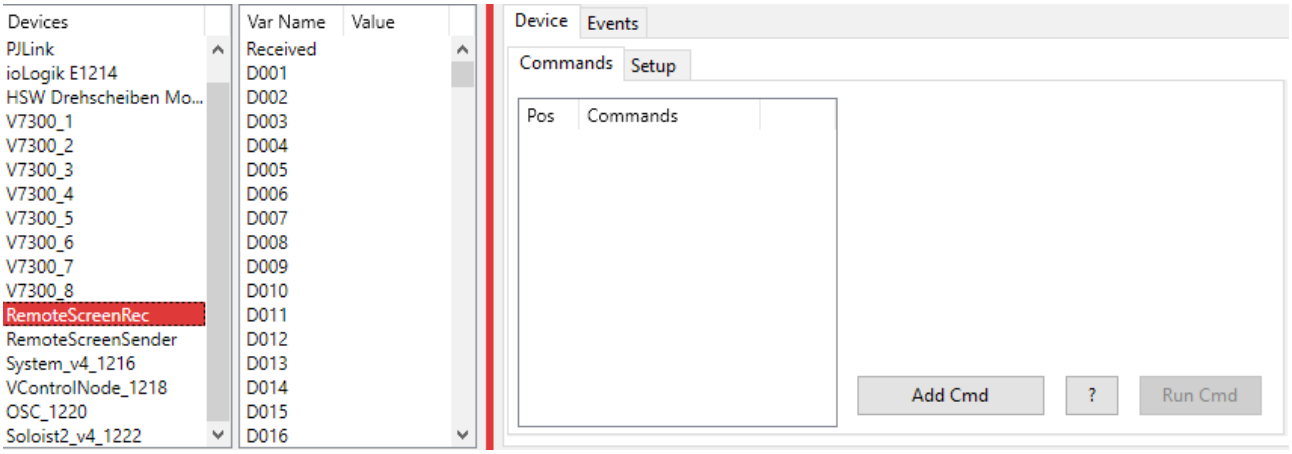


In the example above we want to send the Temperature value of a weather station to a progress bar. But the Temperature comes in a format such as “20,5”. If multiplied by 10, the result is 205, which is an integer value. Imagine we have a progress bar with minimum value -300 and maximum 500, then it can now show Temperatures from -30 to 50°C.

Getting Join Values

Another method to update the user variables of the RemoteScreenReceiver is to send a Get message to V-Control Designer. This is done by the *GetDigital*, *GetInteger* and *GetString* command. If one of these commands is sent from the *RemoteScreenSender*, the Designer send a message to the *RemoteScreenReceiver* that updates the device variable. If the variable changes, an event is fired.

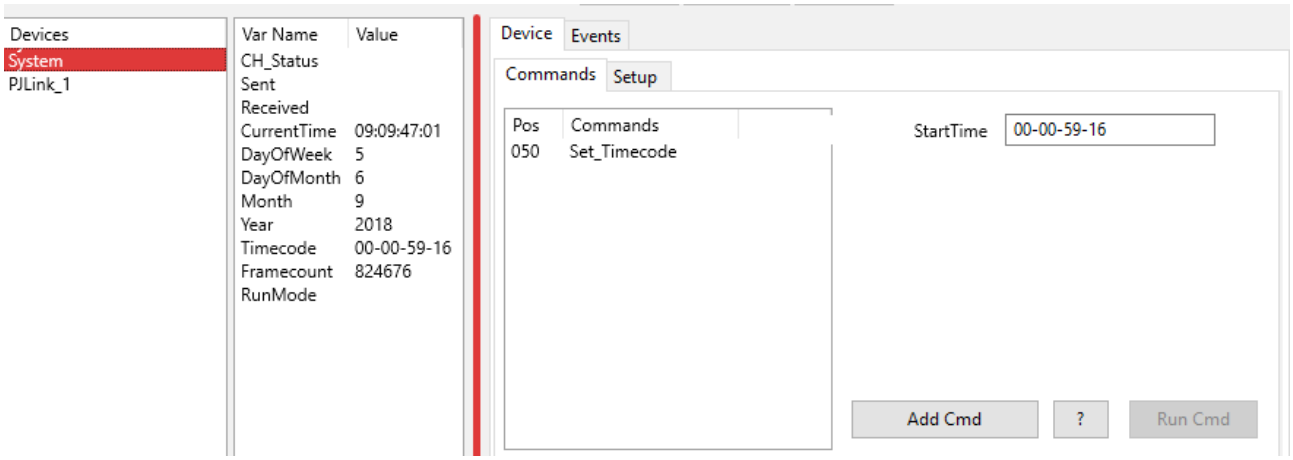
RemoteScreenReceiver



The RemoteScreenReceiver does not have any commands. By default, it has the device variables *D001..D100*, *I001..I100* and *S001..S100*. The RemoteScreenReceiver is only used to generate events on a variable change. How to create events is described in the Events chapter.

System

The *System* device is used to provide Time and Date information.



It has no commands and does not need a channel, because all the communication is internal.

Web Server

One way to create fast responsive web sites is to use the build in web server and WebSockets. This allows us to establish a stable, bidirectional connection between a web application and V-Control. The benefit of a bidirectional connection is, that our web application does not have to poll the V-Control web server to get status changes. As far as a status change of a device occurs, the information is sent to the web application. All we need is to write a handler for this status message and show the new status in our GUI.

In the example below a websocket is initialized

```
<!DOCTYPE html>
<!--
To change this license header, choose License Headers in Project
Properties.To change this template file, choose Tools |
Templatesand open the template in the editor.-->
<html>
    <head>
        <meta content="text/html; charset=UTF-8" http-
equiv="content-type">
        <title>V-Control Web Example</title>
        <meta name="viewport" content="width=device-width,
initial-scale=1.0, user-scalable=no">
        <link rel="stylesheet" type="text/css"
href="./js/libs/jquery-ui-1.10.4.custom.min.css">
        <link rel="stylesheet" type="text/css"
href="./js/libs/jquery-ui-1.10.4.custom.css">
        <link rel="stylesheet" type="text/css" href="main.css">
        <script src="./js/libs/jquery/jquery.js"></script>
        <script src="./js/libs/jquery/jquery-ui-
1.10.4.custom.min.js"></script>
        <script src="./js/libs/v-control.js"></script>
        <script>
            $(document).ready(function() {

//*************************************************************
                //Change the IP with the IP Address of your V-
Control system
                initWebSocket("ws://127.0.0.1:8080"); //use the
same port as on V-Control httpRemote (8080 here)

//*************************************************************
            });
```

For more information please use your favorite search engine and learn more about JavaScript and HTML (if necessary) There is also en example with V-Conntrol 3. This will not work in the same way with V-Control4, but will be a good starting point to learn how to do. https://v-control.com/touch-guis-with-websockets/

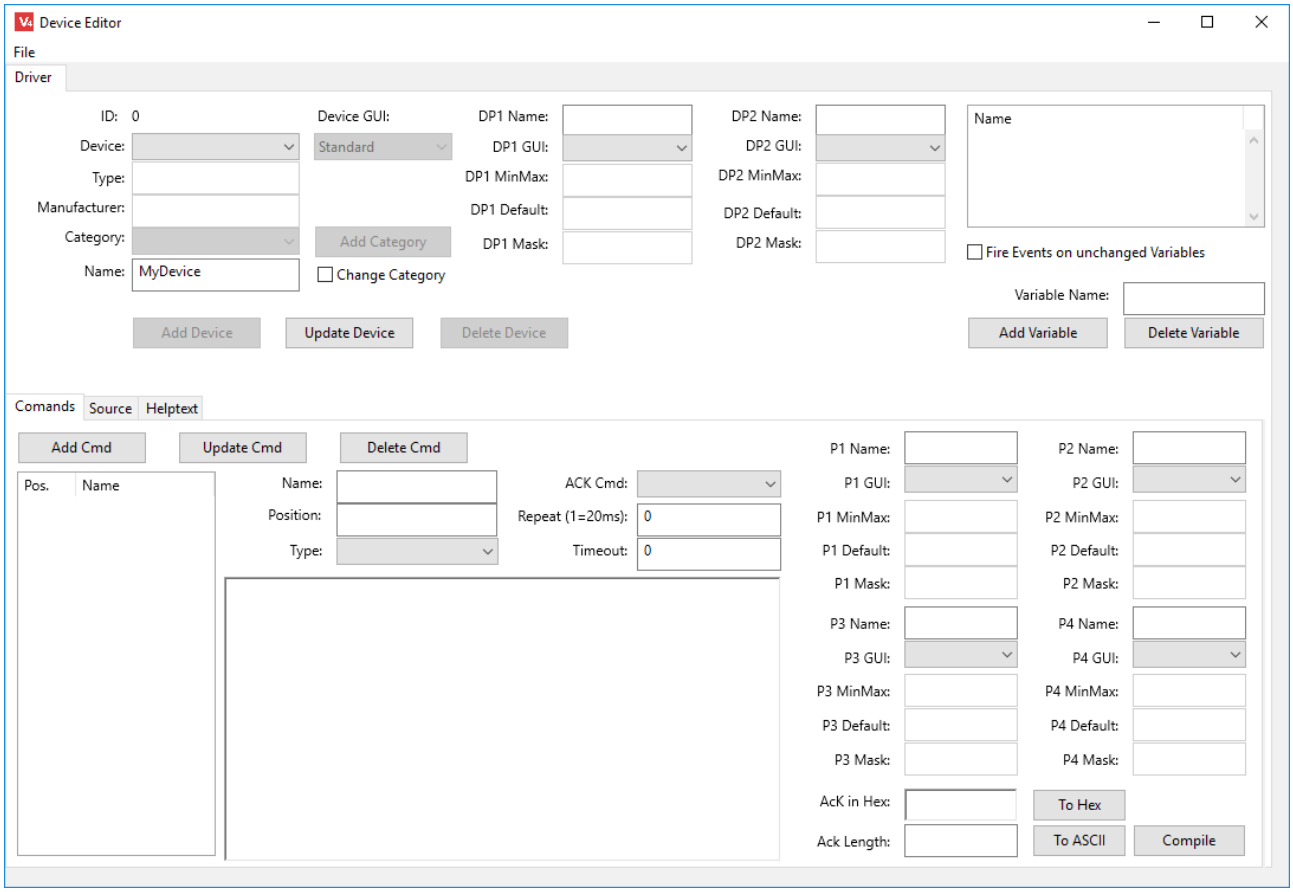
Language / IDE of your Choice

Because V-Control uses an open API, you can use any language / IDE you familiar with and create graphical user interfaces. The API is explained here: Remote Protocol

Device Editor

Device Drivers are very flexible, and for that reason they are also very complex. In fact, creating device drivers is the most complex part in V-Control.

All available device drivers are stored as a template in a database (Devices.rsd). Once they're added to a project, a copy is stored in the project file (*.vc4) too. The Device Editor is used to modify / create these drivers. To modify a driver in an existing project file, the Device Editor is launched via *Configure -> Edit Devices* or the *Device Edit* toolbar button. Now only devices in the current project are available for editing. This means also that all changes to the device driver concern only the current project. The device database with the templates stays untouched. To edit the device templates, the device editor is launched via *Configure -> Edit Device Templates*. In this case, the drivers in the current project stay untouched.



The Device Editor is split into two main sections. The upper part concerns all Device properties, the lower part contains all commands of a device. The image shows the Device Editor for editing devices that are part of the project, not the Device Templates.

Device Properties

In the upper part of the Device Editor the properties of a device are shown. The fields *Type, Manufacturer* and *Category* are used to find devices in the

database. *Device GUI* is not active at the moment and used for later extensions. *Add Category* is available only if the Device Editor was opened via *Configure → Edit Device Templates*. In this case a popup window appears asking for a new category name.

Device Variables

Device Variables are used to store the status of a device. The driver is in charge to set the device variables to a value that represents the current status of the device. As example lets look at a projector. A procetor has some inputs, it can be switched on or off and most of them have a lamp timer. The driver collects information about the device and fill this information in the device variables.

A projector could have the variables PowerState, SelectedInput and LampHours. The driver request now information from the projector that tells if the projector is on or off. This information is stored in the PowerState variable. The driver also asks the projector which is the current selected input, and stores this information in the SelectedInput variable etc.

The number of device variables is not limited. It depends on what the designer of the device driver thought was useful to have.

Each time a Device Variable changed, an event is fired. This event can be used by V-Control as a trigger to start tasks.

By default, the event is fired only if the variable changes, and not if the variable is updated but does not change. What does this mean in detail? Lets do an example:

Assume that the driver asks a projector every second if for the power state. If the projector is switched off, the drivcer wil set the PowerState variable to “off”. Now the projector switches on, and the next time the dreiver requests the power information “on” is returned and the content of the PowerState variable changes from “off” to “on”. An event is fired, and if we have programmed an event handler in V-Control, the content of the variable is checked. If the variable content matches the condition defined in the event handler, a task is fired.

One second later the driver asks again for the power state. Because the projector is still on, a “on” value is returned. By default, no event is fired in this case bacause the PowerState variable does not change. However, there might be some rare situations where you want to fire the event each time a new variable value is requested from the device, no matter if it has changed or not. In this case, check the *Fire Events on unchanged Variables* box.

Parameter

Many commands sent to a device need parameter(s) to make sense. E.g. a CueUp command for a Player need a Timecode to tell the player where to go. An example for a parameter less command is the Play command. This instruction contains all the information that is needed for the driver, simply turn the device in play mode. As these examples show, it is necessary to refer

parameters to a command.

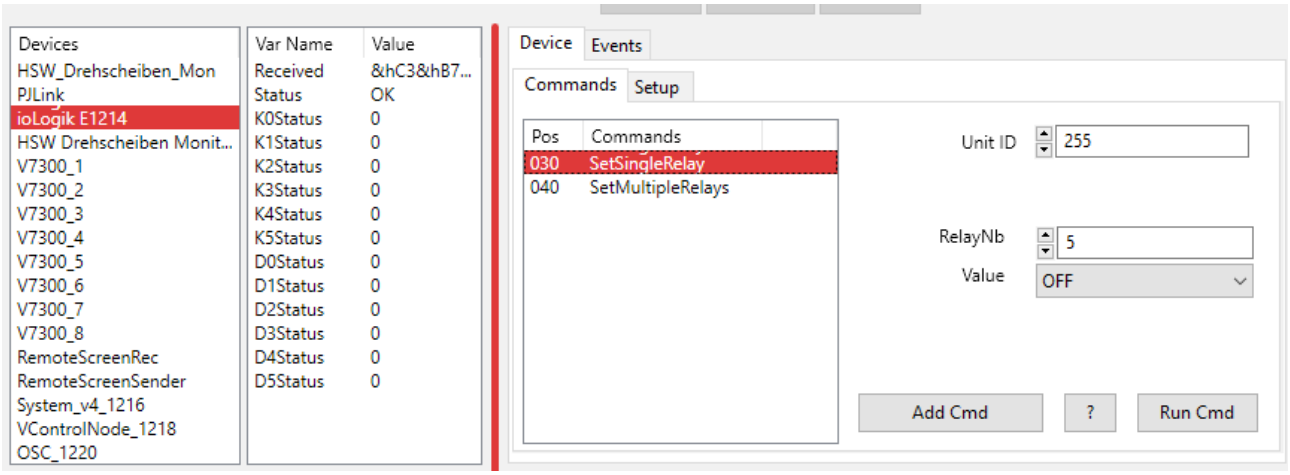
Device Parameter

In V-Control, up to six parameters for each command are available. We distinguish between device parameters (DP1 and DP2) and command parameters (P1 - P4). Both parameter types are equivalent and used in the same way.

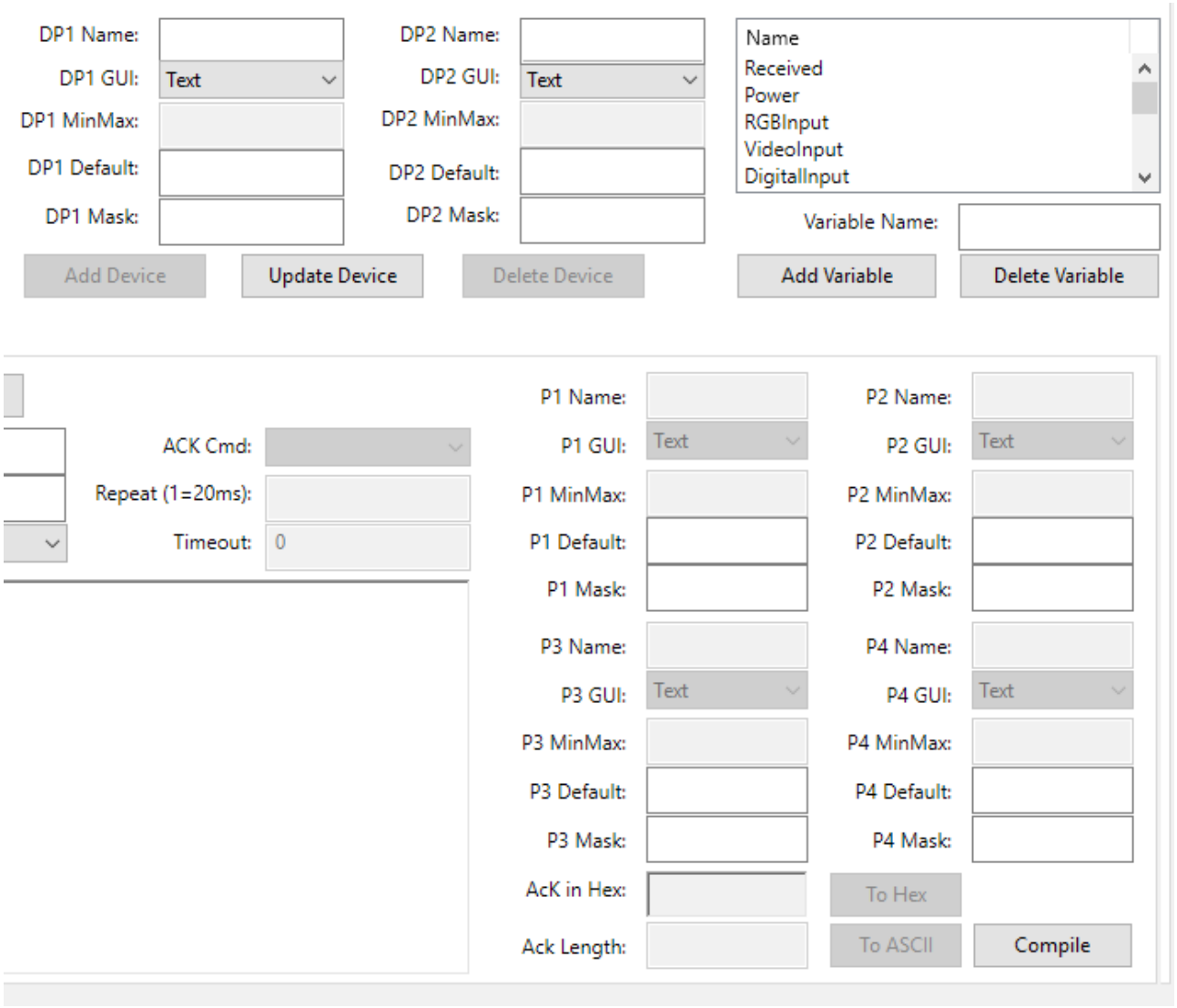
Device parameters affect more the device itself and less the individual functions. They are visible for each command, and all commands share the same Device Parameters.

