

UniSketch TCP Client (aka "Raw Panel")

UniSketch TCP Client connects to a server on port 9923. This server would typically be a third party host system implementing the logic behind the commands sent from the "SKAARHOJ Raw Panel", but it could even be another SKAARHOJ panel with the "TCP Server" device core active in which case the client can "remote control" the server panel. The actions in the table below all relates to such a remote control scenario.

Raw Panel mode essentially is to let the server be a software application written to support the UniSketch TCP Client protocol and thus use a SKAARHOJ panel to simply send triggers such as keypresses, pulses and analog values over to the server which in turn maps them to actions in its domain. This has also historically been referred to as "dumb panel" since the panel does not know anything about the application it's being used in. In case of Raw Panel implementations, only the action "Tie to Remote HWC" is likely to be relevant.

This is a table of actions for UniSketch TCP Client:

<p>Tie to Remote HWC</p> <div> <div>Normal</div> <div>INS CP -</div> <div>UniSketch TCP Client: Tie To Remote HWC</div> <div>HWC: 1</div> <div>+</div> </div>	<p>Will send down / up / encoder pulses / analog values / speed values to the remote HWC by the number listed (unless zero is selected in which case the current HWC number is used). So for instance, if this is applied to a push button, when that button is pressed down, a Down action for that HWC is sent to the TCP Server we are connected to. Likewise, the return value of this element will be the return value retrieved from the remote UniSketch controller.</p> <p>Button colors: Responds to the return value of "HWC#xx="</p> <p>Displays: Responds to "HWCg#xx" and "HWCt#xx" which lets the server send text and formatting or graphics to the client.</p>
<p>Shift Level</p> <div> <div>Normal</div> <div>INS CP -</div> <div>UniSketch TCP Client: Shift Level</div> <div>Level: 0</div> <div>Hold Down</div> <div>Reg A</div> <div>Label: 0</div> <div>+</div> </div>	<p>See description for "Shift Level" from the System Device Core - only this all applies to shift levels on a remote UniSketch controller, not the local.</p>
<p>State</p> <div> <div>Normal</div> <div>INS CP -</div> <div>UniSketch TCP Client: State</div> <div>State: 0</div> <div>Toggle</div> <div>Reg P</div> <div>Label: 0</div> <div>+</div> </div>	<p>See description for "Shift Level" from the System Device Core - only this all applies to shift levels on a remote UniSketch controller, not the local.</p>
<p>Memory</p> <div> <div>Normal</div> <div>INS CP -</div> <div>UniSketch TCP Client: Memory</div> <div>A</div> <div>2</div> <div>Hold Down</div> <div>Label: 0</div> <div>+</div> </div>	<p>See description for "Memory" from the System Device Core - only this all applies to memories on a remote UniSketch controller, not the local. Notice that "Persist" is not implemented either.</p>
<p>Flags</p> <div> <div>Normal</div> <div>INS CP -</div> <div>UniSketch TCP Client: Flag</div> <div>Flag: 0</div> <div>Toggle</div> <div>Invert</div> <div>Feedback Flag: 0</div> <div>Label: 0</div> <div>+</div> </div>	<p>See description for "Flags" from the System Device Core - only this all applies to flags on a remote UniSketch controller, not the local.</p>

API for “Raw Panels”

UniSketch TCP Client can be used to set up an essentially “dumb panel” that only sends action triggers such as keypresses to the server and receives color values for a button and text or graphics for displays. This method is used when UniSketch TCP Client connects to a TCP Server on another SKAARHOJ controller. Likewise any other piece of broadcast hardware can implement a TCP Server that simply uses the same API to exchange information. The function of a “dumb panel” is implemented by using the “Tie to Remote HWC” action on any hardware interface component that is intended to work as such. Thus a “dumb panel” is only dumb to the extent that hardware interface components are consistently mapped to this action - in other words, a configuration mixed with other device cores or system actions is perfectly possible although that introduces more autonomy and “intelligence” in the panel itself.

