1 Managing the visibility of #cmdSetup controls

If you want configuration items of your Setup window not to be displayed or not entered by the user, you can implement three methods to do this:

1. The control selector
2. The tag :enable:
3. The tag :visible:

The control selector lets you manage multiple controls at the same time by hiding the current not relevant controls. Hidden controls will not be queried (no :read: issued) and this will speed up your Setup initialization. The assumption is that these controls are not valid or not needed in the current state of the device. Caution: control variables of the hidden controls are not initialized. Don't use them on controls outside the selector control entry where that control is part of.

The tag :enable: will not hide the control but sets it in a disabled state. Contrary to the hiding of the selector control a disabled control will still issue a :read: command during initialization or when requested via :update:

The tag :visible: should nog longer be used.

1.1 Using the selector control

The primary function of this control is to hide controls that are not needed for the current state of the configuration. In this way you can hide non-active device options or non-valid configuration items for the current state of the device.

The selector can issue a :read: command to get the value for the selector entries, but it can also be an expression (mostly based on previously set control variables).

All device options that require to an active state switch to be turned off/on are a candidate for the selector control. Preferred method to switch these options is the buttonsOn control. In this case you can repeat the read command of the buttonsOn on the selector control to enable the required #cmdsetup controls. On the ON entry you can specify all controls that need to be visible. In the off state the name of the selector control will be displayed.

Note: OFF and ON are the values returned by the device. This can also be 0 and 1 or other values.

```
#cmdSetup buttonsOn Sweep_State Channel_1
:read: C1:SWWV?
:readmath: getElement(getMatch(value,"STATE,[^,]*(,|$)"),1,",")
:write: C1:SWWV STATE,#
:update: Sweep_Settings_visible_in_On_State
:string:
:color: (240,60,0)
CH1_Sweep_Off OFF
CH1_Sweep_On ON

#cmdSetup selector Sweep_Settings_visible_in_On_State Channel_1
:read: C1:SWWV?
:readmath: getElement(getMatch(value,"STATE,[^,]*(,|$)"),1,",")
; selector name will be displayed in OFF state
OFF
ON Channel_1.Sweep_Time Channel_1.Sweep_Start Channel_1.Sweep_Stop
```

Remarks:

- You need to put a :update: command with the selector name on the buttonOn control to update the items under the selector control when switching from state,
- The first line with controls will be used for previews and when the value is empty. Seems the best place to put the entry for the off state, if applicable.
- Don't use selector controls to switch one optional control. Just use the :enable: function,
- But if you have too many complicated :enable: structures on many controls the use of the selector is preferred,
- If a control is in one of the entries of the selector, the selector will control the visibility of this control. Be careful using the page name as control (e.g. Channel.) will put all items on the page under control of this selector,
- When an empty entry of the selector is selected (e.g. OFF) all controlled command are hidden and the selector name is displayed in the setup window,
- You can cascade selector controls when needed,
- When using the selector you cannot use the same control name for controls that are different in function. This can be solved by appending a _ at the end of a control. The controls State, State_ and State__ show the same name in the setup window, but are different controls.

Other type of control that frequently can be used for the selector control is the comboboxHot. Switching between functions of the device may need a different set of parameters to control and there we can use the selector again.

```
#cmdSetup comboboxHot Waveform Channel_1
:write: C1:BSWV WVTP,#
:read: C1:BSWV?
:readmath: getElement(value,1,",")
:update: Wave_selector
Sine SINE
Square SQUARE

cmdSetup selector Wave_selector Channel_1
:read: C1:BSWV?
:readmath: getElement(value,1,",")


SINE Channel_1.Frequency Channel_1.Phase
SQUARE Channel_1.Frequency_ Channel_1.Period_Channel_1.Duty
```

Both can be combined by updating the selector with the buttonsOn and let the selector be controlled by one of the controls in the selector. This control must be in every selector entry except the off-state selector entry. In the example it is Channel_1.Modulation_Type.

```
#cmdSetup buttonsOn Modulation_State Channel_1
:read: C1:MDWV?
:readmath: getElement(value,1,",")
:write: C1:MDWV STATE,#
:update: Modulation_Settings_visible_in_On_State State
:string:
:color: (240,60,0)
CH1_Modulation_Off OFF
CH1_Modulation_On ON

#cmdSetup selector Modulation_Settings_visible_in_On_State Channel_1
```

```
:read: C1:MDWV?
:readmath: getElement(value,1,",")=="OFF" ? "OFF" : getElement(value,2,",")
; selector name will be displayed in OFF state
OFF
AM Channel_1.Modulation_Type Channel_1.AM_Source
FM Channel_1.Modulation_Type Channel_1.FM_Source

#cmdSetup comboboxHot Modulation_Type Channel_1
:read: C1:MDWV?
:readmath: getElement(value,2,",")
:write: C1:MDWV #
:update: Modulation_Settings_visible_in_On_State
Amplitude AM
Frequency FM
```