One example for a device parameter is the Modbus TCP protocol. Some devices require their own Unit ID, even if they are only accessible under a specific IP address. The Unit ID adresses the device and needs to be involved in every command. So we can use a Device Parameter for that Unit ID.

Command Properties



To enter a parameter, an parameter input field is needed. This could be a text field, a spin button or a drop down box. In the example above a spin button is used to enter the Unit ID (Unit ID is a Device Parameter in this case) and the RelayNb. A drop down box determines whether the relay should be switched on or off. The parameter input field is defined by five controls in the device editor. These are the same for Device Parameter and Command Parameter.



1. Name: The name of the parameter. This is just a label that tells the user what kind of parameter is needed.

2. GUI: The type of the input field. Three types are available.

- Spin Button: A field for numeric values
- Drop Down Box: A list of predefined values
- Textfeld: An input field for all kind of parameters

3. MinMax:

- If the GUI type is spin button, then this field contains the minimum and maximum value, separated by a semicolon. If the youser should enter a value between 1 and 19, then 1;19 is entered here.
- If the GUI type is drop down box, then this field contain a semicolon separated list of all possible values. E.G. if the projector has a RGB, a Video and a S-Video input, the content of this field could be “RGB;VIDEO;S-VIDEO”.
- If the GUI type is text field, then this field is inactive.

4. Default:

- If the GUI type is spin button, then this field optionally contains the default numeric value that is used for the parameter. The default value must be in the range defined with the MinMax value.
- If the GUI type is drop down box, this field contains the list item selected by default. The value MUST be one of the semicolon separated values defined in the MinMax field.
- If the GUI type is text field, then this field contains the default value for the text field.

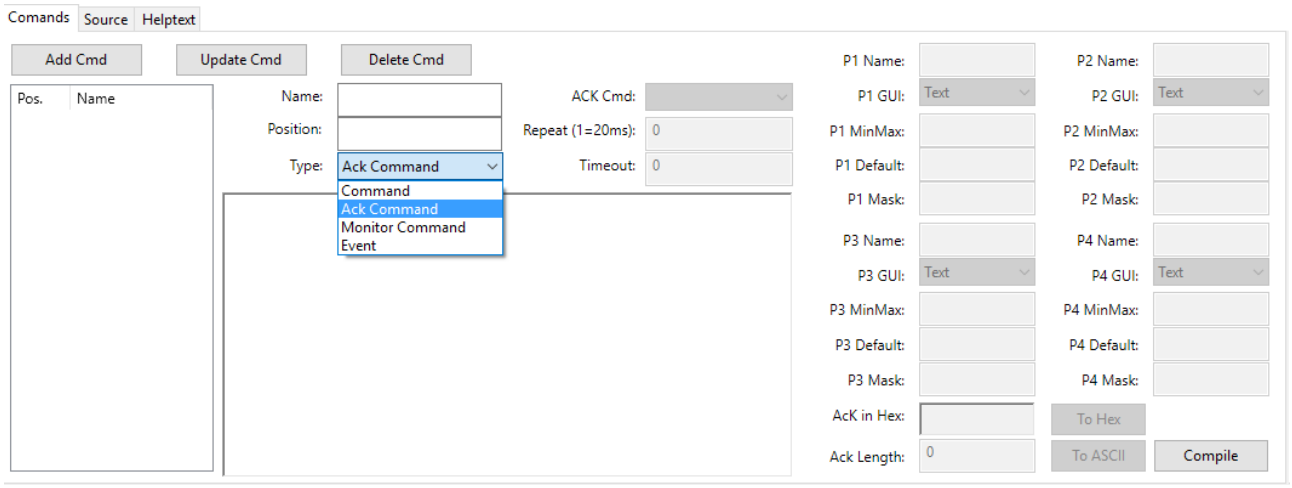
5. Mask:

- Inactive if GUI type is spin button
- Inactive if GUI type is drop down box
- If the GUI type is text field, an optional mask for the field is defined here.

Mask Character	Description
#	Any single digit placeholder. The user can type only a digit character in this position.
.	Decimal placeholder. The decimal placeholder that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes.
,	Thousands separator. The thousands separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes.
:	Time separator. The time separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes.
/	Date separator. The date separator that is actually used is specified in the user's International settings. The character is treated as a literal (formatting) character for masking purposes.
\	Mask escape character. Treat the next character in the mask as a literal. The

	escape character enables you to use the '#', '&', 'A', '?' (an so on) characters in the mask. The escapted character is treated as a literal (formatting) character.
&	Character placeholder. Valid values are the ASCII characters 32-126 and the non-ASCII characters 128-255.
>	Convert all the characters that follow to uppercase. Uppercasing works beyond the ASCII range where appropriate, e.g., ü becomes Ü.
<	Convert all the characters that follow to lowercase. Lowercasing works beyond the ASCII range where appropriate, e.g., Ü becomes ü.
A	Alphanumeric character placeholder, where entry is mandatory. For example, the spec “AAA” specifies three alphabetic characters.
a	Alphanumeric character placeholder, where entry is optional.
9	Digit placeholder where entry is optional.
C	Character or space placeholder, where entry is optional. It operates like the '&' placeholder.
?	Alphabetic placeholder.
Any literal	All other symbols are displayed as literals for formatting purposes.
~	Reserved for future use. If you use “~” it will trigger an exception error. Use \ ~ instead.

Command Types



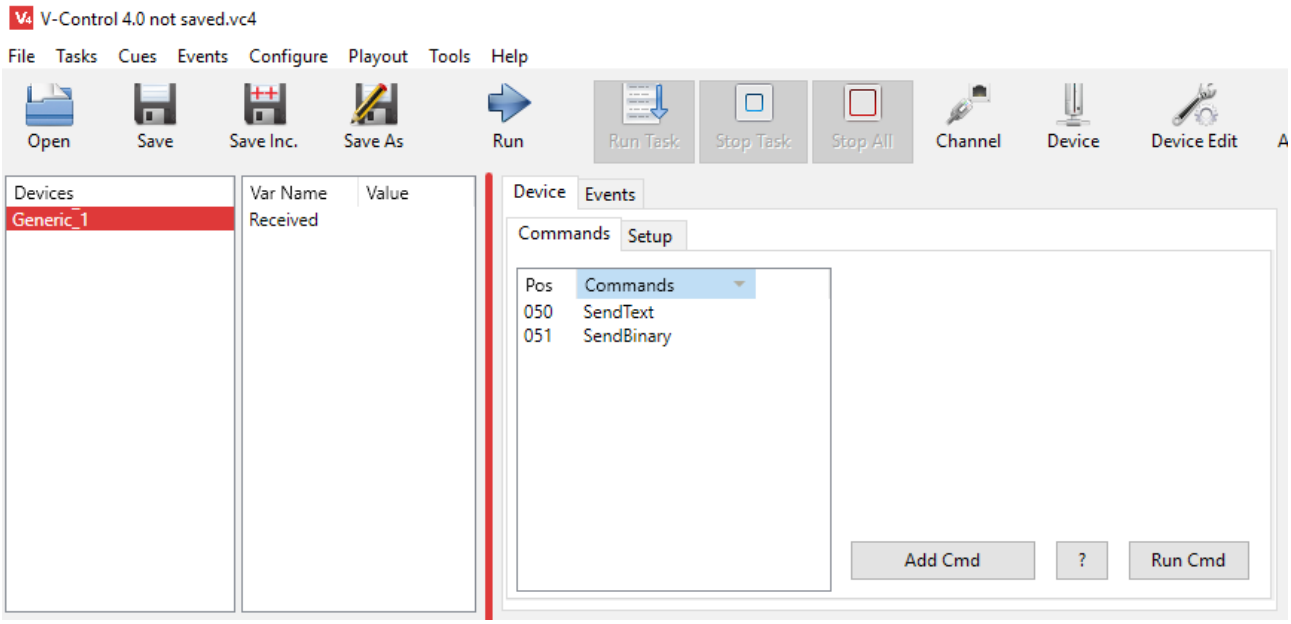
A device driver as 4 different types of commands:

1. **Command**: This is a command that appears in the command list. These are the commands that the user uses to control the device.
2. **Ack Command**: An Ack(nowledge) command is a command that is executed after a command or Monitor Command was send. These commands are used to process the data returned from the device.
3. **Monitor Command**: This is a command that is executed periodically. It is used to request the status of a device. An Ack Command is used to process the result and fill device variables with information.
4. **Event**: To be exact, an event is not a command. An event occurs if a device send information to a driver without being requested before. This Data is handled here in a very similar way then in Ack Commands.

Create New Driver

In the following example, we create a new driver for the Adtec SOLOIST 2 MPEG player. I use this old fashioned device because we see all aspects of device drivers here. To start this job, it is necessary to have the protocol provided by the manufacturer and study it. The protocol describes the RS232 settings (baudrate, start bits, stop bits, parity) and the control cable pin assignment as well as the commands the device understand. The default RS232 parameters are 38400 baud, 8 data bits, 1 stop bit and no parity, so we need to create a channel with these parameters. In this example we call it “soloist”.

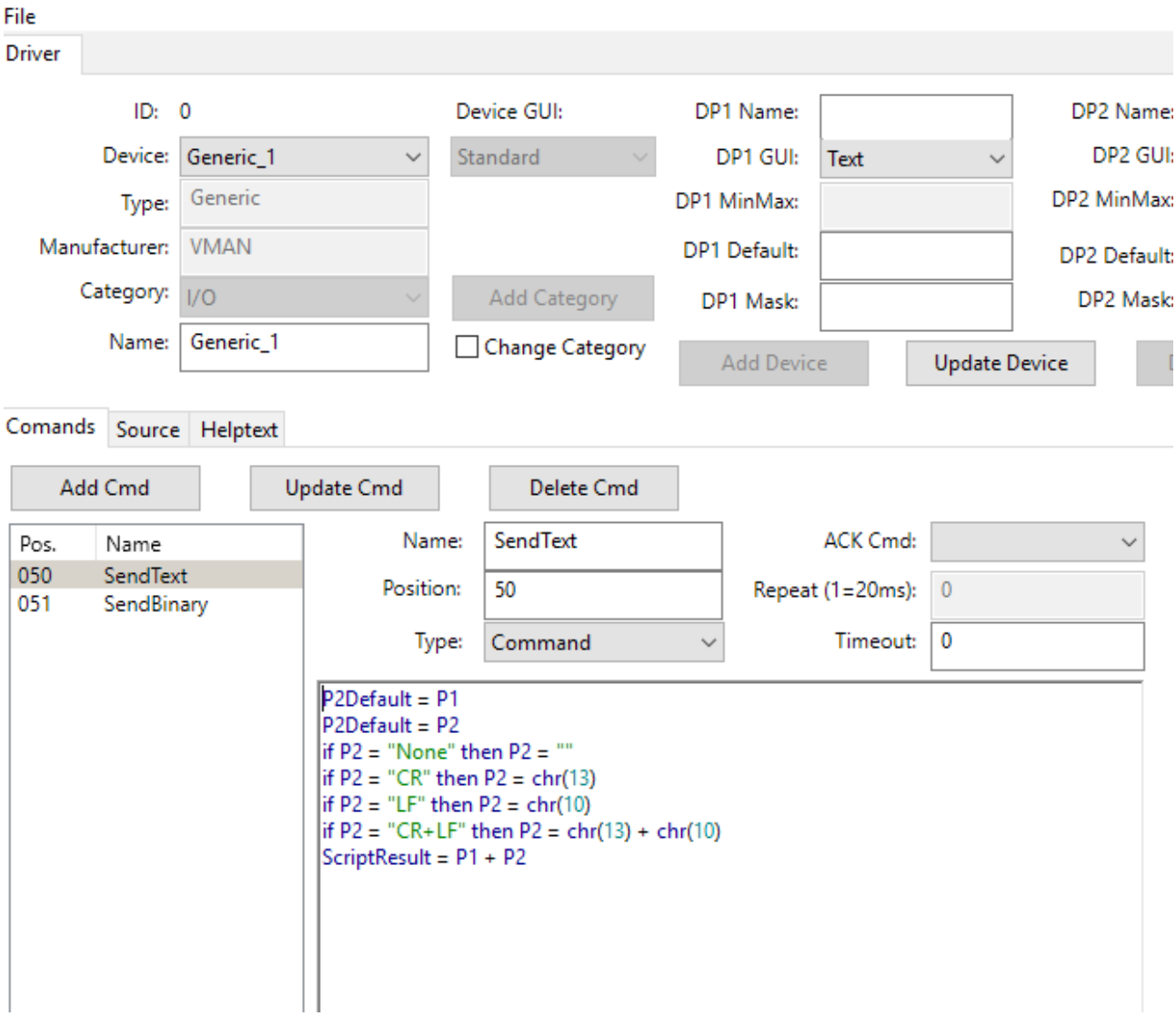
There is no device using a protocol that is similar to Adtec's SOLOIST 2, so we load the Generic Device from the I/O category (see Device Setup) and assign the channel “soloist” to this device. Then the project is saved as e.g. soloist_test.vc4.



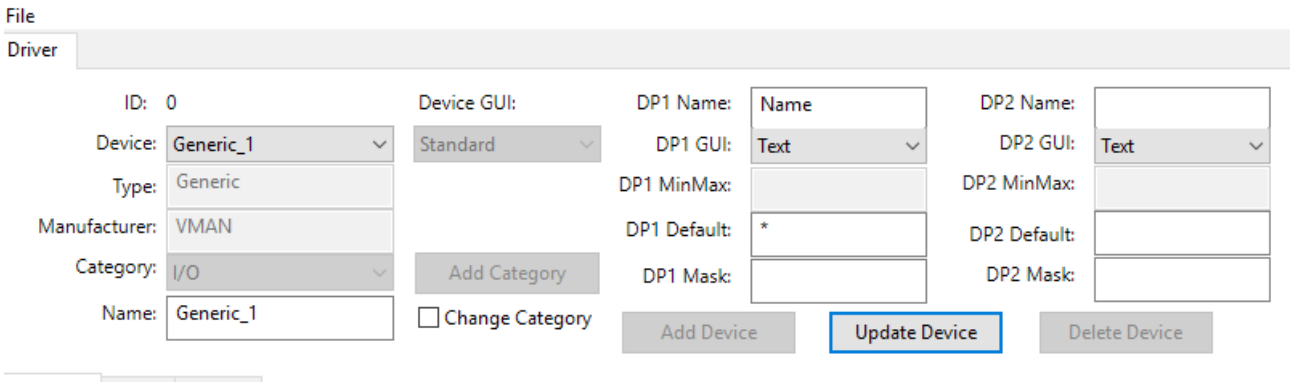
Simple Commands

To start we need to launch the device editor via main menu *Configure -> Edit Devices* and select the device *GenericDevice_1*.

The two commands OnData and SendString are not used and can be deleted. To do that click on the command and then click on *Delete Cmd* (see screenshot)

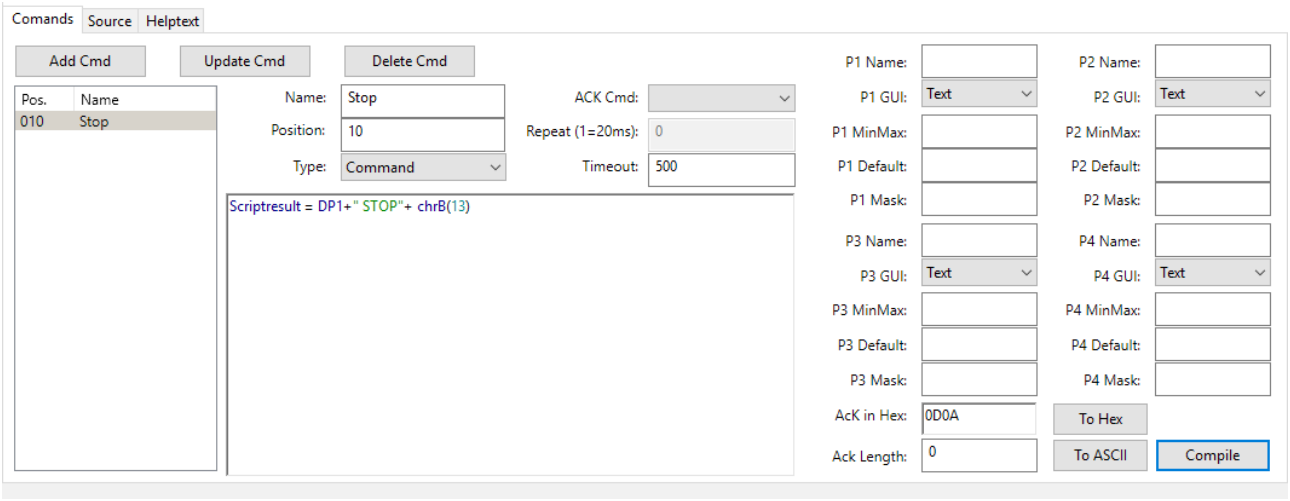


Each command for the SOLOIST 2 player need at least one parameter - the name of the device. The “*” character is used as a universal name and every SOLOIST accept this sign as his name. Because the SOLOIST name concern the device and not the command it is implemented as device parameter. In the field *DP1 Name* enter “Name” as identifier and “*” in the field *DP1 Default*. To add this parameter to the device click *Update Device*.