In the following the term “client” is used for the panel with the UniSketch TCP Client device core running and the term “server” is used to indicate the broadcast device or software to which the client connects.

TCP settings

A SKAARHOJ controller with the “UniSketch TCP Client” device core will need to be set up with an IP address and it will attempt a connection to this IP address on **port 9923**. (See further down for how to change this port number.)

All communication forth and back is ASCII lines and terminated by <NL> (newline, “\n”)

Handshaking

After the TCP server responding on port 9923 accepts the connection, it will receive the command “**list<NL>**” from the UniSketch TCP Client. In response to this command, the server must respond with any initial data it wishes to dump followed (or preceded) by “**<NL>ActivePanel=1<NL>**” (Notice: text and graphics must come after “<NL>ActivePanel=1<NL>” is sent, in fact text and graphics should probably respond to the “map” command). This will confirm to the UniSketch TCP Client that it has been initialized and it will start to evaluate actions for the panels hardware interface components.

Periodically (like every 3 seconds) the UniSketch TCP Client will send the command “**ping<NL>**” to which the server must respond in some unspecified way, suggested “ack<NL>” for example. If the server does not respond to pings, the client will disconnect and reconnect.

Periodically (like every 60 seconds) the UniSketch TCP Client will send the command “**list<NL>**” to which the server can respond with state information (like button colors, including graphics, text). It’s not mandatory, more like a provision to compensate for any lost communication that might have resulted in the panel being out of sync with the server - something that ideally should not happen of course since all state information should have been perfectly shared over time.

The client will send “**BSY<NL>**” to the server if it feels it receives content quicker than it can process it. The server should respond by holding back new content until “**RDY<NL>**” or a “ping<NL>” is received from the client. Generally a whole bunch of data (like graphics and text) can be offloaded at any one time without fear of overload or missing packets since transport layers in TCP will take care of queuing, but the BSY / RDY commands are here to make sure the queue doesn’t grow out of hand. If it does, the panel will keep processing the queue and seem to lag behind in processing new commands.

The server is of course responsible to continuously update the client with new state information as necessary in relation to changes on the server.

Inbound TCP commands - from server to client

In general, see the documentation for the “TCP Server” device core which lists the basic command set supported. However, for the application of “Dumb panels” we will not use any of the registers (shift, state, flags, mem etc.) and focus only on exchange of information in relation to hardware interface components (HWCs).

The basic incoming commands the server could send are listed in this table:

Command	Description
HWC#xx=yy	<p>Status On/Off/Dimmed</p> <p>xx is the HWc number, yy is a byte defining the state of the component.</p> <p>The state, "yy", often translates into a color such as off / dimmed / on, but may also contain simple on/off binary information:</p> <p>Bit 0-3 forms a number from 0-15:</p> <ul style="list-style-type: none"> • 0 = Off • 2,3,4 = On (where 2=red, 3=green, 4="On" color (white or yellow by default)) • 5 = dimmed "On" color. <p>Bit 4: Blink flag for monocolour buttons. If set, a monocolour button will blink. This is to provide a way to indicate a different "on" value like a red (2) or green (3) but for a button that can otherwise just show "on".</p> <p>Bit 5: Output bit (32); If this is set, a binary output will be set if coupled with this hwc. Generally: Let bit 5 follow whether the "On" color (2,3 or 4) is commanded and let it be off if 0 or 5 is commanded.</p> <p>Bit8-11: Blink bits: If set 0001, the button will blink with a frequency of about 4 Hz, If set to 1000, the button will blink with a frequency of about 0.5Hz, if set to 1100 it will blink with a 0.5Hz frequency and a 75% duty cycle. The bits are a simple enabling mask against the systems millisecond clock and other combinations can create other blinking patterns.</p> <p>Most typically you would send these values back: 0 ("Off"), 36 (32+4 for "On") and 5 (for "Dimmed")</p>
HWCx#xx=yy	<p>Status On/Off/Dimmed</p> <p>xx is the HWc number, yy is a 16 bit word defining the extended output of the component.</p> <p>The rightmost 10 bits of this word is the value.</p> <p>Bits 11 and 12 are reserved for the individual output types to define.</p> <p>The leftmost 4 bits of this word is the output type:</p> <p>0=none</p> <p>1=Output Strength: Value from 0-1000, used to set a strength indication on and LED bar or position of a motorized fader.</p> <p>2=Directional Output Strength: <i>[future, todo]</i></p> <p>3=Shows steps 0 (no LEDs, off), 1=first, 2=second, 3=third etc. If beyond the number of LEDs, the full bar will light up dimmed.</p> <p>4=VU metering for audio (values 0-1000)</p>

