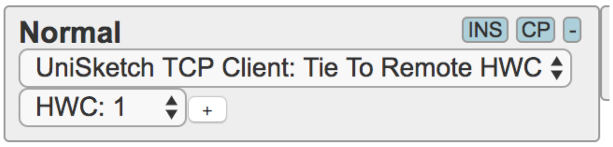
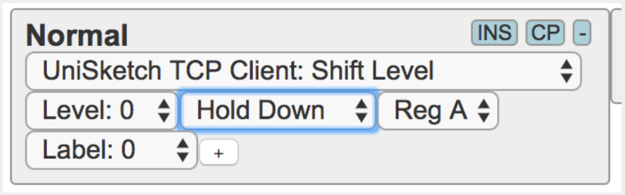
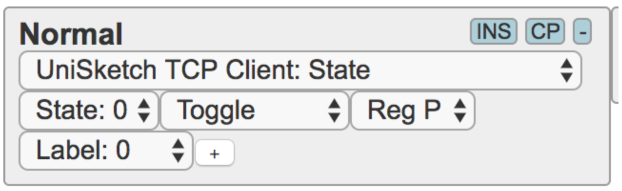
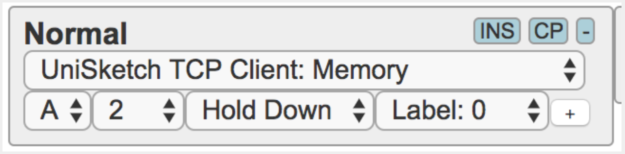
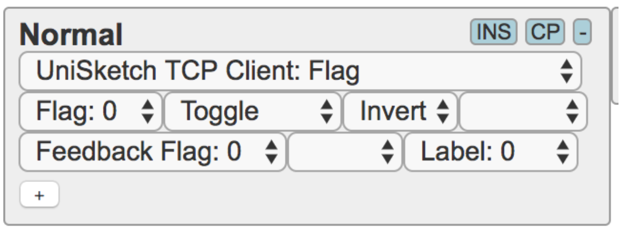
UniSketch TCP Client

UniSketch TCP Client connects to a server on port 9923. This server would typically 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.

Another typical use 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 would be the “dumb panel” use case. In this case, only the action “Tie to Remote HWC” is likely to be relevant.

This is a table of actions for UniSketch TCP Client:

Tie to Remote HWC  Normal UniSketch TCP Client: Tie To Remote HWC HWC: 1	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. Button colors: Responds to the return value of “HWC#xx=” Displays: Responds to “HWCg#xx” and “HWCt#xx” which lets the server send text and formatting or graphics to the client.
Shift Level  Normal UniSketch TCP Client: Shift Level Level: 0 · Hold Down · Reg A Label: 0	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.
State  Normal UniSketch TCP Client: State State: 0 · Toggle · Reg P Label: 0	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.
Memory  Normal UniSketch TCP Client: Memory A · 2 · Hold Down · Label: 0	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. Notice that “Persist” is not implemented either.
Flags  Normal UniSketch TCP Client: Flag Flag: 0 · Toggle · Invert Feedback Flag: 0 · Label: 0	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.

API for “Dumb 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 extend 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**.

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 ”**ActivePanel=1<NL>**” (Notice: text and graphics must come after ”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 continouosly 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 should support is listed in this table:

Command	Description
HWC#*xx*=*yy*	**Status On/Off/Dimmed** *xx* is the HWc number, *yy* is a byte defining the state of the component. This state often translates into a color such as off / dimmed / on, but may also contain simple on/off binary information: Bit 0-3 forms a number from 0-15: • 0 = Off • 2,3,4 = On (where 2=red, 3=green, 4="On" color (white or yellow by default)) • 5 = dimmed "On" color. Bit 4: Blink flag for monocolor buttons. If set, a monocolor 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". 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. Most typically you would send these values back: 0 ("Off"), 36 (32+4 for "On") and 5 (for "Dimmed")
HWCc#*xx*=*yy*	**Externally imposed button color: index or rrggbb** *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 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) Bit 6: Defines the interpretation of bits 5-0; If set, bits 5-0 represents the component color with "rrggbb". 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): • 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 The colors marked "(Bicolor)" are the only ones recommended for use with red/green bicolor buttons on panels.

Command	Description
HWCt#*xx*=*string*	**Display text, tokenized string** *xx* is the HWc number, *string* is a string tokenized by a vertical pipe character, "|", where each position represents a given parameter being either an integer, boolean or string. The format of *string* follows this: [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] *string* may not be longer than 63 chars • [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 formated: 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. 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. Please check out the section later in this document for examples!