The first command we implement is the STOP command. The command string looks like this: “* STOP” + Chr(13). As feedback (Acknowledge) the SOLOIST 2

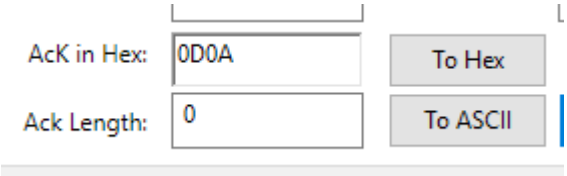
send “OK”+ CR+LF+CR+LF. CR means Carriage Return and LF Line Feed. CR is the ASCII Coder 13, or 0D in hexadecimal. LF is 10 or 0A in hexadecimal.



To implement the command follow the instructions in the screenshot.

- Command Name: Because the command is a Stop command the name is obvious. In V-Control command names are used as function calls. Therefore names like GoTo, Exit or Sub are illegal, because they are Basic functions respectively reserved keywords. If one of those reserved words is used, the compiler generates a compiler error while compiling the script.
- The number in the *Position* field serves as sort criterion in the command list.
- The Type determines the type of command (see **Command Types**)
- The source code of the command. The variable ScriptResult has a special job. It contains the command string, that is sent via channel to the device. This means that the variable ScriptResult must contain a string that causes a Stop command for the Adtec player. The line `ScriptResult = DP1 + “ STOP” + ChrB(13)` assigns parameter DP1 to ScriptResult. DP1 is the first of two available device parameter and contains in this case the name of the SOLOIST. Totally up to six parameter (DP1,DP2,P1,P2,P3,P4) are available, but this time we use only DP1. Then a space character followed by STOP (“ STOP”) is added. The ChrB function converts an Integer value to ASCII code. The SOLOIST 2 expect a Carriage Return (ASCII code 13) as terminating character which is added by the ChrB function. Now the command string is complete.
- The command timeout in milliseconds. If there is no response (Acknowledge) within this time, a Timeout message is created.

The Timeout leads to the question, how V-Control knows if, respectively when, an Acknowledge is present. That tells the field Ack in Hex: Here the expected response is entered. The screenshot below shows that the 0D0A is expected.



As Ack String, only Hex code is accepted. So you need to convert everything to Hex. If you enter text in the *Ack in Hex* field and press the button *To Hex*, the text converts ASCII String to hex code.

But the SOLOIST 2 not only send 0D0A as Acknowledge. The complete string is OK0D0A0D0A. Well, we don't care about that. If we received 0A0D then we know that the message was received and processed by the Soloist.

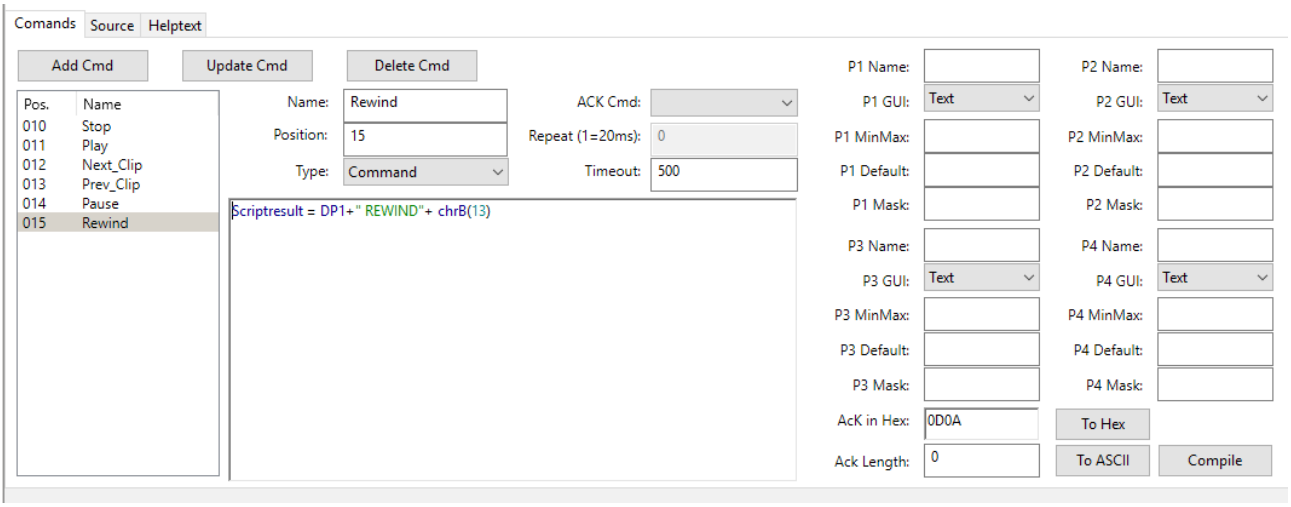
The first command is complete. A click on the button *Add Cmd* saves the command. Because V-Control compiles the script, we have to click the *Compile* button. If there is an error, the compiler posts a compiler error, otherwise the device is usable now. To do that, close the Device Editor and select the Device GenericDevice_1 again (see screenshot below). The Stop command now appears in the command list. The device parameter Name has the “*” character as default value. Alternatively we can use the real SOLOIST 2 name, e.g. “SOLOIST”.

If the command is working it is recommended to save the project. Until now, the driver exists only in the project file (“soloist_test.vc4” in this example), and this file contains all the work we did. If the driver is complete, we will export it and then import it to the device database. To complete the driver we start the Device Editor via main menu Configure -> Edit Devices again. The table below shows the remaining simple commands for the SOLOIST 2 player. Proceed in the same way as we did with the Stop command.

Name	Position	Type	Timeout	Script	Ack in Hex
Play	11	Comands	500	Scriptresult = DP1 + “ PLAY” + chrB(13)	0D0A
Next_Clip	20	Comands	500	Scriptresult = DP1 + “ NEXT” + chrB(13)	0D0A
Prev_Clip	21	Comands	500	Scriptresult = DP1 + “ PREVIOUS” + chrB(13)	0D0A
Still	22	Comands	500	Scriptresult = DP1 + “ PAUSE” + chrB(13)	0D0A
Rewind	23	Comands	500	Scriptresult = DP1 + “ REWIND” +	0D0A

				chrB(13)	

The result should look like this:



Finally click the *Compile* button and test the driver.

Now the general conditions to control the device are created. As pointed out in the SOLOIST 2 protocol, the general command format for the device is:

<NAME><SEP><COMMAND><SEP>[ARGUMENT(S)][SEP]<CR>

Parameters in angle brackets <> are mandatory Parameters in square bracket [] are optional

- NAME: SOLOIST Name, up to 20 characters. The SOLOIST 2 is capable to daisy chain the RS232 signal to other SOLOIST players. Thus it is necessary to have a unique ID for each device which is represented by the name. The “*” symbol addresses all player, whatever the name of the device is.
- SEP: Separator character, SPACE (“ ”), comma(“,”) or semicolon (“;”)
- COMMAND: ASCII string for the command, e.g. “PLAY”, “STOP” or “PLAYSPOT”
- ARGUMENTS: some commands need arguments (parameter) to make sense. E.g. the PLAYSPOT command need the clip that has to be played as parameter.
- CR: Carriage Return (Enter Key, ASCII Code 13)

Respecting the above instructions, the command string for a simple PLAY command sent to a device named “SOLOIST” looks like this:

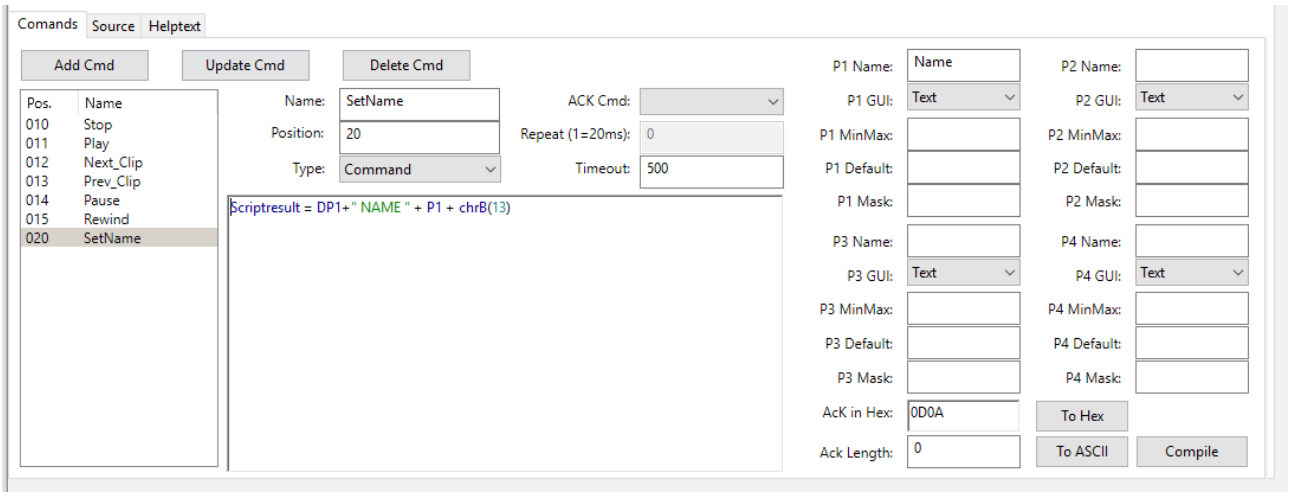
```
“S0L0IST PLAY” + chr(13)
```

Complex Commands

In this chapter the SOLOIST 2 driver is enhanced with a couple of more complex commands. These commands need one or more parameters. In the chapter before, we used the Device Parameter 1 (DP1) to address the SOLOIST (DP1 contain the name of the SOLOIST). The following four commands use more then one parameter. Additionally the three types of input fields (Text

Field, Spin Button and Drop Down Box) are introduced.

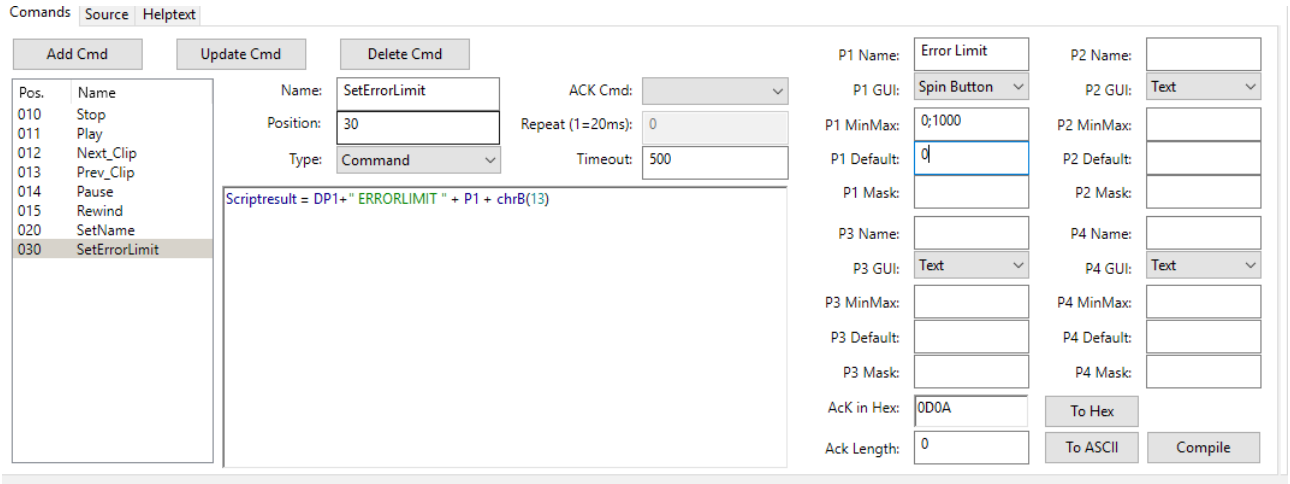
The first new command is the “SetName” command. With this command we can change the SOLOIST's ID. Start the Device Editor again (menu *Configure -> Edit Device...*) and select the device GenericDevice_1.



The creation of this command is very similar to the commands described in the chapter before. The difference is that we use one more parameter (P1). The field *P1Name* shows the parameters name. Because the command shall change the name we call the parameter “SetName”. *P1 GUI* stays at Text, because it makes sense to use a text field to enter a new name. In difference to DP1 we don't use a default value.

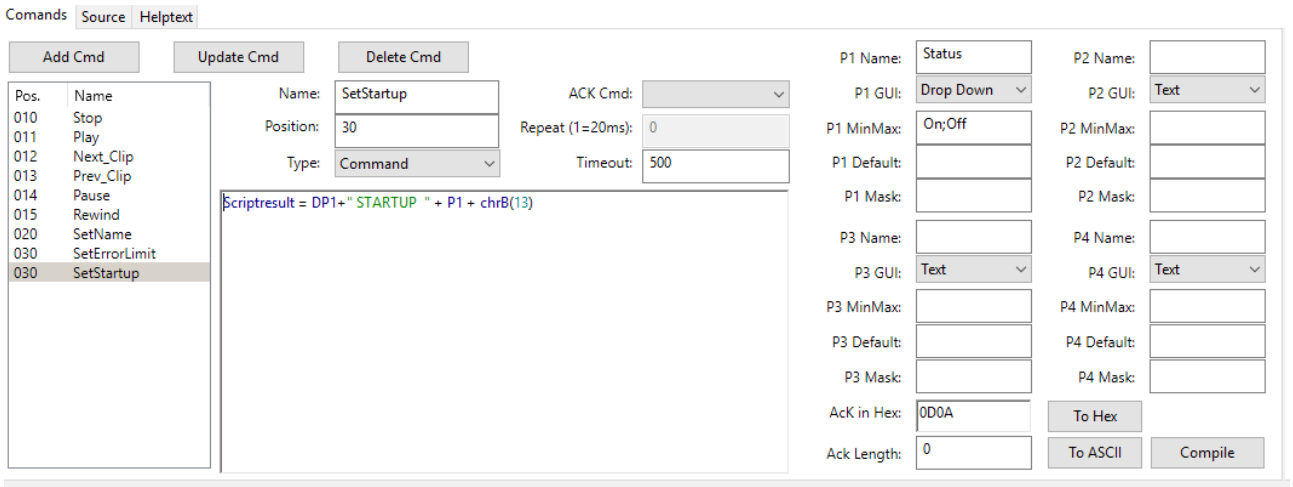
Crucially again is the line `Scriptresult = DP1+“ NAME ” + P1 + chrB(13)`. As before, DP1 contains the current ID (name) of the SOLOIST player. Then the command string “ NAME ”, with leading and succeeding space character is added. P1 contains the new name for the SOLOIST and ChrB(13) is for the carriage return (ASCII code 13) termination character. To save the new command click *Add Cmd* or *Update Cmd* (if the command was edited) followed by *Compile*.

The next command is SetErrorLimit, which determines how often the player try to read from hard disk, before playing interrupts. Valid parameter is a number between 0 and 1000. To enter this parameter we want to use a Spin Button. The Device Editor now looks like this:



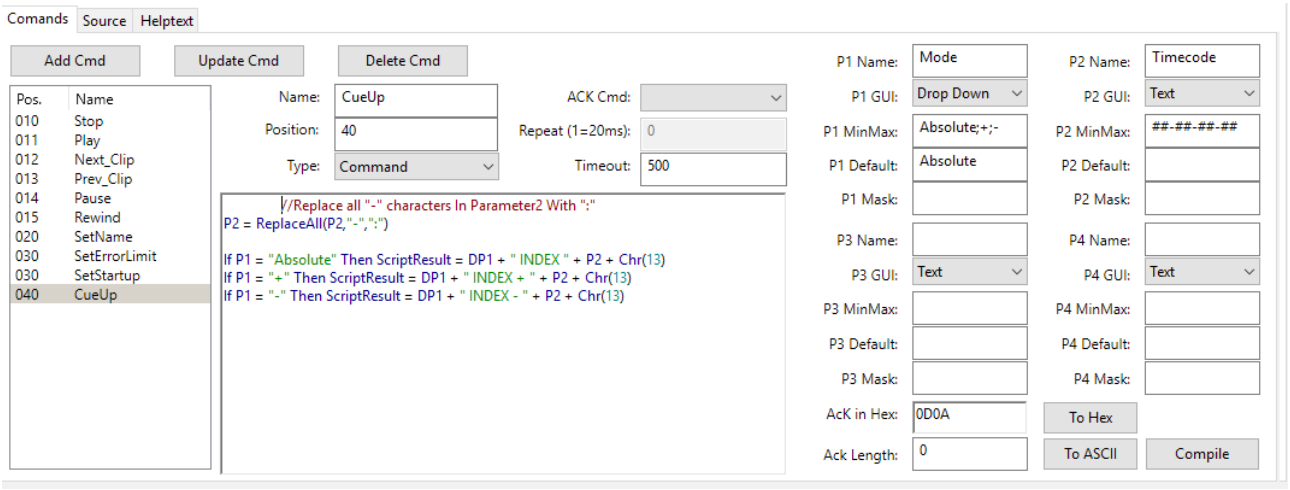
P1 Name contains the designation of the parameter as usual (“Error Limit”), but *P1 GUI* is set to Spin Button. *P1 MinMax* contain, separated by semicolon, the minimum and maximum numeric value that P1 might have. As default value 0 is selected, because usually there should be no reading error. Comparing to the previous command, the script is the same except that we replace “ NAME ” by “ ERRORLIMIT ”. Don't forget to click *Add Cmd* or *Update Cmd* followed by Compile.

The StartUp command determines how the SOLOIST player behaves on Power On. If StartUp = On, the player starts playing after power on, if StartUp = Off the player do nothing. There are only two possible values as parameter, “On” and “Off”, so we use a Drop Down Box as input field.