Command	Description
HWCc#xx=yy	<p>Externally imposed button color: index or rrggbb</p> <p>xx is the HWc number, yy is a byte defining the color of the component if it is supposed to be set externally and not reflect the panel default</p> <p>Bit 7: If set, the color of the component is defined by this value, otherwise the panel default will be used (it's an enable-bit)</p> <p>Bit 6: Defines the interpretation of bits 5-0; If set, bits 5-0 represents the component color with "rrggb". If clear, bits 5-0 represents an index from 0-16 pointing to a preset color from this list (all of which are selected to be visually distinct from each other):</p> <ul style="list-style-type: none"> • 0: DEFAULT_COLOR, // Default • 1: 0, // Off • 2: 0b111111, // White • 3: 0b111101, // Warm White • 4: 0b110000, // Red (Bicolor) • 5: 0b110101, // Rose • 6: 0b110011, // Pink • 7: 0b010011, // Purple • 8: 0b110100, // Amber (Bicolor) • 9: 0b111100, // Yellow (Bicolor) • 10: 0b000011, // Dark blue • 11: 0b000111, // Blue • 12: 0b011011, // Ice • 13: 0b001111, // Cyan • 14: 0b011100, // Spring (Bicolor) • 15: 0b001100, // Green (Bicolor) • 16: 0b001101, // Mint <p>The colors marked "(Bicolor)" are the only ones recommended for use with red/green bicolor buttons on panels.</p>

Command	Description
HWc#xx=string	<p>Display text, tokenized string</p> <p>xx is the HWc number, <i>string</i> is a string tokenized by a vertical pipe character, " ", where each position represents a given parameter being either an integer, boolean or string.</p> <p>The format of <i>string</i> follows this:</p> <p>[value][format][fine][Title][isLabel][label 1][label 2][value2][values pair][scale][scale range low][scale range high][scale limit low][scale limit high][img]</p> <p><i>string</i> may not be longer than 63 chars</p> <ul style="list-style-type: none"> • [value] is a 16 bit integer representing the numerical value to be shown. If empty, it will not render at all (like format=7). • [format] defines how [value] is formatted: 0=Integer, 1=10e-3 Float w/2 dec. points, 2=Percent, 3=dB, 4=Frames, 5=1/[value], 6=Kelvin, 7=Hidden, 8=10e-3 Float w/3 dec., 9=10e-2 Float w/2 dec., 10=1 Textline (Title & value=size 1-4), 11=2 Textlines (Label 1, Label 2 & value=size 1-2). Default if empty is Integer. • [fine] is a boolean (0/1) that sets whether the fine indicator is shown under title bar. • [Title] defines the title string shown in the top of the display. Up to 10 chars long. • [isLabel] is a boolean (0/1) that sets if the title bar should be rendered as a "label". This is a convention used on SKAARHOJ controllers to indicate whether the content of a display shows the state of a given parameter (the current value) or if the display shows a label that indicates what will happen if the associated control component is triggered. Default is to show "state" which is indicated by a solid bar underlying the text. In "label" mode the title is rendered with only a thin line underneath. • [label 1] First text line under title. If [label 2] is omitted it will be printed in large font (5 chars). Up to 10 chars long. If small text is preferred without invoking [label 2], please set [value2] to something. • [label 2] Second text line under title. If not empty, both [label 1] and [label 2] will print in small letters. • [value2] Represents a second value. This is used if you use [label 1] and [label 2] as prefixes for [value] and [value2] along with settings for [values pair] • [values pair] ranges from 1-4 and indicates 4 variations of boxing of value pairs. • [scale][scale range low][scale range high][scale limit low][scale limit high] indicates different types of scales in the bottom of the graphic that can show a range of a given value. • [img] is an index to a system stored media graphic file. <p>Please check out the section later in this document for examples!</p>