And finally an example of cascading the selector controls.

```
#cmdSetup buttonsOn Burst_State Channel_1
:read: C1:BTWV?
:readmath: getElement(getMatch(value,"STATE,[^,]*(,|$)"),1,",")
:write: C1:BTWV STATE,#
:update: Burst_Settings_visible_in_On_State State
:enable: inList(Channel_1.Waveform,"SINE SQUARE RAMP PULS ARB")
:string:
:color: (240,60,0)
CH1_Burst_Off OFF
CH1_Burst_On ON

#cmdSetup selector Burst_Settings_visible_in_On_State Channel_1
:read: C1:BTWV?
:readmath: getElement(getMatch(value,"STATE,[^,]*(,|$)"),1,",")
; selector name will be displayed in OFF state
OFF
ON Channel_1.Burst_Type Channel_1.Burst_Burst_Type_Selector (+other controls)

………other controls

#cmdSetup radio Burst_Type Channel_1
:read: C1:BTWV?
:readmath: getElement(getMatch(value,"GATE_NCYC,[^,]*(,|$)"),1,",")
:readformat: u
:write: C1:BTWV GATE\_NCYC,#
:update: Burst_Type_Selector
:string:
N_Cycles NCYC
Gated GATE

#cmdSetup selector Burst_Type_Selector Channel_1
:read: C1:BTWV?
:readmath: getElement(getMatch(value,"GATE_NCYC,[^,]*(,|$)"),1,",")
NCYC Channel_1.Burst_Period Channel_1.Burst_Trigger
GATE Channel_1.Burst_Period_ Channel_1.Burst_Trigger_
```

1.2 Using :enable: tag

The primary function of this tag is to hide one *(or a few)* control that is invalid for the current state of the configuration. Where the selector control gets its input from a issued :read: command, this tag will switch it's associated control via the logic state of the expression on the tag, where *true* equals on and *false* off. Some controls will create variables with their name or page-name.control-name if pages are used, they are: Number, NumberInt, NumberDual (Adds 1 & 2 to the names), buttonsOn, radio, combobox, comboboxHot. These control variables can be used to control other controls.

Some examples:

```
:enable: Load==1 (no page present)
:enable: Channel_1.Load_Impedance==100000 (numeric test)
:enable: Channel_1.Load_Impedance==”100000” (string test)
:enable: Channel_1.Load_Impedance==”HZ” (other string test)
:enable: Channel_1.Load_Impedance!=”HZ” (is not test)
```

Testing for more than one value/variable is also possible:

1. One variable can have more valid values: use inList() function
2. Multiple variables need to be valid: use logical operator

Some examples:

```
:enable: inList(Channel_1.Waveform,"SINE SQUARE RAMP") (test for multiple
strings)
:enable: Channel_1.Burst_Cycle=="1" && Channel_1.Burst_Trigger=="INT"
```

2 Using functions to check device response

A :read: tag receives a response of the device. This can be only a value, one keyword and a value or a whole set of values with or without keywords. In this paragraph we have a look at some returned messages (string) and the functions we can use to get the required values of this string.

2.1 Scanning the device response for a value

There are multiple way to get the wanted value depending on the format of the response.

Here is a *(non-complete)* summary:

Response	Used functions	Result	Number
10.345	none	10.345	y
10.345Hz	:readformat: u	10.345	y
FRQ, 10.345Hz	:readmath: getElement(value,1,",") :readformat: u	10.345	Y
FRQ 10.345Hz	:readmath: getElement(value,1," ") :readformat: u	10.345	Y
C1:BSWV FRE 10.345Hz	:readmath: getElement(value,2," ") :readformat: u	10.345	Y
C1:BSWV FRE, 10.345Hz	getElement(getMatch(value,"FRE,[^,]*(,|$)"),1,",") :readformat: u	10.345	Y
C1:SWWV STATE,ON	:readmath: getElement(value,1,",") :string:	ON	N

Below are some more in-depth examples.