Command	Description
HWCg#*xx*=*yy*:*string*	**Display graphic, 64x32 pixels** *xx* is the HWc number, *yy* is an index 0-2 and *string* is 1/3 of the image data encoded in base64 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. 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). SKAARHOJ has a helpful tool to convert images to 64x32 monochrome images: http://skaarhoj.com/FreeStuff/GraphicDisplayImageConverter.php Example:
ActivePanel=1	**Activates panel** Send this to activate the panel when “list” is received from the panel

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*=*string*	**Trigger action from hardware component** *xx* is the HWc number, *string* contains information about the trigger: *string* can have any of these forms: • “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.

Command	Description
map=zz:xx	**Local HWc to External HWc mapping information** *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
BSY	Busy message
RDY	Ready message
list	Initialization status request, return "ActivePanel=1<NL>"
ping	Keep-alive, return "ack<NL>"
_model	_model=SK_MODEL (future)
_topology	_topology=HTMLsvg (future)
_hwcs	_hwcs=jason with HWC data (future)

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.

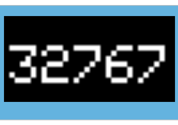
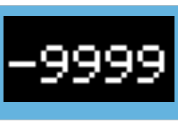
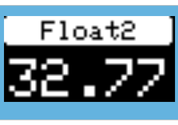
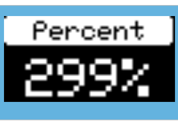
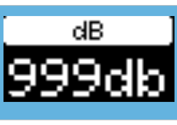
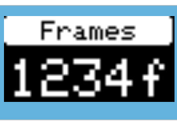
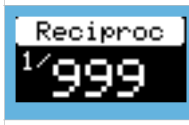
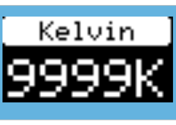
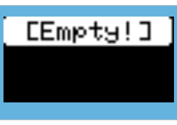
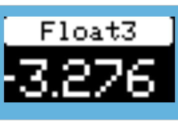
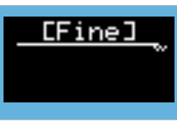
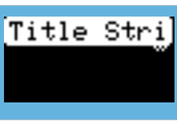
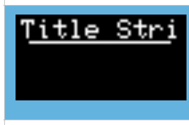
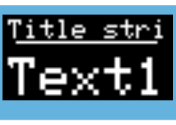
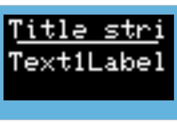
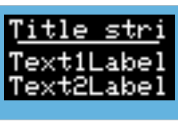
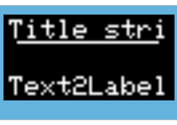
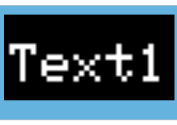
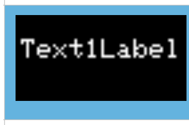
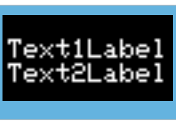
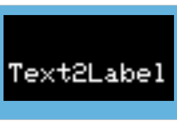
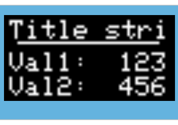
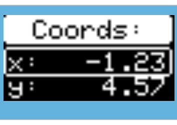
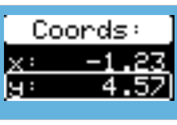
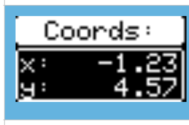
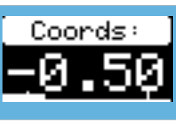
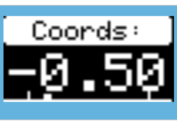
Future work

Further work is expected on the client, in particular ability to announce more details about itself, like model, 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 16 bit integer	*-9999* 16 bit integer, negative	*32767|1||Float2* Float with 2 decimal points	*299|2||Percent* Integer value in Percent	*999|3||dB* Integer value in dB	*1234|4||Frames* Integer in frames
					
999|5||Reciproc Reciprocal value of integer	*9999|6||Kelvin* Integer formated as Kelvin	*9999|7||[Empty!* format 7 = empty!	*-3276|8||Float3* Float with 3 decimal points, optimized 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 title string (and with "fine" indicator)
					
|||Title String|1 Title string as label (no "bar" in title)	*|||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|Text2Label* Printing two labels of 10 chars - automatically the size is reduced	*|||Title string|1||Text2Label* Printing only the second line - automatically 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 labels - automatically the size is reduced	*||||||Text2Label* Printing only the second line - automatically the size is reduced	*123|||Title string|1|Val1:|Val2:|456* First and second value is printed in small characters 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/ value 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 value	*-500|1||Coords:||||||2|-1000|1000|-700|700|2* A moving dot scale added below value			