SKAARHOJ DEVICE CORES

Command	Description
HWCg#xx=yy:string	<p>Display graphic, 64x32 pixels</p> <p>xx is the HWc number, yy is an index 0-2 and <i>string</i> is 1/3 of the image data encoded in base64</p> <p>Sending a 64x32 monochrome images to the panel is done by sending three consecutive lines, each representing 86, 86 and 84 bytes of the image data respectively, totalling 256 bytes. The index from 0-2 is used to indicate which part of the image is represented in the line. Always send them in this order. When index 2 reaches the client it will assume that all image data has been received and write it to the display.</p> <p>The 256 byte monochrome image data itself represents the image starting with byte 7 in the first byte being the upper left pixel (1=on, 0=off) and then progressing to the right and down (reading direction).</p> <p>SKAARHOJ has a helpful tool to convert images to 64x32 monochrome images:</p> <p>http://skaarhoj.com/FreeStuff/GraphicDisplayImageConverter.php</p> <p>Example:</p>
ActivePanel=1	<p>Activates panel</p> <p>Send this to activate the panel when "list" is received from the panel.</p> <p>It's recommended to prepend ActivePanel with <NL> in order to make sure, the full command gets noticed. Cases with short disconnects of the connection has proven to be vulnerable to missing this command which results in no initialization.</p>
list	<p>Asks panel to reveal some information about itself</p> <p>Returns _serial and _model</p>
Mem, Flag#, Shift, State...	See TCP Server device core (as mentioned previously in this document) for these commands - they are typically not relevant for Raw Panel implementations in third party systems.
SleepTimer=xx	Sets the global sleep timer in milliseconds: This is the number of milliseconds that shall pass before the panel will enter sleep. If zero, sleep is disabled.
SleepTimer?	Will request the global sleep timer value from the panel.
WakeUp!	Will wake up the panel if it was asleep.
encoderPressMode=xx	<p>In xx:</p> <p>bit 0: If set, encoders will return "Press" on "act down" (as well as press after holding down for 1 second). Default is 1.</p>

Outbound TCP commands - from client to server

This lists the outgoing commands from the client and which the server should understand and respond to.