P1 Name contains the designation of the parameter as usual (“Startup”), but *P1 GUI* is set to Drop Down. *P1 MinMax* contain, separated by semicolon, the possible values, that P1 might have. Comparing to the previous command, the script is the same except that we replace “ ERRORLIMIT ” by “ STARTUP ”. Don't forget to click Add Cmd or Update Cmd followed by Compile.

The last command is the CueUp command. This command lets the SOLOIST jump to a position in the currently loaded clip. The position is entered as timecode (HH-MM-SS-FF) which addresses an absolute position in the clip. But the SOLOIST also knows relative positions. E.g. “+ 00-01-02-00” jumps forward one minute and two seconds from the current position. “- 00-01-02-00” jumps one minute and two seconds back. So we need three parameters (Soloist Name, absolute or relative position and the position as time code.



The parameter *P1* determines the mode (absolute, relative +, relative -), and is implemented as Drop Down Box. The default value is “Absolute”, because this is the most often wanted mode. *P2* is the parameter for the time code and implemented as text field. *P2 Mask* set the input mask for this parameter. In this case, only numeric values are accepted, separated by “-” character. We can not use the “:” as separator in a mask, because it is used for time formats (HH:MM:SS) and not for Time code (HH-MM-FF-SS).

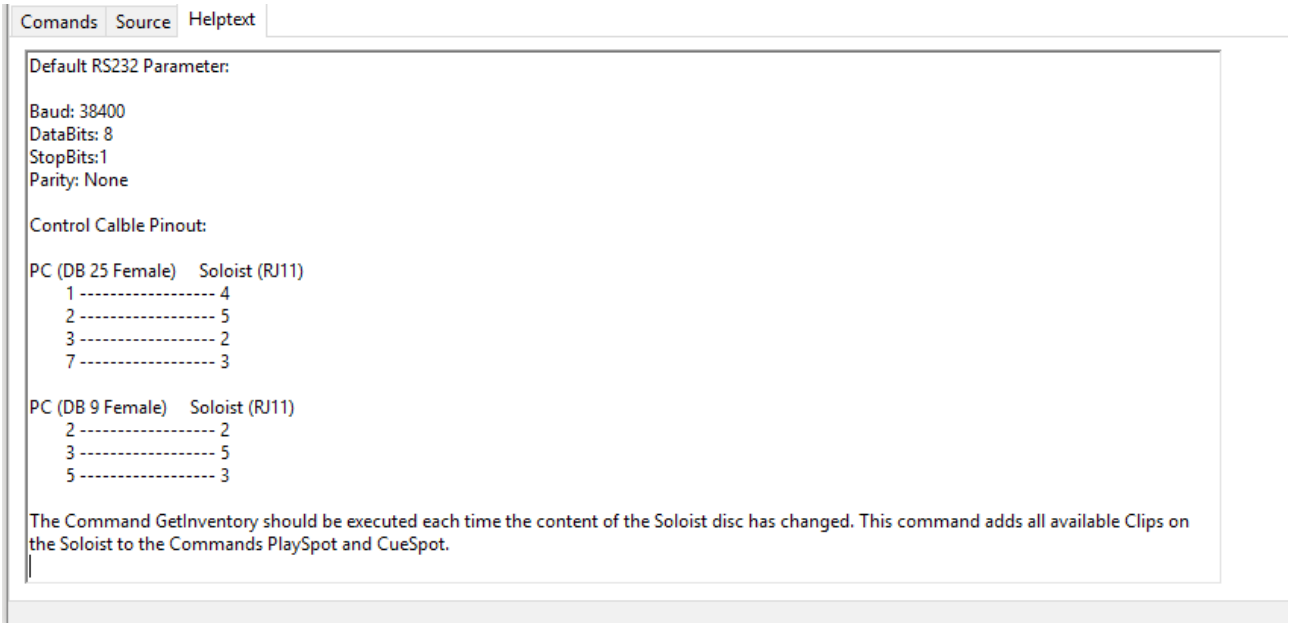
This time, the script is a little more complex:

```
//Replace all "-" characters In Parameter2 With ":"
P2 = ReplaceAll(P2,"-",":")

If P1 = "Absolute" Then ScriptResult = DP1 + " INDEX " + P2 +
Chr(13)
If P1 = "+" Then ScriptResult = DP1 + " INDEX + " + P2 + Chr(13)
If P1 = "-" Then ScriptResult = DP1 + " INDEX - " + P2 + Chr(13)
```

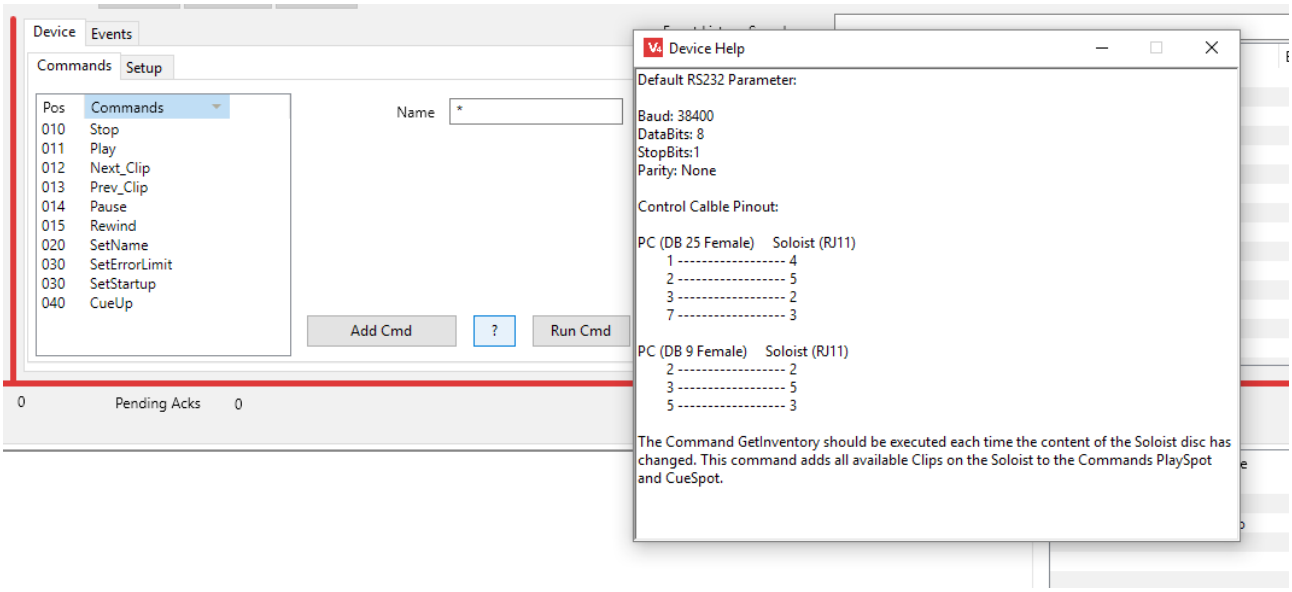
The SOLOIST expects the time code in the format “HH:MM:SS:FF”, with “:” as separator and not “-”. P2 uses “-” as separator, so we have to replace any “-” sign by a “:”. This is what `P2 = ReplaceAll(P2,“-”,“:”)` does. Then we compare the value of P1 with “Absolute”. It true (P1 = “Absolute”) then no “+” or “-” sign is leading the time code and for the SOLOIST this is an absolute value. Don't forget to click *Add Cmd* or *Update Cmd* followed by Compile.

Hep Text

Every device driver can have a help text. The text is for the user to give him some hints or necessary information to use the driver and / or the device. It is often used to describe the RS232 properties like baud rate, data bits, parity etc. or the pin assignment for the control cable.



If the help text is complete, the button Update Device saves the text in the database. In the Device GUI it is now retrievable via Help button *?*.



Monitor and Acknowledge Comands

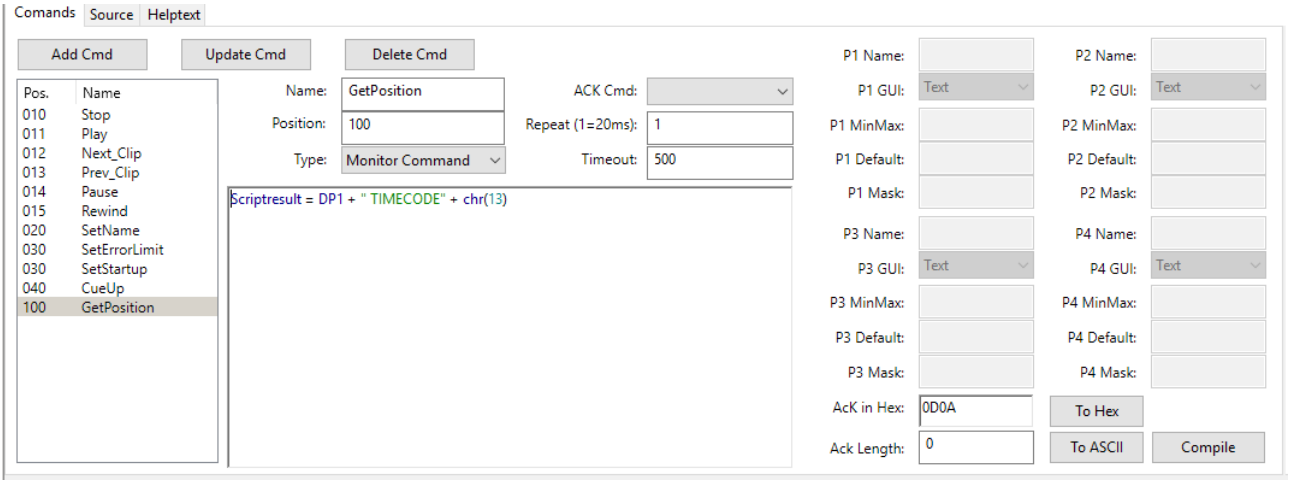
In all our examples we used the ACK string (ACK in Hex:). If the attached device send this string as response, we know that the command was understood and executed (usually, some devices send two acknowledges, one if the command was valid and one when execution is finished). If the expected string is not received in the time determined by Timeout, a Timeout error occurs. But there are acknowledges that need to be processed further.

For a player, V-Control should always know the playback position. The Soloist has a command “TIMECODE”, which returns the current position as timecode. We want to have this useful information in a device variable. We need a combination of Monitor command and Ack command to do this.

Monitor Commands

A monitor command is a command that is executed periodically. It is used to request the status of a device. This can be a projectors lamp hours or power status, or in the following example the timcode position of a player.

To create a *Monitor Command* select it in the commands type field (drop down box).



The fields *Timeout* and *Ack in Hex* are used in the same way then before. Two things are new:

1. All command parameters are disabled. This is because this command is started by a timer. There is no user interaction when this command executes, so there is no chance to modify any parameters. But we can use the device parameters default values. In this case this is necessary, because we need the Soloist name for the command string.
2. A new field *Repeat* is enabled. Here we determine how often this command is executed. Values entered here can be between 1 and 3000. To determine how often this command executes multiply this value with 20ms. So 1 means this command is executed every 20ms (Milliseconds), 5 means every 100 ms.

It is important to know that there is only one timer per device. If you have two Monitor Commands, both using 1 as Repeat value, they are not fired every 20ms. In this case they are fired every 40ms (first 20ms slot command 1, second 20 ms slot command 2).

If you want to monitor i.e. a projectors lamp hours, input channel and power state, it does not make sense to fire these request very fast. Usually approx. 1 second is enough. One Second means a Repeat value of 50.

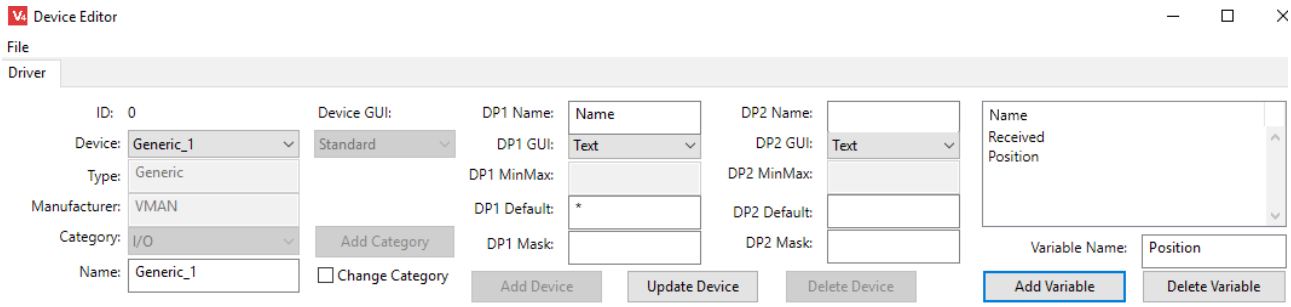
For the timecode position it makes sense to request it very fast, so we use 1 as repeat value here.

The command is complete, whats missing is processing the result and store it in a device variable. This is done by an *Ack Command*. The Ack Command is selected in the *Ack Cmd* drop down box. Because we do not have the command right now we need to create it and assign the Ack command to the GetPosition

command later.

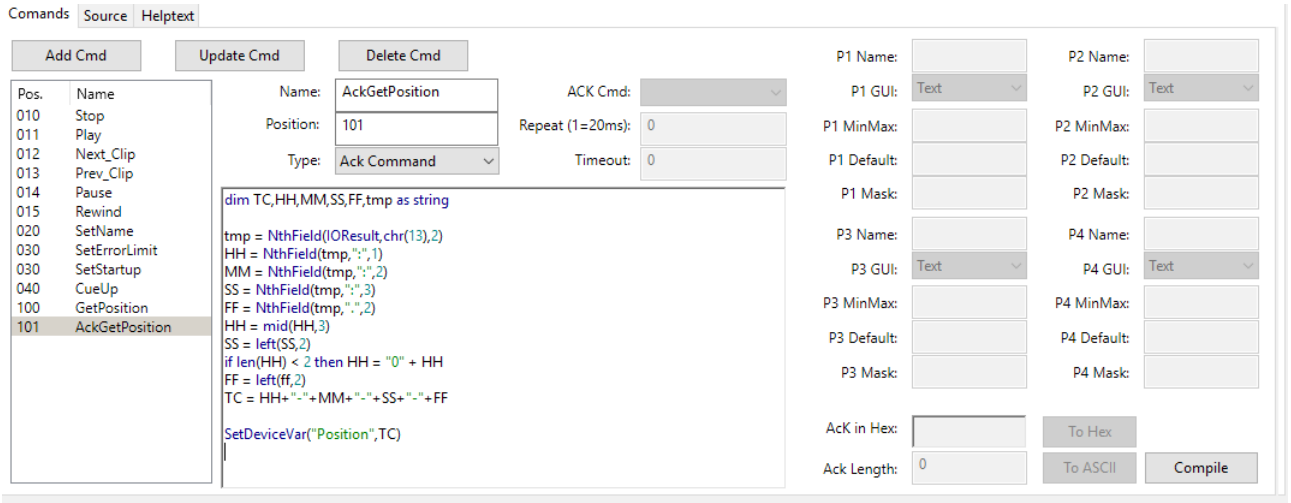
Ack(nowledge) Commands

The result of the GetPosition command has do be stored in a device variable. So the first step is to create one.



Enter the variable name in the *Variable Name* field and click the *Add Variable* button.

Next, create a new command as shown below.



Here we have a quite complex script to fetch the timecode from the Soloist returning message. This is because the soloist returns the timecode in the format `OK<CR><0A> H:MM:SS.FF<CR><0A>`, but we need `HH-MM-SS-FF`

OK, lets go through the lines:

Here wie declare variables we need, all of type string in this case.

```
dim TC,HH,MM,SS,FF,tmp as string
```

The variable IOResult is permanently available and does not have to be declared. The Answer from the Soloist is stored here. If you don't know how `NthField` works please read the NthField documentation in the Compiler section.

The content of IOResult is `OK<CR><0A> H:MM:SS.FF<CR><0A>`, so we fetch the right part of the first `<CR>`

```
tmp = NthField(IOResult,chr(13),2)
```

now tmp has the content `<0A> H:MM:SS.FF`. Next we grab the Hours

```
HH = NthField(tmp,":",1)
```

HH is now <0A> H

then we grab Minutes and Seconds

```
MM = NthField(tmp,":",2)
SS = NthField(tmp,":",3)
```

Because the Frame delimiter is ".", we grab the frames by

```
FF = NthField(tmp,".",2)
```

Now get the Hours

```
HH = mid(HH,3)
```

HH = H now (one digit hours)

Make HH as two digit with preceding "0"

```
if len(HH) < 2 then HH = "0" + HH
```

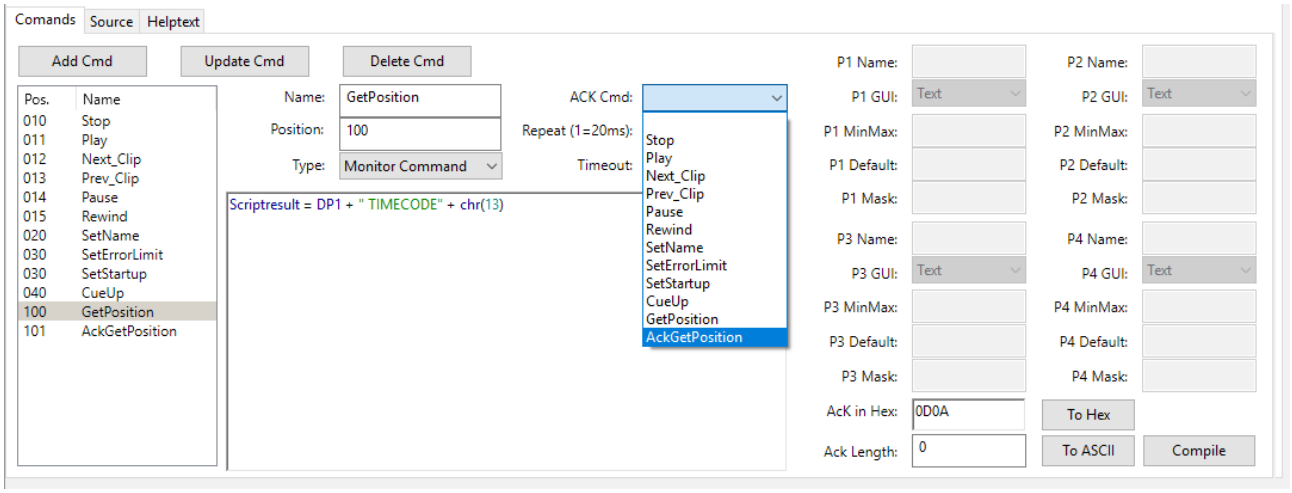
Build the Timecode String

```
TC = HH+"-"+MM+"-"+SS+"-"+FF
```

Set the Device Variable

```
SetDeviceVar("Position",TC)
```

Now we assign this Command as Ack Command for the GetPosition command. Select the GetPosition command, and use the drop down box *Ack Cmd* to select AckGetPosition.