2.1.1 Getting the value from a key,value string response

Key, value strings are commonly used to send multiple values as a response.

```
Message: C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V
```

2.1.1.1 Function getMatch

This function returns a substring from the received string based on a regex. The most common use is the search for a keyword and associated value separated by a commas.

The universal form used (*will find the key,value in the middle of a string or at the end*) is:

```
getMatch(value,"KEYWORD,[^,]*(,|$)")
```

Examples:

```
Message: C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V
Function: getMatch(value,"FRE,[^,]*(,|$)")
Result: FRQ,100HZ

Message: C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V
Function: getMatch(value,"AMP,[^,]*(,|$)")
Result: AMP,2V
```

2.1.1.2 Function getElement

This function gets a substring from a string with a specific separator. The separator is specified by a regex and a number needs to be supplied for the index after splitting the string. Most common separators are a space or a comma. Index number starts at 0.

Regex for one or more spaces: “[ ]+”

Examples for comma:

```
Message: FRQ,100HZ
Function: getElement(value,1,",")
Result: 100HZ

Message: C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V
Function: getElement(getMatch(value,"FRE,[^,]*(,|$)"),1,",")
Result: 100HZ
```

2.1.1.3 Tag :readformat:

Finally we want to separate the real value from the appended unit with the :readformat: tag.

Example:

```
Message: 100HZ
Function: :readformat: u
Result: 100
```

2.1.1.4 Combined example

Most *key,value* response can be processed with this universal solution:

```
Message: C1:BSWV WVTP,SINE,FRQ,100HZ,PERI,0.01S,AMP,2V
Function: :readmath: getElement(getMatch(value,"FRE,[^,]*(,|$)"),1,",")
          :readformat: u
Result: 100
```

2.1.1.5 Tag :string:

Some device responses are default treated as a numeric value. Therefor the :string: tag is required when using text responses. Known commands are buttonsOn, radio and checkbox.

This will work with numbers:

```
#cmdSetup buttonsOn State Channel_1
:read: C1:BWWV?
:write: C1:BWWV #
CH1_Off 0
CH1_On 1
```

But this requires the :string: tag:

```
#cmdSetup buttonsOn Sweep_State Channel_1
:read: C1:SWWV?
:readmath: getElement(getMatch(value,"STATE,[^,]*(,|$)"),1,",")
:write: C1:SWWV STATE,#
:string:
CH1_Sweep_Off OFF
CH1_Sweep_On ON
```

Note: buttonOn will turn the indicator on based on the response on the SECOND entry. In above cases 1 and ON.

2.1.1.6 Tag :readformat:

Normally :readformat: u will remove the units from the response, see 2.1.1.4. But in some cases this does not work. You can use getMatchGroup if the unit does not change with the value.

Example:

```
#cmdSetup number Symbol_Rate Channel_1
:read: C1:SRATE?
:readmath: getMatchGroup(value,"VALUE,(.*?)Sa/s",1)
:write: C1:SRATE VALUE,#
Sa/s 1u MaxArbSrate
```

2.1.2 What to do if a value is not always present?

With the selector control you want to be able to switch the function/option, but in off state the required value is no longer present in the message.

2.1.2.1 The if function

This is where we can use the if function so we can check the value of two keys.

```
Message-1: C1:MDWV STATE,ON,AM
Message-2: C1:MDWV STATE,OFF
Function: :readmath: getElement(value,1,",")=="OFF" ? "OFF" : getElement(value,2,",")
           :readformat: u
Result Message-1: AM
Result Message-2: OFF
```

2.1.2.2 The match function

Match can check if a substring is part of the returned response by e.g. trying to find the keyword via a regex search ".*KEYWORD.*"

```
Message-1: COUP TRACE,OFF,FCOUP,ON,PCOUP,OFF,ACOUP,OFF,FDEV,0HZ
Message-2: COUP TRACE,ON
Function: :readmath: match(value,".*FCOUP.*") ? getvalue : "OFF"
Result Message-1: ON
Result Message-2: OFF
getvalue = getElement(getMatch(value,"FCOUP,[^,]*(,|$)"),1,",")
```

2.1.3 What to do if I need multiple values?

For the control multi a number of values separated by a space is needed. This can be done by using getMatchGroup. In this case we use a regex again to find the key and strip the units from the values. Finally the values are concatenated in one string separated by spaces.

```
Message: C1:HARM HARMSTATE,ON, HARMORDER,4,HARMAMP,1.264911064V,HARMDBC,-10dBc
Function: :readmath: getMatchGroup(value,"HARMDBC,(.*?)dBc",1) + " " +
getMatchGroup(value,"HARMAMP,(.*?)V",1)
Result Message: -10 1.264911064
```

2.2 Testing your :readmath: tag in the calculator

Normally you need the device to be operational and you have to reload the config every time you want to test a change to the :readmath: tag. It is however possible to test this without the device or the config itself. Just capture the result of the :read: request and use the calculator function to test your :readmath: tag.