Command	Description
HWC#xx[.mask]=string	<p>Trigger action from hardware component</p> <p>xx is the HWC number, <i>string</i> contains information about the trigger.</p> <p>Fourway buttons will also add the <i>mask</i>, which is a period followed by a number 1,2,4, or 8 indicating which edge was pressed on the button, respectively Up, Left, Down, and Right.</p> <p><i>string</i> can have any of these forms:</p> <ul style="list-style-type: none"> • "Down" : the component (typically a button or a GPI trigger or encoder knob) is pressed down (or held down for one second with encoders) • "Up" : the component is released again • "Press" : represents that Down and Up happened essentially simultaneously - a pulse • "Abs:yy" : A change, yy, to an absolute position (for example a T-bar). yy ranges 0 to 1000 • "Speed:yy" : A change, yy, to a speed (for example a spring loaded joystick). yy ranges -500 to 500 • "Enc:yy" : Pulses, yy, from an encoder. The sign indicates direction.
map=zz:xx	<p>Local HWC to External HWC mapping information</p> <p>zz is the native HWC number on the client panel and xx is the external HWC number used in communication with the server (the xx found in any other HWC command in this API). The command is issued initially and when changes in this mapping appears. It can be helpful for the server to know which HWCs are actually active on the panel. The information about the native HWC number is currently not of much interest but may be in the future. Notice how an external HWC may be associated with multiple native HWCs.</p> <p>Changes in the map can be used to track if a display may need update. For instance, the map is zeroed out in case of a sleep timeout on the panel and re-gains its values when it returns from sleep, thus giving the server a chance to repopulate the displays of the hardware components.</p>
BSY	Busy message
RDY	Ready message
list	Initialization status request, return "ActivePanel=1<NL>"
ping	Keep-alive, return "ack<NL>"
_model	_model=[Model / Product Key]
_serial	_serial=[Serial number]
_version	Software version from panel.
__topology	_topology=HTMLsvg (future)

Command	Description
_hwcs	_hwcs=json with HWC data (future)
_state:[reg]=xx	Informs about state register value (sent when changed)
_shift:[reg]=xx	Informs about shift register value (sent when changed)
_isSleeping=[0/1]	Informs about whether the panel sleeps or not (sent when changed)
_sleepTimer=xx	Returns the sleep timer in milliseconds: This is the number of milliseconds that shall pass before the panel will enter sleep. If zero, sleep is disabled.

Get started with test servers written in Python 3

We have written a few Python 3 scripts that will help you to get started quickly implementing support for SKAARHOJ panels in your software application. They can be downloaded from GitHub: <https://github.com/kasperskaarhoj/SKAARHOJ-Open-Engineering/tree/master/DeviceCoreFiles/UniSketchTCPClient>

When you run any of the scripts they will set up a TCP server on the host computer and listen on port 9923. A SKAARHOJ panel working as a UniSketch TCP Client and trying to connect to the IP address of the host computer will interact with the scripts.

They are a great resource to learn from and experiment with to get up to speed with integrating SKAARHOJ panels with your broadcast software or hardware solution.

We have put videos on YouTube as well that demonstrates these scripts with panels.




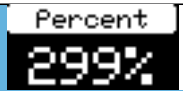




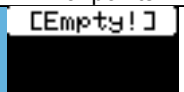
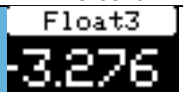

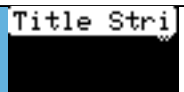
Future work

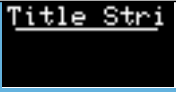
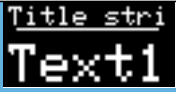
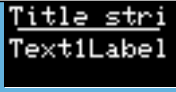
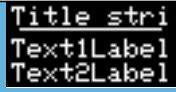
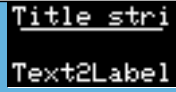

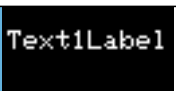
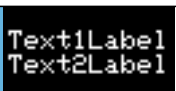
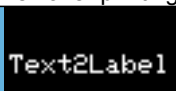
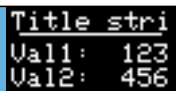
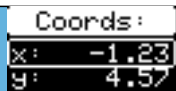




Further work is expected on the client, in particular ability to announce more details about itself, like number and nature of hardware interface components and the panel topology. This can lead to more generic support on the server side.