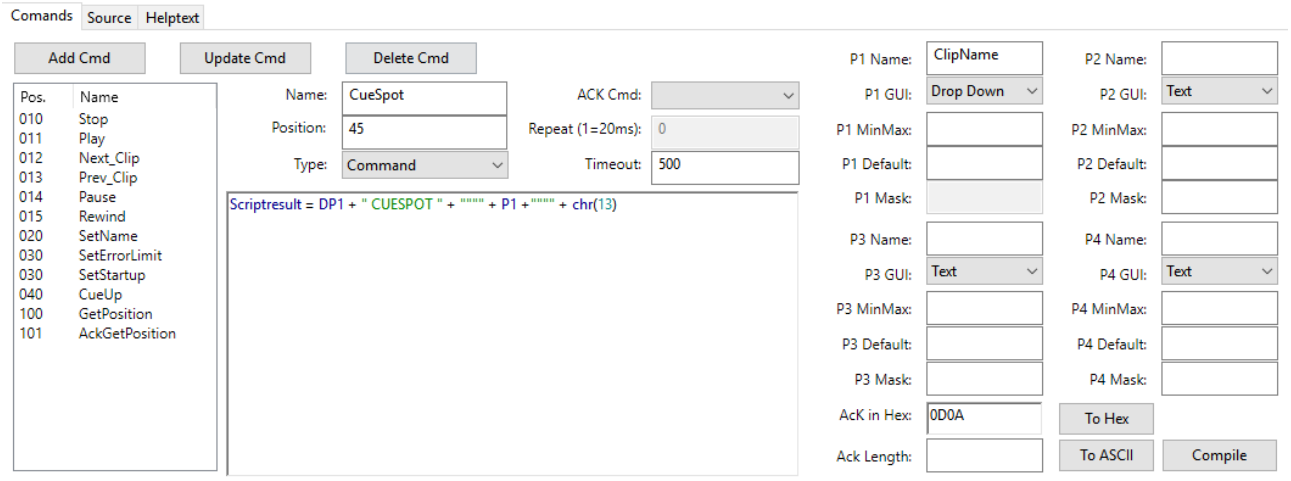


Automatic Parameters

This example shows how to automatic set the MinMax range of parameters. The SOLOIST knows the commands CUESPOT and PLAYSPOT, both need the name of the spot (clip) as parameter. It is of course possible to proceed in the same way than e.g. with the SetName command, and implement a text field that contains the clip name. But the SOLOIST also knows the command INVENTORY, which returns a list of all available clips on the machine. This leads to a more comfortable way to select a clip. We use the INVENTORY command to get all available clips, and then we put this list in the MinMax definition of

the CUESPOT and PLAYSPOT command. This presents the end user a picklist, and there is no need to type the clip names (and make mistakes).

To start with the example we create the CUESPOT command as shown below:



The script is only one line:

```
ScriptResult = DP1 + " CUESPOT " + """ + P1 + """ + Chr(13)
```

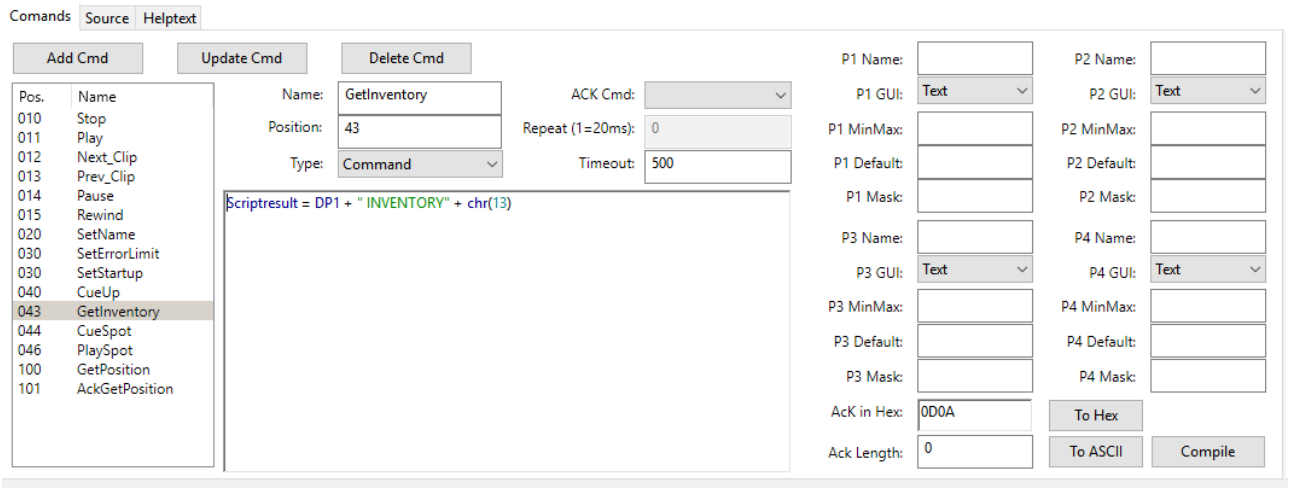
Respecting the SOLOIST protocol, the spot name has to be in quotation marks. The four quotation marks (“””“) before and after P1 do this job. P1 GUI is a Drop Down list that contains a list of available clips. But at this time we don't know how many clips are present on the SOLOIST player, so it stays empty.

Then we create the command PLAYSPOT in the same way

```
ScriptResult = DP1 + " PLAYSPOT " + """ + P1 + """ + Chr(13)
```

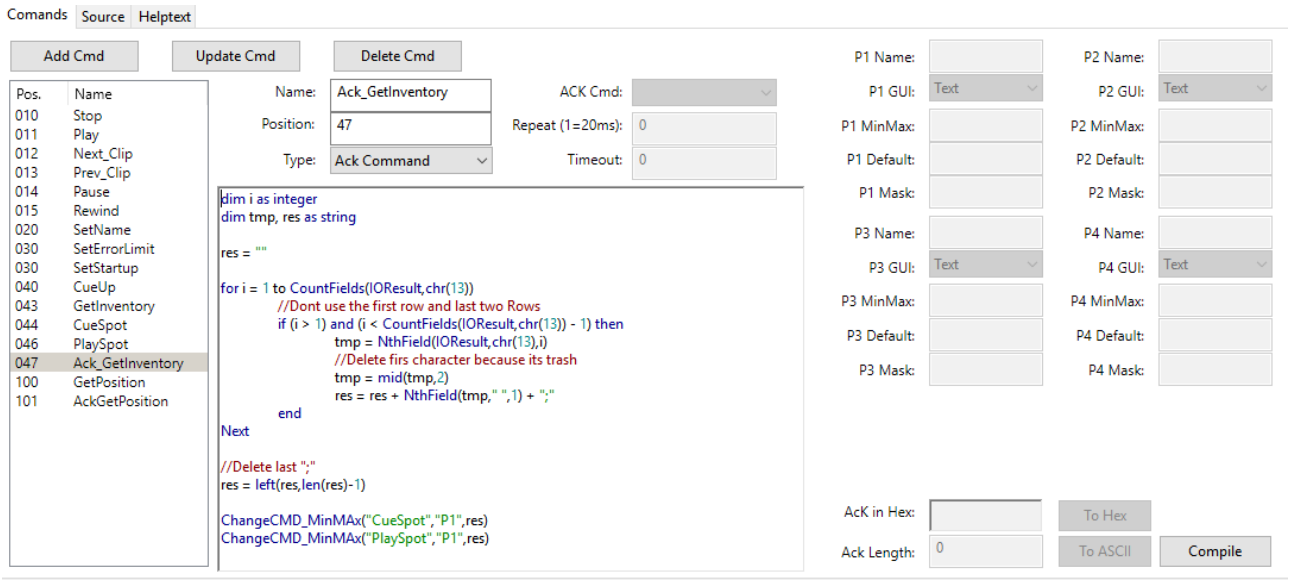
Now we have two new commands, but until now they are not useful because we can't select a clip name as parameter.

To get all available clips we create the command “GetInventory”. His job is to receive the clip list and fill the MinMax definition of PlaySpot and CueSpot commands. The GetInventory command is looks like this:



The GetInventory command does not fill the MinMax definitions. The command only tells the SOLOIST that the inventory list is requested. To read the list and fill the MinMax definitions, we use an acknowledge command, in this case

ACK_GetInventory.

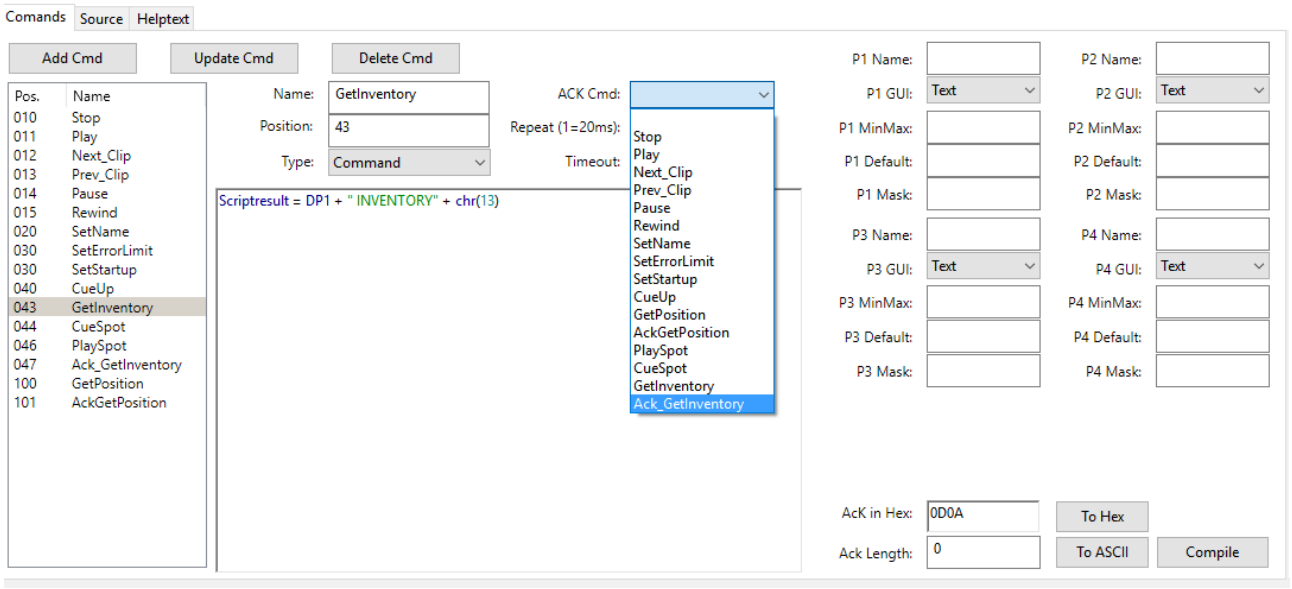


The script is quite complex, so here some explanations:

```
Dim i As Integer
Dim tmp, res As String

res = ""
For i = 1 To CountFields(IOResult,Chr(13))
   //Dont use the first row And last two Rows
   If (i > 1) And (i < CountFields(IOResult,Chr(13)) - 1) Then
      tmp = NthField(IOResult,Chr(13),i)
      //Delete first character because it's trash
      tmp = Mid(tmp,2)
      res = res + NthField(tmp," ",1) + ";"
  End
Next

//Delete last ";"
res = Left(res,Len(res)-1)
//Replace current MinMax Value with res
ChangeCMD_MinMax("CueSpot","P1",res)
ChangeCMD_MinMax("PlaySpot","P1",res)
```

Line 1 declares i as Integer variable. It is used in the For .. Next loop as counter. Tmp is used as temporary variable and res is used to hold the new MinMax list. The function `CountFields(IOResult,Chr(13)` tells how many lines (every line is terminated by chr(13)) in IOResult. The For .. Next loop is processed as many times, as chr(13) codes present in IOResult. The first line and the two last lines of IOResult don't contain any valid information, that's why the If .. Then statement present. The statement `tmp = NthField(IOResult,Chr(13),i)` assigns content of the i'st line to tmp. Then the content of tmp is added to res, followed by a semicolon.

Now we have to change the GetInventory command an tell it to use the acknowledge command.



Set Ack Cmd to “ACK_GetInventory” (Update Cmd and Compile)

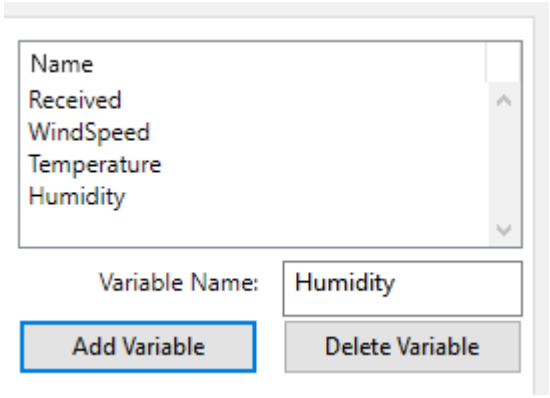
Events

From a device drivers point of view, an event is a message received from a device without requesting it before.

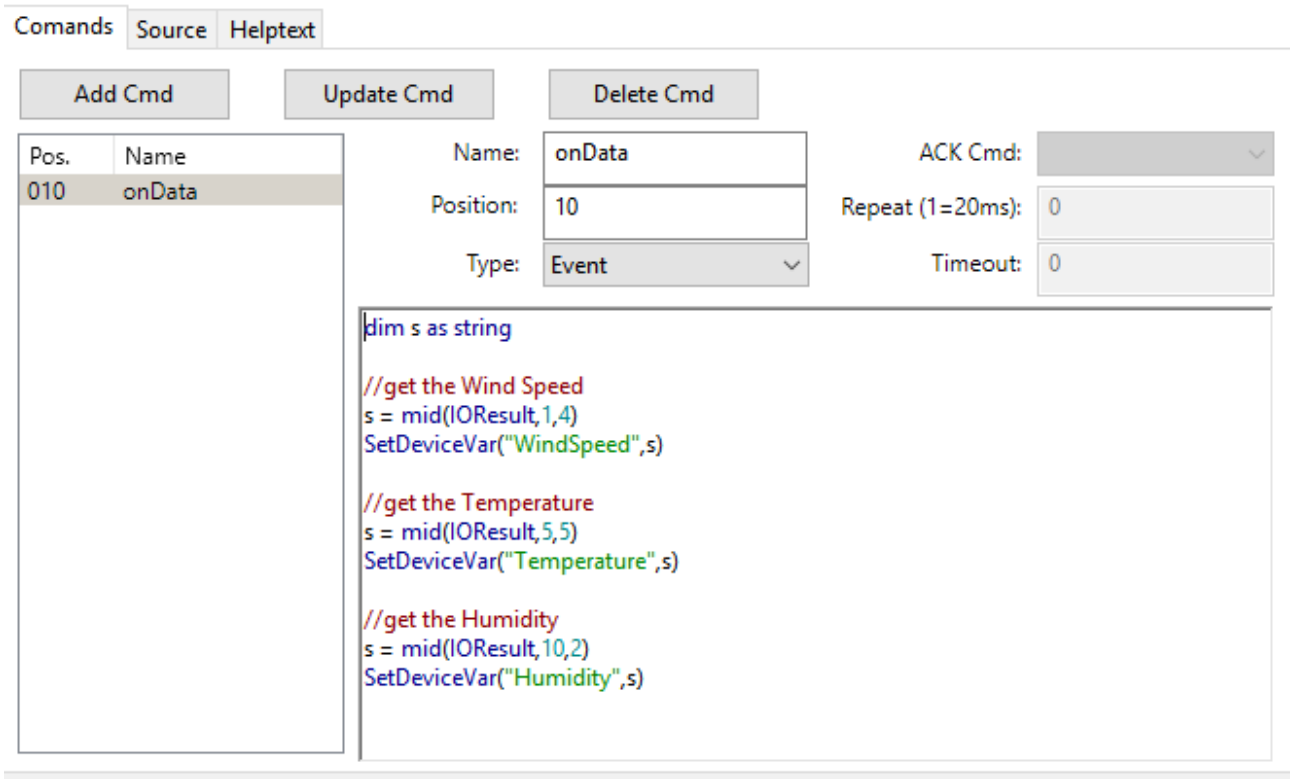
In this example I use a weather station that sends Wind Speed, Temperature and Humidity every second. The format of the string is 4 digit wind speed, 5 digit temperature and two digit Humidity, all packed in one string.

12,323,5087 means 12,3 km/h wind speed, 23,50°C and 87% Humidity. There is no termination character, but we have a defined length (11 characters).

First we need 3 device variables to store the received data



Then we need a new command of type *Event*. Here we use the *ACK Length* field to determine the amount of data that we need to process the event. If the Channel, that is associated with the device receives the data, it will look to the *Ack in Hex* and / or *Ack Length* field. If one of them matches then the data is send to the device and the event is processed.



Export Device Driver

To transfer a driver from one system to another, the Export / Import menu item in the device editor is used. Via device editors menu *File -> Export...* the currently selected driver is exported to a file.

In the save dialog enter / select a file name and click Save.

This is also used to move new device drivers to the Devices.rsd database, which has all the drivers shipped with V-Control. If you created a new device like in the examples above, then this device exists only in your project file. Export it and the import it in the Devices.rsd by selecting *Configure -> Edit Device Templates*. Then import the new device.