This is how it works:

1. Open the calculator and enter the device response as variable "value" in the calculator, as follows: `var value="`*`response`*`"` and hit enter
2. Now past your :readmath: formula in the calculator without the :readmath: and hit enter
3. the result will be shown in de field above it
4. Not the correct result? Change formula and try again.

Example:

```
var value="FCNT STATE,ON,FRQ,0HZ,DUTY,0,REFQ,1e+07HZ,FRQDEV,0ppm"
getElement(getMatch(value,"REFQ,[^,]*(,|$)"),1,",")


Result: 1e+07HZ
```

You can also store the result in a new variable and display it:

Example:

```
var value="FCNT STATE,ON,FRQ,0HZ,DUTY,0,REFQ,1e+07HZ,FRQDEV,0ppm"
var result=getElement(getMatch(value,"REFQ,[^,]*(,|$)"),1,",")
displayVar(result)


Result: 1e+07HZ
```

Note: If the input field turns blue the entry has been accepted, red means an error on the input

2.3 Tricks with your :read: tag

2.3.1 Sending multiple commands

The :read: tag is not limited to sending one query command. You can send ":write:" commands before you do your query. This can be helpful if you need to set the device in a certain state before the query is send. Some device will send different data on the same query depending on the state of the device. You can put an optional [*OPC] command between your other commands when needed. Variables cannot be used in these commands.

Example:

```
#cmdSetup multi Harmonics_Order_7 Channel_2
:read: C2:HARM HARMORDER,7; C2:HARM?
:readmath: getMatchGroup(value,"HARMDBC,(.*?)dBc",1) + " " +
getMatchGroup(value,"HARMAMP,(.*?)V",1)
:write: C2:HARM HARMORDER,7,HARMDBC,#,HARMPHASE,#
number Ampl dBc -80 0
number Phase ° 0 360
```

2.4 Tricks with your :write: tag

2.4.1 Using underscore in :write:

If your SCPI command requires you to send a underscore it will be changed to a blank unless you escape this underscore using a backslash.

Example:

```
#cmdSetup radio Sweep_Marker Channel_1
:read: C1:SWWV?
:readmath: getElement(getMatch(value,"MARK_STATE,[^,]*(,|$)"),1,",")
:readformat: u
:write: C1:SWWV MARK\_STATE,#
:string:
Off OFF
On ON
```

2.4.2 Modified :read: input

If you want to write a completely different string than you receive with your :read: query, you can use :readmath: with an "if" construct to do this. Make sure your modified strings on the :readmath: match those on the parameter entries and you have to replace any spaces with underscores.

Example:

```
#cmdsetup radio CH1-CH2_Phase_Mode Coupling
; TEMP replacement using virtual key strokes
:read: MODE?
:readmath: getElement(value,1," ")=="INDEPENDENT" ?
"VKEY_VALUE,4,STATE,1;[*OPC];VKEY_VALUE,13,STATE,1" :
"VKEY_VALUE,5,STATE,1;[*OPC];VKEY_VALUE,17,STATE,1"
:write: #
:string:
Independent VKEY_VALUE,4,STATE,1;[*OPC];VKEY_VALUE,13,STATE,1
Phase_Locked VKEY_VALUE,5,STATE,1;[*OPC];VKEY_VALUE,17,STATE,1
```

3 The Setup window

3.1 Entering values in input fields

3.1.1 Input field turns Red

When an input field turns to red your input value is out of range. This can also happen when the device switches from mode, where a read will not return the required value anymore. Try

:emptyfield: tag for controls: number and numberInt to solve this. This value must however be within the specified range.

3.1.2 Input field turns Purple

When an input field turns to purple you changed the value but you didn’t Set it. This color change happens when you set the focus on another input field

4 Cavities

- You cannot use a dash (-) in a control name if you want to use it on an :enable: tag. This is seen as a minus sign and will give a variable error in Java.