You are also very welcome to post suggestions and comments on the features to kasper@skaarhoj.com

Text based graphics

The displays on SKAARHOJ controllers are generally 64x32 pixel graphical displays - "tiles". Sometimes many of them are pooled together on a single, larger display, other times they are individual LCDs on a SmartSwitch. The easiest way to leverage the displays is to send a string with text / value content to the display. This is done with the command "HWCt#xx=string" as documented above. This section lists a number of example strings along with their rendered result. In the table you will find the string that resulted in a given graphic just below the graphic itself. The string is in italics and a comment is given below the string as well:

					
32767	-9999	32767 1 Float2	299 2 Percent	999 3 dB	1234 4 Frames
16 bit integer	16 bit integer, negative	Float with 2 decimal points	Integer value in Percent	Integer value in dB	Integer in frames
					
Reciprocal	Kelvin	[Empty!]	Float3	[Fine]	Title Str

999 5 Reciproc Reciprocal value of integer	9999 6 Kelvin Integer formatted as Kelvin	9999 7 [Empty! format 7 = emp- ty!	-3276 8 Float3 Float with 3 deci- mal points, opti- mized for 5 char wide space. Op to +/-9999	1 [Fine] 1 Fine marker set (the curvy thing on the right of the line), title as "label"	1 Title String no value, just ti- tle string (and with "fine" indica- tor)
					
Title String 1 Title string as la- bel (no "bar" in ti- tle)	Title string 1 Text1Label Text1label - 5 chars in big font	Title string 1 Text1Label 0 Adding the zero (value 2) means we will print two lines and the text label will be in smaller printing	Title string 1 Text1Label Tex- t2Label Printing two la- bels of 10 chars - automatically the size is reduced	Title string 1 Text2Label Printing only the second line - au- tomatically the size is reduced	Text1Label Text1label - 5 chars in big font, no title bar.
					
Text1Label 0 Adding the zero (value 2) means we will print two lines and the text label will be in smaller printing	Text1Label Text2Label Printing two la- bels - automati- cally the size is reduced	Text2Label Printing only the second line - au- tomatically the size is reduced	123 Title string 1 Val1: Val2: 456 First and second value is printed in small charac- ters with prefix labels Val1 and Val2	-1234 1 Coords: x: y: 4567 2 A box around the first label/value line	-1234 1 Coords: x: y: 4567 3 A box around the second label/val- ue line
					
-1234 1 Coords: x: y: 4567 4 A box around the both label/value lines	-500 1 Coords: 1 - 1000 1000 -700 700 1 A solid bar scale added below val- ue	-500 1 Coords: 2 - 1000 1000 -700 700 2 A moving dot scale added be- low value			

These graphics are generated from the test Python scripts. They can be very useful to experiment with other combinations.

Pixel graphics

Totally custom pixel graphics are another format you can use to generate content for the displays. Find sample graphics here:

<https://github.com/kasperskaarhoj/SKAARHOJ-Open-Engineering/tree/master/64x32%20Graphics>

To assist you in converting such graphics to the code you can send, we have provided a graphic conversion tool online which also outputs the 3 lines of consecutive commands, "HWCg#xx=", that you need to send them to the panel:

You will recognize the three lines in this screenshot as being the commands needed to send a graphic to the client panel. The “{” is substituted with the HWC number. There are ample examples of this in the Python 3 test scripts.

Find the graphical conversion tool here: <http://skaarhoj.com/FreeStuff/GraphicDisplayImageConverter.php>

Device Configurations

Device configuration options exist:

- Index 0: **Port number:** If different from 0, then this is the port number the controller will try to connect to on the device core IP

Example: If two UniSketch TCP Client device cores are active on the same IP 192.168.10.250, then setting “DC1:0=9234” will mean that the second device core (because of “1”) will try to connect on port 9234 instead of port 9923.

Changelog

January 2019:

- Added format “9” (XX.XX float) to graphics rendering
- Added format 10 and 11 for graphics rendering
- Added software version output to Raw Panel (UniSketch TCP Client)
- Added commands for handling, changing and reading sleep mode

- Added command (HWCx) for extended return values (like strength, VU meters, setting value of motorised faders).
- Internal changes in State, Shift and sleep mode is reported automatically to host system
- Recommends now to prepend ActivePanel=1 with <NL> to avoid missing initialisation in some cases of disconnect/reconnect