Import Device Driver

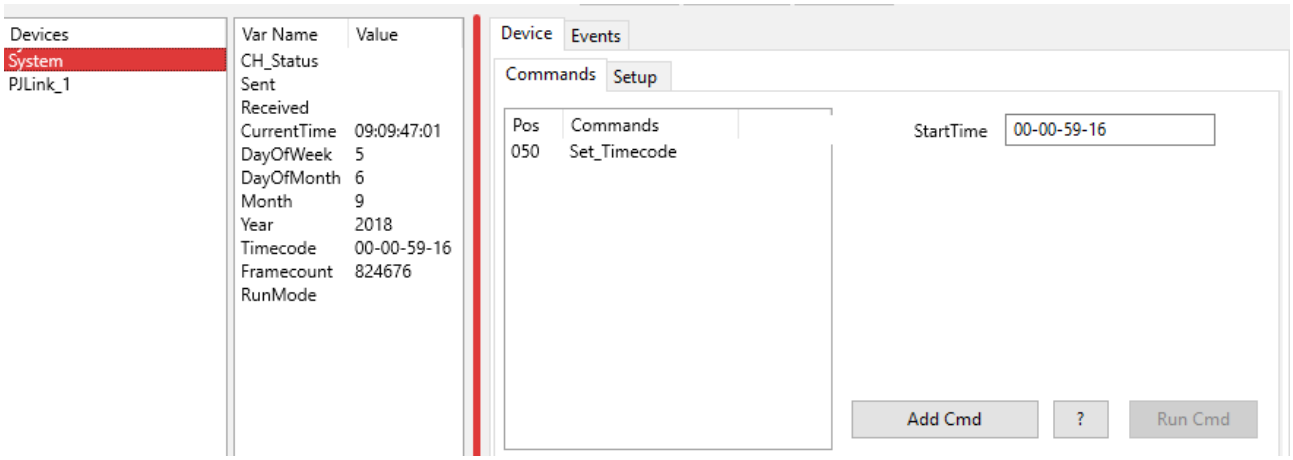
To transfer a driver from one system to another, the Export / Import menu item in the device editor is used. The Import menu item is available only if the device editor is launched via Configure -> Edit Device Templates The File -> Import command imports a previously exported driver. A dialog box asks for the name of the new driver.

Device Drivers

The following chapters provide information for some device drivers shipped with V-Control. These drivers are used for special tasks.

System

The *System* device is used to provide Time and Date information.



The device variable CurrentTime can be used in timelines for daily tasks. There is also a Timecode variable. This timecode can be set by the Set_Timecode command and is also for timeline use.

TCP Connectors

For some devices not direct accessible by V-Control, we provide TCP connectors. A TCP connector is a piece of software that controls the device native and provides a network interface to connect to other software products such as V-Control.

The TCP connector does all the communication with a device internally and provides a simple control protocol via network to communicate with V-Control. This also has the advantage that devices not available via network now network controllable. The V-Control connector is running on a computer where the device is connected to (typically via USB). This can be the same machine where V-Control is running, but doesn't have to.

So if you need to control a midi device for example, and this device is far away from your V-Control machine, then you can use an additional computer with a MIDI interface and the V-Control Connector software installed near the MIDI device. V-Control then opens a TCP connection to this machine and from V-Controls point of view this is the MIDI interface.

MIDI

There is no native support for USB MIDI devices in V-Control, but we have a nice work workaround. The TCP MIDI Connector interfaces between V-Control and your USB MIDI Device.

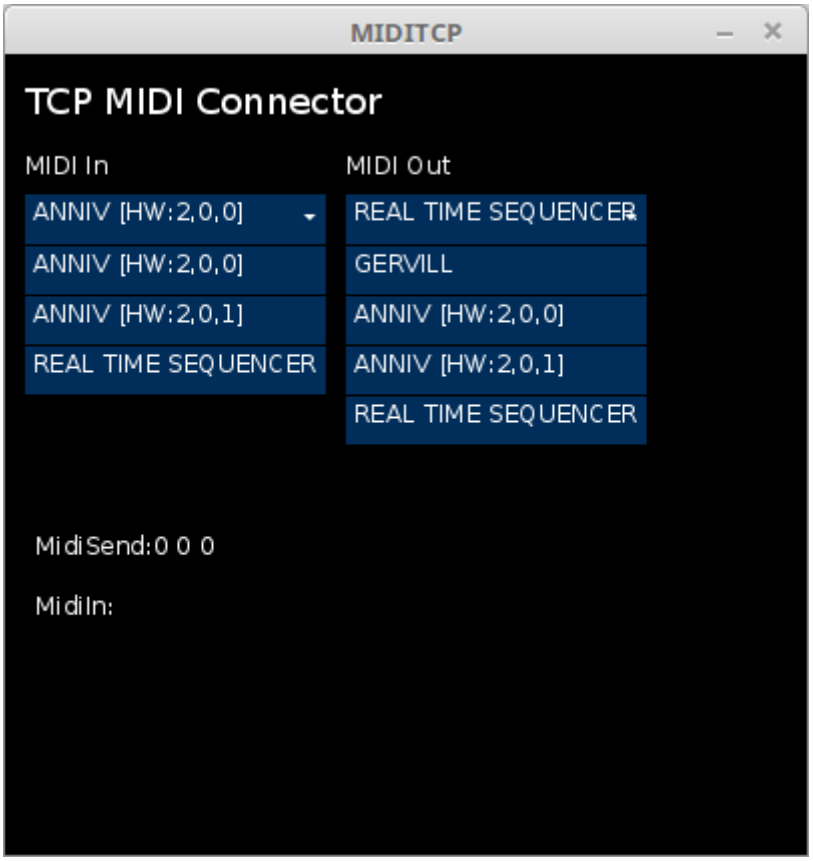
Note: This also works with rtpMIDI devices if a rtpMIDI driver is installed on your system.

To connect to the TCP MIDI Connector create a channel with the following specifications:

Channel Type: TCP Client

Port: 10002

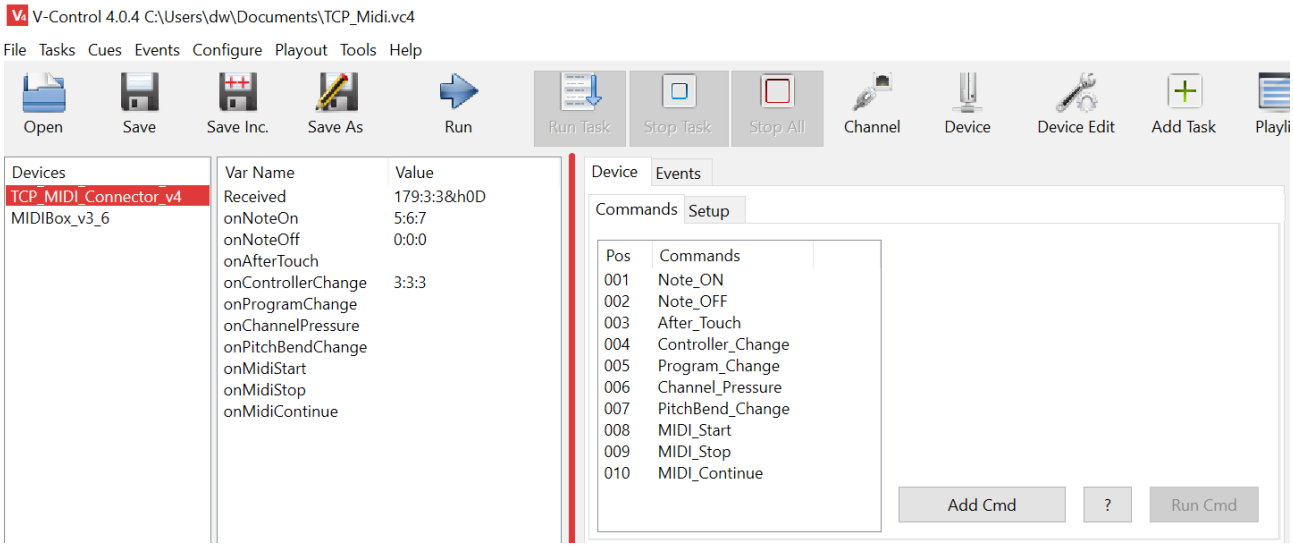
IP Addres: IP Address of the system with the TCP MIDI Connector (could be 127.0.0.1 if on the same machine then V-Control)



The TCP MIDI Connector is a Java software. On Windows, we provide a package with Java included, this is the reason why the zip file is relatively big. On other platforms you must install Java first.

Launch MIDITCP. In the left dropdown box, select the MIDI input. Use the right box for MIDI output. Once a MIDI Input and Output is selected, the configuration is finished. The software remembers the Input and Output and will use these interfaces next time it starts.

On the V-Control side, now use the TCP_MIDI_Connector_v4 driver.



The driver has a couple of device variables that can be used to fire events if a MIDI message is received.

Art-Net

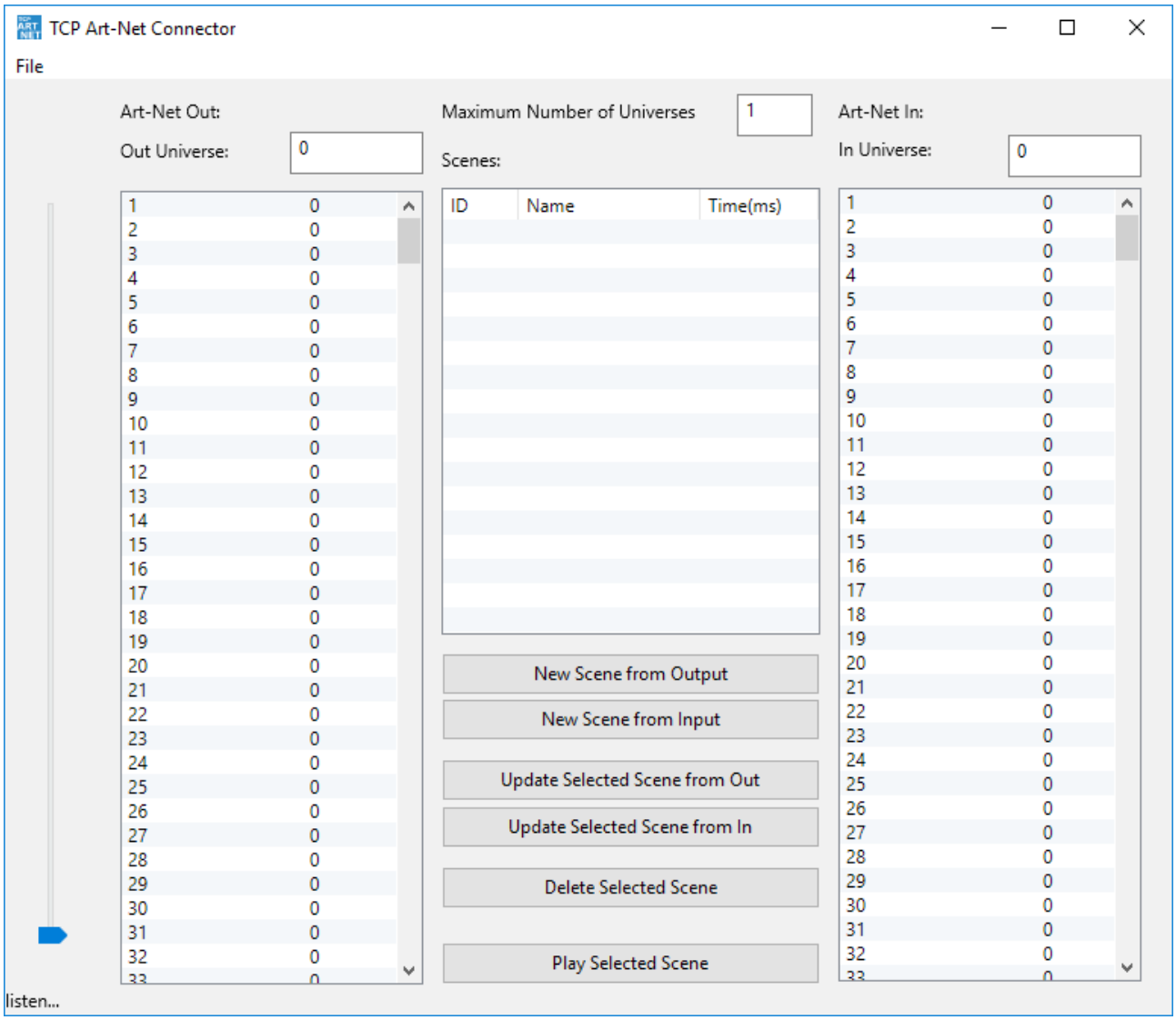
V-Control has a very basic native Art-Net Support (Driver Art-Net_v4). For more complex Art-Net / DMX Commands, we have the TCP Art-Net Connector.

As other V-Control Connectors, the TCP Art-Net Connector is a Software that can run on the same machie as V-Control, but also on a machine somewhere in the network. The last option might be useful if you have to control a huge number of Art-Net universes and channels.

A maximum of 256 universes is supported.

TCP Art-Net Connector runs on Windows 64 Bit, Linux 64 Bit and Raspberry Pi (Model 2 and 3).

Main GUI



The left list shows the output values of the currently selected output universe. To change the universe, change the value in the *Out Universe* textbox. To change a value, select the channel in the channel(s) column and use the slider. Alternatively click in the value column to change it.

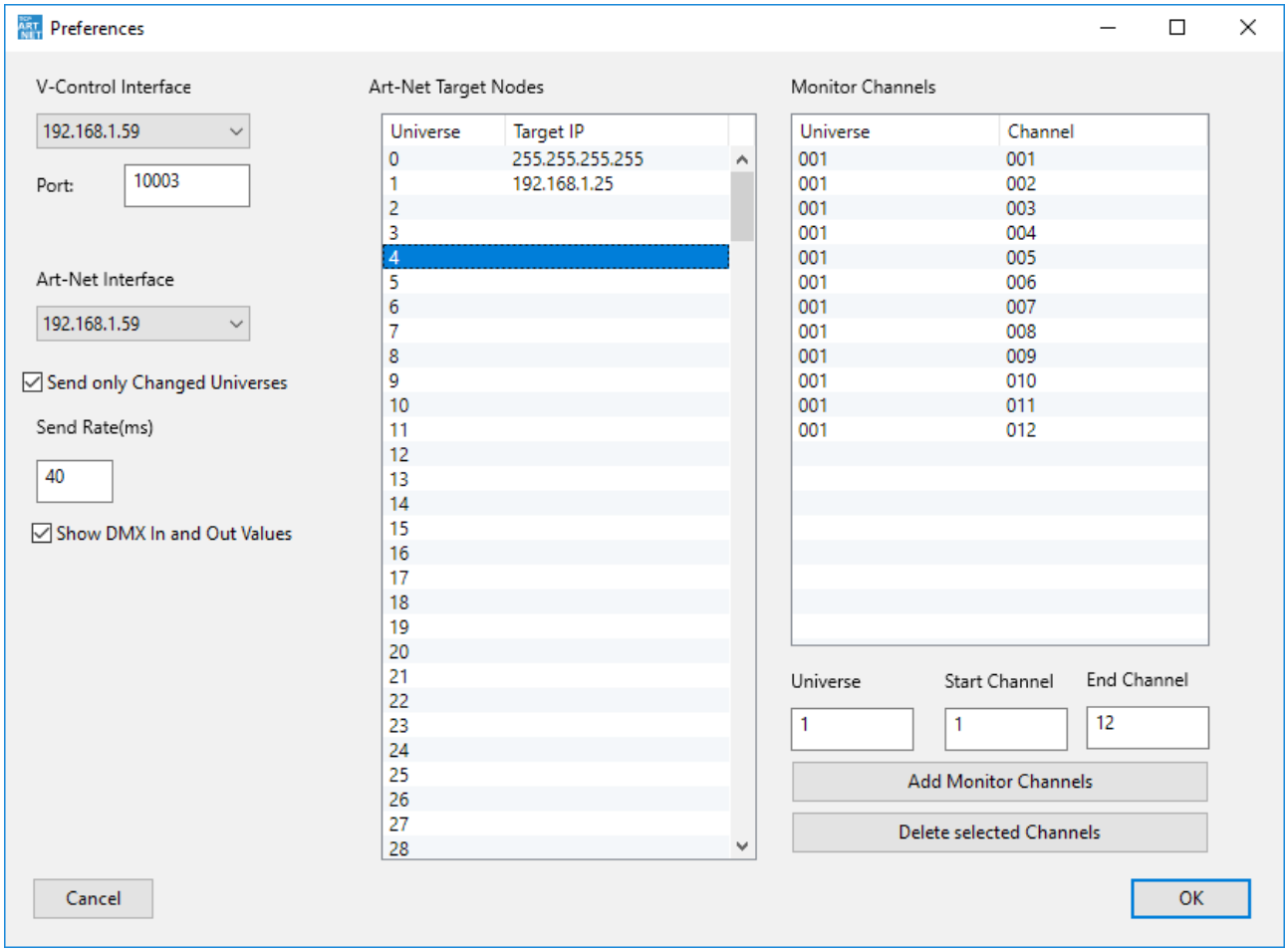
Double click the Channel number to enter a name here. Naming DMX channels is very helpful during programming to remember what a channel is used for.

The DMX values of the currently selected input universe are shown in the right listbox. Here you can select the universe by the *In Universe* textbox.

The listbox in the middle shows the avaiable scenes. A scene is a complete set of DMX values and can contain up to 256 universes. Scenes can be created from output or input values.

The *Maximum Number of Universes* reduces the bandwidth on the notwork output. Here you can select how many of the 256 universes you will use. It starts always by 0, so if 10 is entered here, universes 0-9 can be used.

Preferences



If the system that is running the TCP Art-Net Connector has more then one network interface, it is a good idea to use different interfaces for Art-Net and V-Control communication. Select the V-Control inteface in the *V-ontrol Interface* drop down box. The Art-Net interface is selected in the *Art-Net Interface* drop

down box. **ATTENTION:** because the network interface is bound to a specific IP address, 127.0.0.1 (localhost) will not work as V-Control channel, even if the TCP Art-Net connector is running locally on the V-Control machine.

The *Send only changed Universes* checkbox works as follows: If any of the universes that have a valid IP address have changed its content (DMX values), then these universes will be sent. In this case, the TCP Art-Net Connector sends only if something has changed, and only those universes that have changes in. If nothing has changed nothing wil be sent. For most modern Art-Net devices this is OK, however, if the Art-Net devices you are controling needs permanent DMX values, then uncheck this.

The *Send Rate (ms)* determines the update frequency of DMX values. The default is 40 ms, which equals 25 frames per second.

The *Show DMX In and Out Values* checkbox determines if the DMX values will be shown in the main GUI. Because of the underlying XOJO Framework, GUI updates are CPU expensive. On Raspberry Pi, this should only be checked for testing and maintenance.

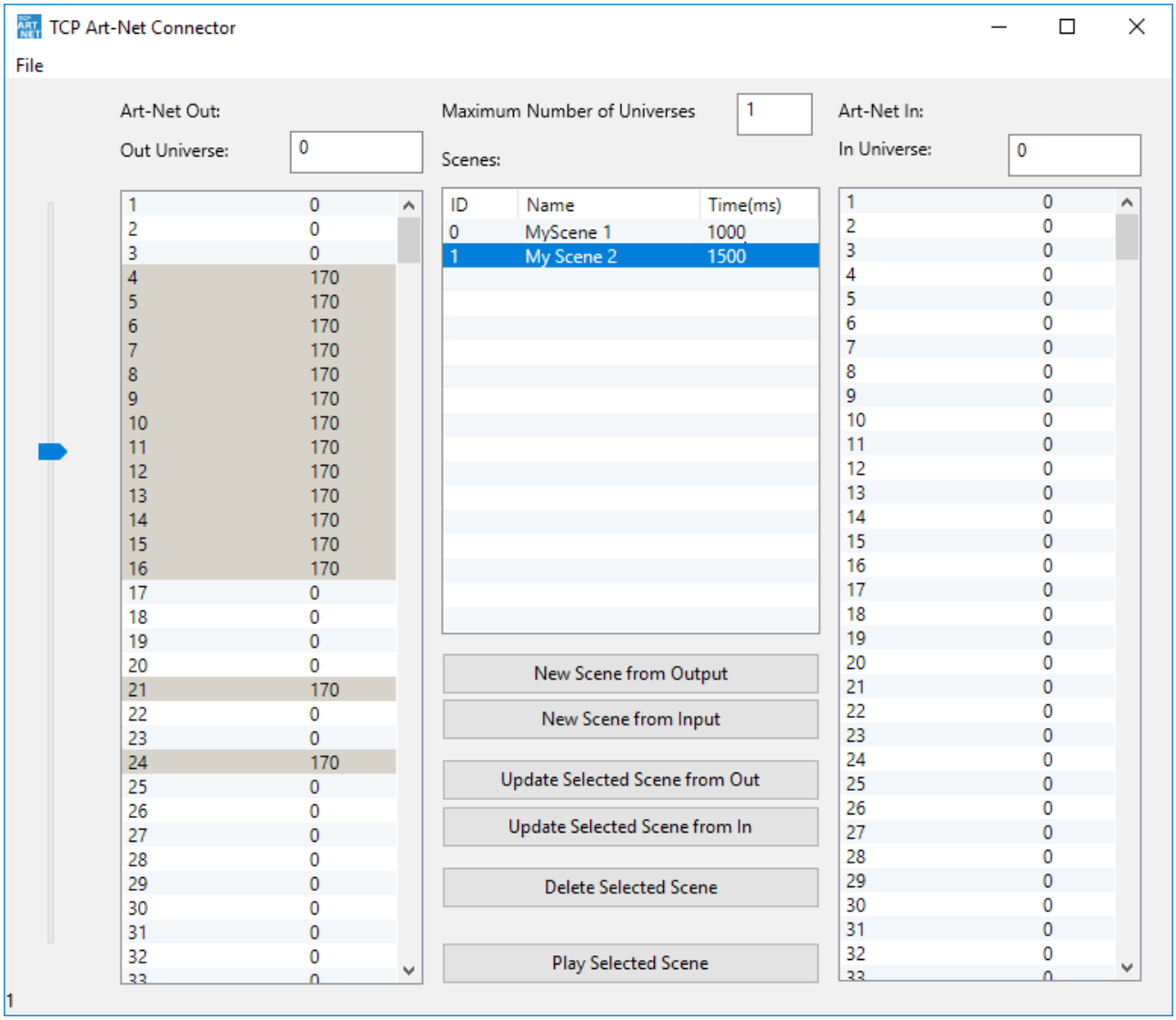
Art-Net target Nodes

This list is important. For each universe, we need a valid IP address here. If a universe has no IP address assigned, it will not send or receive anything. To reduce the bandwith on your network you should use unicast here if possible (universe 1 in the screenshot above makes use of it). For broadcasting, use the broadcast address of your network (universe 0 in the screenshot above). Multiple Art-Net Targets can have the same IP Address. If you have an Art-Net node with 4 universes, then all these 4 universes will have the same IP address.

Monitor Channels

TCP Art-Net Connector can send notifications to V-Control if a DMX value has changd. This can be used to trigger V-Control by Art-Net. To reduce the bandwidth and processing load for V-Control to the minimum that is needed, you have to select the DMX channels for monitoring. Enter the universe of the monitor channels in the *Universe* textbox, and then the *Start Channel* and *End Channel*. Finally click the *Add Monitor Channels* button. To delete Monitor Channels, select them in the listbox and click the *Delete selected Channels* button.

New Scene from Output



To create a scene from output, select a universe in the *Out Universe:* textfield. Then select the channels you want to change in the left listbox. **Make sure to click the channel number, not the value.** Now move the slider to the position you need. This procedure is good if you want to set more then one channel to the same value. If you need individual DMX values for the channels then you can enter the value directly by clicking on the value. Alternativeliy you canm select other channels by selecting then and use the slider.

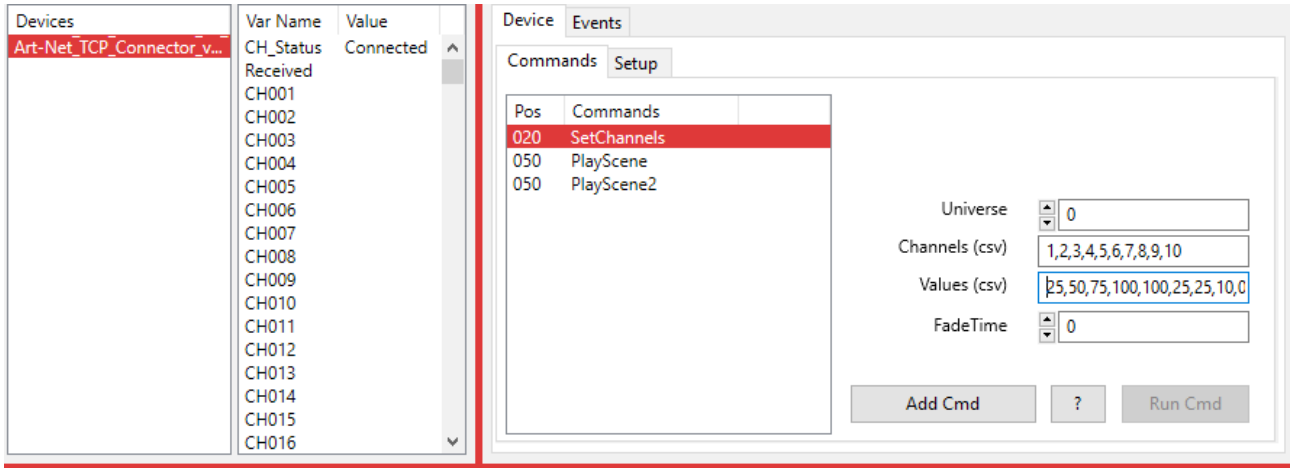
If more then one universe is involved in the scene, then change the universe and proceed like you did before. If all channels of all universes are set, click the *New Scene from Output* button. A new scene is added to the list. By default, the scene name is the same as the scene ID. Change the scene name by Double clicking it and enter a meaningful name.

The fade Time for a scene is entered by double clicking the *Time* column. Enter here the time in ms.

New Scene from Input

Creating scenes from Output will not work good on a complex DMX show. For that reason, TCP Art-Net Connector can create scenes from input. In this case you capture the output of a light controller / desk. Press the *New Scene from Input* button to create a new scene. The scene captures all values of all universes that are send to the TCP Art-Net connector.

Use with V-Control



To use TCP ARt-Net Connector we need a TCP channel for comunication. **ATTENTION:** because the network interface is bound to a specific IP address, 127.0.0.1 (localhost) will not work as V-Control channel, even if the TCP Art-Net connector is running locally on the V-Control machine. You allways need to enter the real IP address.

The Device has 512 Device Variables, one for each DMX channel. If the TCP Art-Net connector detects a value change on one of the selcted Monitor Channels, the Device Variables for those Channels are updated and can be used as Event Trigger.

The *SetChannels* command is used to set some DMX channels to a new value. Select the Universe in the *Universe* field. Enter the DMX channels involved in the command as comma separated list (csv) in the *Channels (csv)* field. Then enter a DMX value for each channel, also as csv list. The *Fade Time* determines the duration that is used to fade these channels from their current value to the final value.

The *Play Scene* command uses a drop down box to select a scene that is available in the TCP Art-Net Connector. The available scenes are updated automaticallyvery 3 seconds by a monitor command.

Play Scene2 does the same then *Play Scene*, the difference is that you have to enter the scene name in a textfield.

Elgato Stream Deck

The TCP_StreamDeckConnector integrates Elgato Stream Deck in V-Control. It is designed to work with the original Stream Deck Software from Elgato. To use the TCP_StreamDeckConnector you need to install the original Software from Elgato's website. Third party software might work as well but is not tested so far.

With the TCP_StreamDeckConnector you can call tasks and control the Playlist. All you need is to setup a launcher, point to the TCP_StreamDeckConnector's exe file and set some arguments to lauch it.

Prepare V-Control

V-Control needs to be in Remote Mode. See Options TCP Remote for details. The default TCP Port is 10101. Check the *Enabe TCP Remote Service* Checkbox.

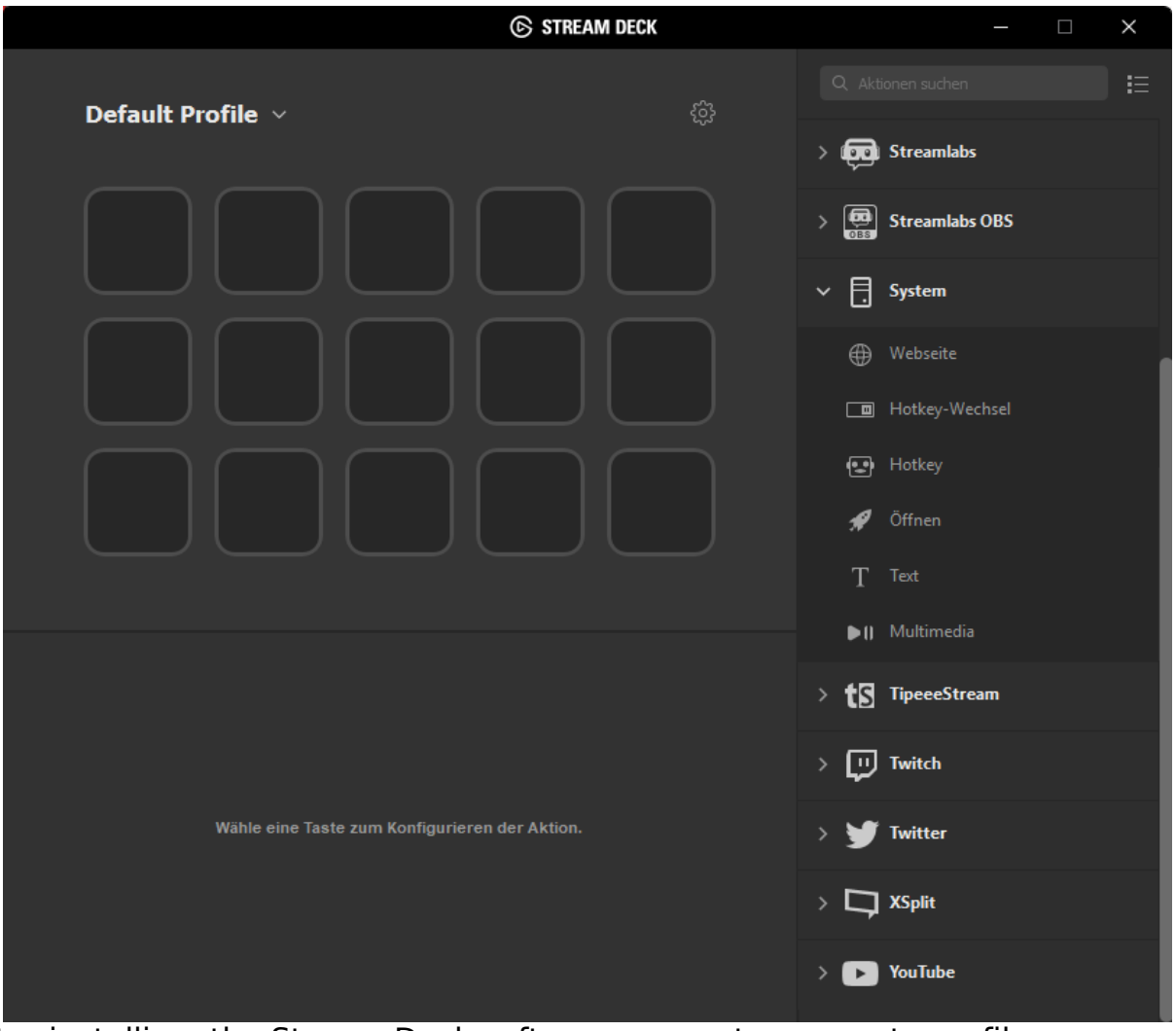
Prepare TCP_StreamDecConnector

TCP_StreamDeckConnector has no visible parts. It's only job is to connect to V-Control, send a command and quit. For that reason, you need to configure it by it's ini file. The ini file is very simple and has only two entrys.

```
Port=10101
Address=127.0.0.1
```

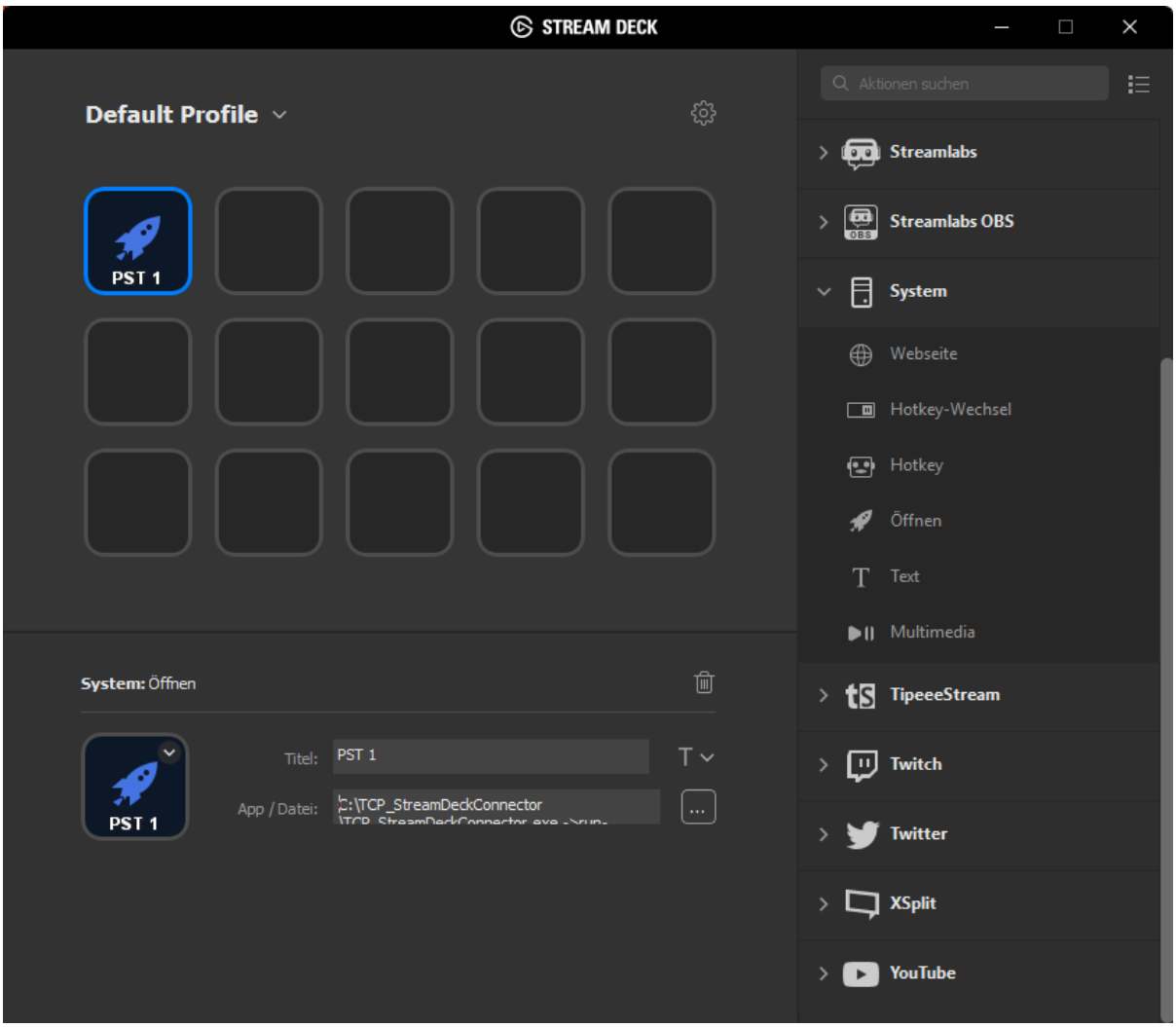
If TCP_StreamDeckConnector, Elgato Steram Deck Software and V-Control are running on the same computer, and V-Control uses the default TCP Remote Port, then there is nothing to configure. In any other case change the Port and / or Address. The Address must be the V-Control Address, the Port must be the Remote Port configured in V-Control.

Prepare Elgato Stream Deck Software



After installing, the Steram Deck software presents an empty profile.

Now add a new launcher. In this example we want to call a task that is recalling a preset in a E2 for example. So we need a task in V-Control that is doing this. Let's assume this task is named “Preset 1” (very inventive), and we want to label the button as “PST 1”



Drag the launcher to a button field and enter the button Title. Now click the three dots in the *App/File* field and select TCP_StreamDeckConnector.exe. Click in the text field and move the cursor to the end of the filepath. V-Control needs two parameters, telling it what to do. The first parameter is the *->run* parameter, telling V-Control that a task should run now. The second parameter determines the task itself.

Here is the complete entry:

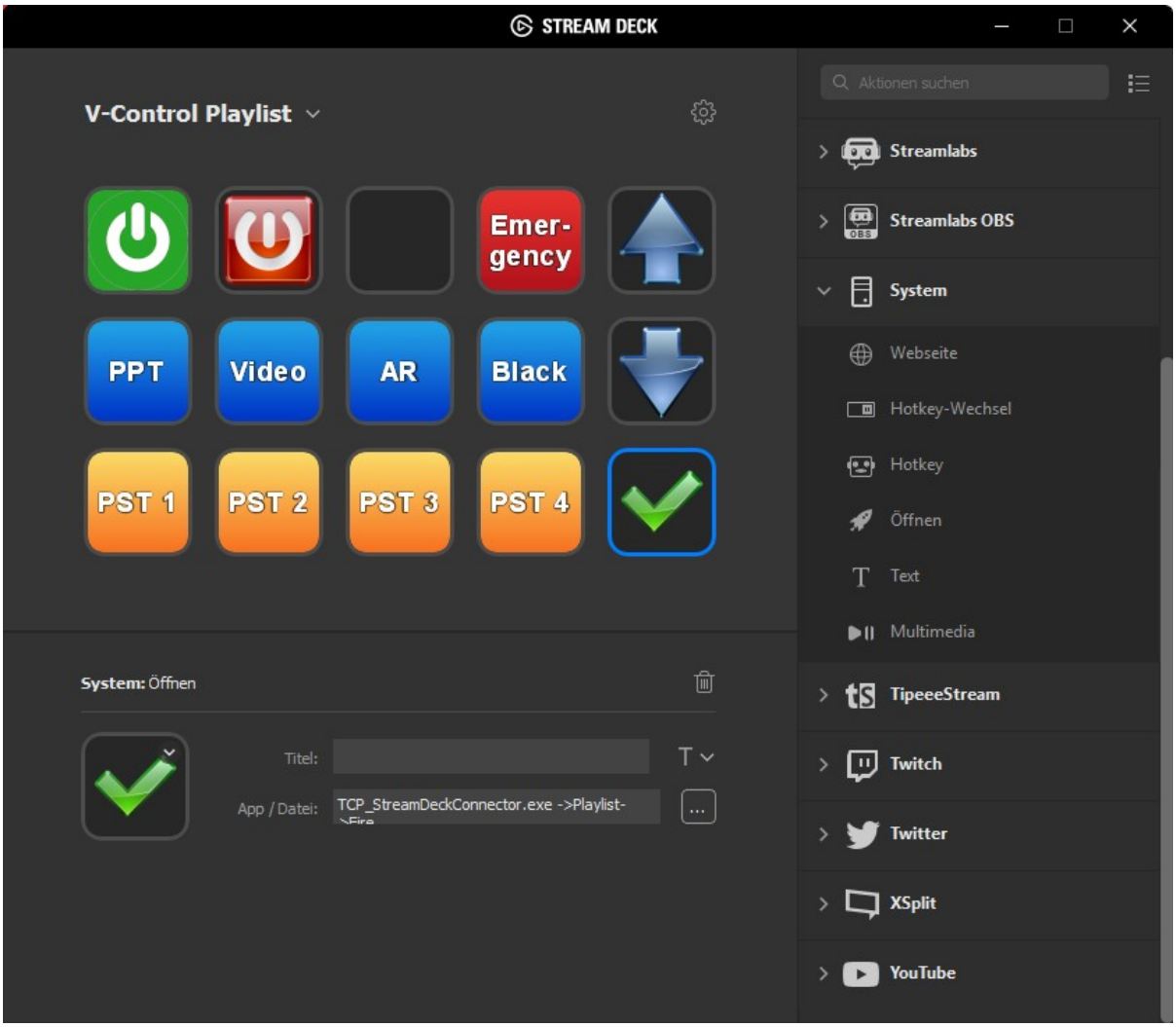
```
C:\TCP_StreamDeckConnector\TCP_StreamDeckConnector.exe ->run->Preset 1
```

The second type of commands are Playlist commands. These commands are used to move in the Playlist up and down and fire a task from the Playlist. The method is the same as running tasks. Add a launcher and select TCP_StreamDeckConnector.exe. The first parameter is ->Playlist, the second one can be Up, Down or Fire.

Here are the examples:

```
C:\TCP_StreamDeckConnector\TCP_StreamDeckConnector.exe ->Playlist->up
C:\TCP_StreamDeckConnector\TCP_StreamDeckConnector.exe ->Playlist->down
```

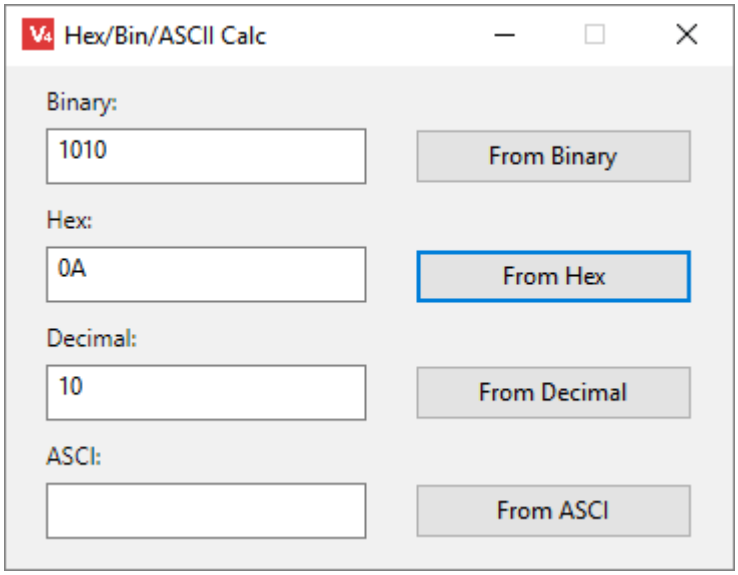
C:\TCP_StreamDeckConnector\TCP_StreamDeckConnector.exe ->Playlist->Fire



Tools

Hex/Bin/Dec/ASCI conversion

The Hex/Bin/Dec/ASCI Tool (available via main menu *Tools -> Hex/Bin/Dec/ASCI/Calc*) converts data from one of the four formats to all other.



To convert e.g the hex number 4142, enter the number in the Hex field and click the From Hex button. As result, the hex value is shown in binary, decimal and ASCI representation.

Remote Protocol

A V-Control system that offers its service via network or RS232 communication is called a node. Via this service, nodes can communicate with other nodes or applications. The service give access to the tasks and attached devices. V-Control services are available via TCP, UDP, RS232 and, thanks to the build in webserver, HTTP. The communication protocol for TCP, UDP and RS232 connections is the same, the HTTP protocol differs a little.

To enable one or more of the services, see the V-Control Options chapter.

TCP, UDP and RS232 Message Format

Start of Transmission	MSG Type (as string)	Separator	CMD (as string)	Separator	Parameter (as string)	End of Transmission
Chr(4)	0=command 1=result 2=event 3=ProgressMSG	chr(5)	command string	chr(5)	strings separated by chr(2)	Chr(6)

Websocket Message Format

There are two valid ways to control V-Control via web interface. The first one is to use a link in the page such as All LightsOff. This is very convinient if a simple GUI is needed. For complex, self updating GUIs the best solution is to use Websockets. The built in V-Control web server supports websockets from version 3.8 and later.

Start of Transmission	MSG Type (as string)	Separator	CMD (as string)	Separator	Parameter (as string)	End of Transmission
%04	0=command 1=result 2=event 3=ProgressMSG	%05	command string	%05	strings separated by %02	%06

Start of Transmission

The beginning of a new message is always $H04 or ASCII Code 4. If used in http protocol, this the “/cgi-bin/” string is used instead (see Examples).

Msg Type

ID	Description
	Command, a message was sent from the client to the Server (V-Control).
1	Result, this message contain the result of a client's request. The Data is stored in the Parameter Field.
2	Event, the server sent a message without a leading client request.
3	ProgressMSG, the server sent a message that contain a progress status

Separator

The ASCII code 5 ($H05 in hex.) separates the individual parts of the message. If using http, the separator is the string “%05”.

CMD

The Command sent to the server.

Parameter

Parameters used by the command (optional in some cases)

End of Transmission

The end of a message is always $H06 or ASCII Code

Web Interface

The connection from the html page to a task or command is made by a link. Usually, a link is used to navigate to another web page. Clicking on a link opens a new browser window, or replaces the current windows content with the linked page. But if we make a link to a task, there is no web page returned (and we do not want this). The task is launched and the return is only an acknowledge, saying that the task is executed or not. So we have to take care that the acknowledge message is not replacing our current browser content. One solution is to work with frames.

Frames divide a web page into two or more sections. In this example, we have 2 sections (frames). The ack frame is a located at the bottom with 30 pixel height, the main frame uses the remaining space at the top of the ack frame. The frame definition is done in the file index.html, located in V-Control httpRoot folder. index.html is the page that is shown if you type the V-Control web server url in your browser (i.e. http://192.168.1.45:8080). The listing below shows the content of index.html.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Frameset//EN"
"http://www.w3.org/TR/html4/frameset.dtd">
<html>
<head>
<title>V-Control Web Example</title>
</head>
<frameset rows="*,30">
<frame src="main.html" name="main">
<frame src="ack.html" name="ack">
<noframes>
 <body>


 </body>
</noframes>
</frameset>
</html>
```

The definition of the two frames is done by <frameset rows=“*,30”>, saying that the bottom frame is using 30 pixel, and the top frame the remaining space. Then, the top frame is linked to the file main.html, and its name is set to “main”. The bottom frame is linked to ack.html and its name is set to ack. Via this name, the frame is accessible as target for a link. The page (file) main.html can contain a link to a task, and set the target for this link to ack.

```
<a href=“cgi-bin/scrRunTask%05AllLightsOff” target=“ack”>All
LightsOff</a>
```

The link refers to the directory cgi-bin, which is a virtual directory. All links to this directory are interpreted as command. A task with the name “AllLightsOff” is called. Notice that the target is set to ack. The browser will show the result of the command in the frame ack. The “%05” is used as separator between the command (scrRunTask) and the parameter (AllLightsOff). If a command has more than one parameter, the parameters are separated by “%02”.

If you like more information about html coding have a look at http://www.htmlcodetutorial.com and http://www.w3schools.com .

HTTP Message Format

Start of Transmission	CMD (as string)	Separator	Parameter (as string)
/cgi-bin/	command string	%05	strings separated by %02

Examples

TCP/UDP/RS232 Examples

To launch a task (Script or Cuelist) the scrRunTask comand is used. In the

example below the name of the task we want to launch is “ExampleTask”.

in Basic

```
Dim CMD As String

CMD = Chr(4)+"0"+Chr(5)+"scrRunTask"+Chr(5)+"ExampleTask"+Chr(6)
```

in Delphi / Pascal

```
var

cmd: string;

cmd := Chr(4)+'0'+Chr(5)+'scrRunTask'+Chr(5)+'ExampleTask'+Chr(6);
```

As result, the node send an acknowledge message.

Chr(4)	1	chr(5)	scrRunTask	chr(5)		Chr(6)

HTTP Examples

To launch a task (Script or Cuelist) via web browser, a link to the /cgi-bin/ directory attached by the command is made. The html file must be copied in the Webservers Document directory. It is then available at http://NodeAddress/Filename.htm. The complete example file is listed below.

```
<html>
<head>
<title>Remote</title>
</head>
<body>
<a href="/cgi-bin/scrRunTask%05wait1" name="ExampleTask"
target="anyFrame"><h1>Wailt1</h1></a>
</body>
</html>
```

Device Manager

CMD	Parameter	Result
devGetDeviceList		List of available Devices separated by chr(2)
devGetChannelList	DeviceName	List of channels assosiated with this device seperated by “,”
DevGetDeviceProperties	DeviceName	ID+chr(2)+DriverName+chr(2)+Name+chr(2)+Category+chr(2)+Helptext+chr(2)+Script+chr(2)+DP1Name+chr(2)+DP1GUIT

		ype+chr(2)+DP1MinMaxDef+chr(2)+DP1Default+chr(2)+DP1Mask+chr(2)+DP2Name+chr(2)+DP2GUIType+chr(2)+DP2MinMaxDef+chr(2)+DP2Default+chr(2)+DP2Mask+chr(2)+Manufactor+chr(2)+DeviceGUI
devGetCommandList	DeviceName	List of available commands of the selected device separated by chr(2)
devGetCommandProperties	DeviceName(2)CommandName	CommandName+chr(2)+Position+chr(2)+Script+chr(2)+TabID+chr(2)P1Name+chr(2)+P1GUIType+chr(2)+P1MinMaxDef+chr(2)+P1Mask+chr(2)+P1Default+chr(2)P2Name+chr(2)+P2GUIType+chr(2)+P2MinMaxDef+chr(2)+P2Mask+chr(2)+P2Default+chr(2) P3Name+chr(2)+P3GUIType+chr(2)+P3MinMaxDef+chr(2)+P3Mask+chr(2)+P3Default+chr(2) P4Name+chr(2)+P4GUIType+chr(2)+P4MinMaxDef+chr(2)+P4Mask+chr(2)+P4Default+chr(2) Timeout+chr(2)+AckCmd+chr(2)+NextCmd+chr(2)+AckString+chr(2)+AckLen
devRunCommand	DeviceName(2)ChannelList(2)Command(2)DP1(2)DP2(2)P1(2)P2(2)P3(2)P4	DeviceName(2)ChannelList(2)Command(2)DP1(2)DP2(2)P1(2)P2(2)P3(2)P4(2)Ack

Script Manager

CMD	Parameter	Result	Example
scrGetTasks		List of available tasks separated by chr(2)	(4)0(5)scrGetTasks(5)Task1(2)Task2(2).....Taskn(6)

scrRunTask	Task Name	“0” if successful or Error Number	(4)0(5)scrRunTask(5)TaskName(6)
scrStopTask	Task Name	“0” if successful	(4)0(5)scrStopTask(5)TaskName(6)
scrSetGlobalVar	VarName(2)Value	“0” if successful	
scrGetGlobalVar	VarName	Var Value	

CMD	Parameter	Result	Example
plsUp		Move up	(4)0(5)plsUp(5)(6)
plsDown		Move down	(4)0(5)plsDown(5)(6)
plsFire		Fire selected Task	(4)0(5)plsFire(5)(6)

Change Log

Version 4.7 (SEPTEMBER-2018)

- Unordered List ItemAdded Search Field for Tasks
- Search fields for Tasks and Events clears if ESC key is pressed
- Global Variable List moved from Program Tab to its own area. This list is permanent visible now
- New Task Model: Get rid of the old model with Threadpool Manager. Instead of having only one TaskType that is configured to be Cuelist, Script or Timeline, we now have an own class for every Task Type that shares the same lass interface This makes it easier to add new programming features in the future.
- Cut Copy Paste from Cuelist Cues to Timeline Cues and vice versa
- New Timeline design, Timeline works again
- TDevice: If receiving data from channel, and there is no pending ACK and no Event routine, then the data is deleted
- New Device Option: Fire On Unchanged Variable. Default is false which means a variable change event occurs only if the variable value is set AND the value is different from the old one. If this is checked (true) then the event is fired every time a variable is updated, even if the new value is the same then the old value
- Bugfix: If Precompile all is selected, fake compile errors for scripts are gone now

Version 4.6 (JUNE-2018)

- Add new Device variable Sent (shows the last command string)
- Help now works again showing the help file with the default pdf viewer
- Bugfix: SingleToStr function now works correct in Linux (MemoryBlock.LittleEndian = true)
- Bugfix: FloatStringToStr function now works correct in Linux (MemoryBlock.LittleEndian = true)
- Bugfix: Device Variable update was sometimes broken
- Driver Update: TCP MIDI Connector now works better with high Midi traffic
- Driver Update: Moxa ioLogic now has additional device variables showing the counter value if inputs are configured as counter
- New feature: The Engine Status Refresh Timer is now adjustable from 1 to 500 ms
- Added Reconnect Timer for UDP Sockets that fire in case of Error
- Change: precompiler Messages are now shown in status bar instead of Splash Screen
- Added MaximumBuffer option. If the ReceiveBuffer of a channel exeeds this amount then the buffer is cleared
- Added new DeviceVariable CH_Status which shows the channel status

(open, close, error etc.)
- New Driver: TCP Art-Net Connector
- New Connector: TCP Art-Net Connector supports up to 256 universes and capturing scenes from input

Version 4.4 (MAY-2018)

- GUI: Message shows “Not in Run Mode” if a task is started without being in Run Mode
- GUI: Added Toolbar Button for Countdown Window
- GUI: Toolbar now shows correctly on High DPI Screens on Windows
- IOManagaer: IOMan.OnTimerSendAction: if Globals.EnableEvents = False then remove all ioCommands
- IOManager: Incomming Data is now buffered before processd. The Buffer is read by a timer triggered function now.
- ScriptManager: Endless repeats don't let System hang on Stop (Added “If Cancel then return” to Interpreter.bas)
- New Driver: Onvif TCP Connector
- New Driver: Art-Net
- New Driver: MIDI TCP Connector

Version 4.3 (FEB-2018)

- Create Condition Template uses Variant as Default Event Variable Type
- Options->Remote is visible now
- Resolved: V-Control could freeze if a TCP connection is lost
- Device Variable Received now shows “TCP Closed” if a TCP Channel is not open for reading / writing
- New Options -> Show Engine Status Settings that enables / Disables Engine EventMessage Stack, IOQueue and Pending Acks values
- Delay now works more preceise

Version 4.0 (NOV-2017)

V-Control 4.0 comes with a new redesign of the control engine and an updated user interface.