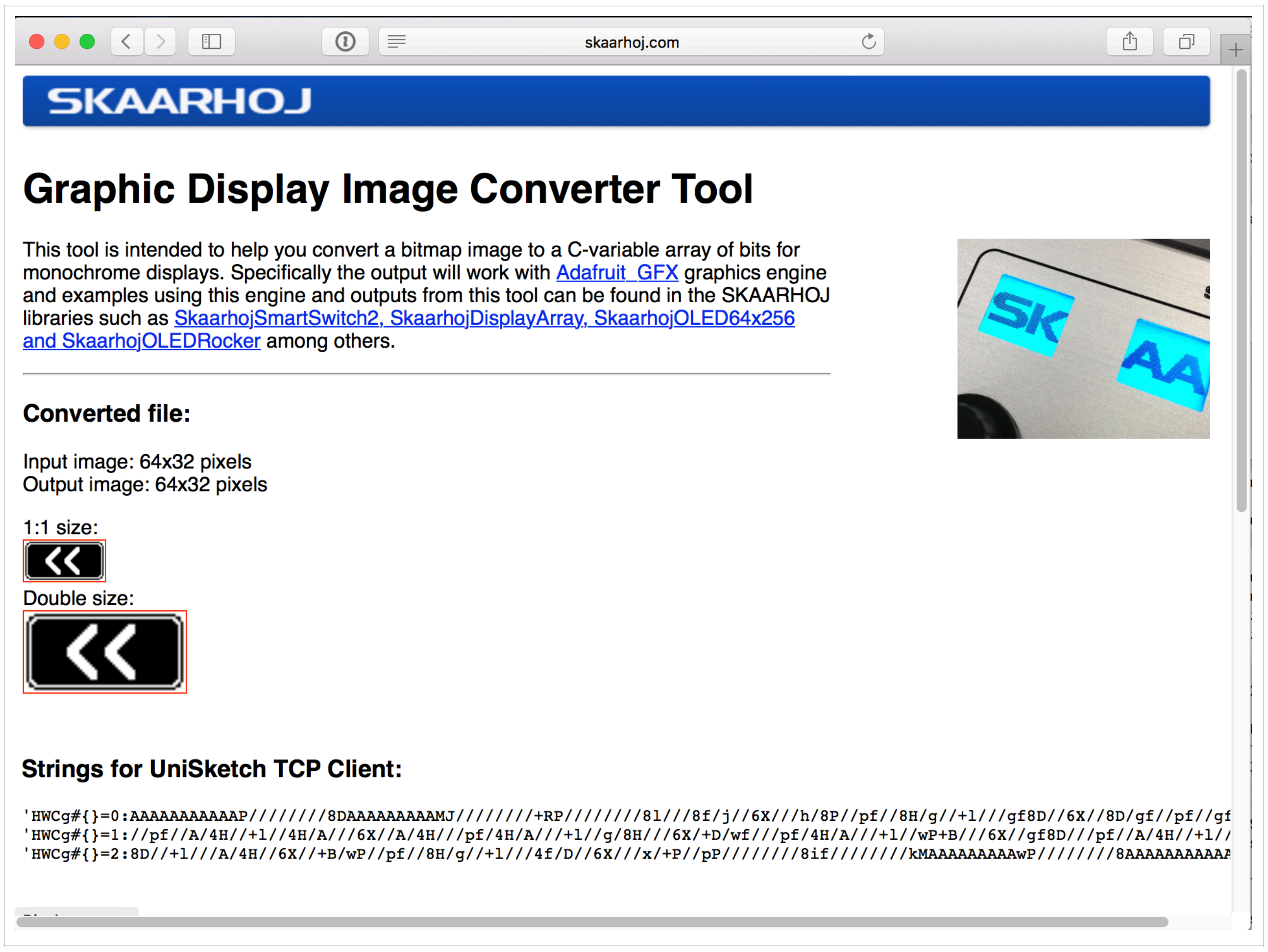
These graphics are generated from the test Python scripts. They can be very useful to experiment with other combinations.

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:


skaarhoj.com

SKAARHOJ

Graphic Display Image Converter Tool

This tool is intended to help you convert a bitmap image to a C-variable array of bits for monochrome displays. Specifically the output will work with Adafruit_GFX graphics engine and examples using this engine and outputs from this tool can be found in the SKAARHOJ libraries such as SkaarhojSmartSwitch2, SkaarhojDisplayArray, SkaarhojOLED64x256 and SkaarhojOLEDRocker among others.

Converted file:

Input image: 64x32 pixels
Output image: 64x32 pixels

1:1 size:

Double size:

Strings for UniSketch TCP Client:

```
'HWCg#{}=0:AAAAAAAAAAAP///////8DAAAAAAAAAAMJ///////+RP///////8l///8f/j//6X///h/8P//pf//8H/g//+l///gf8D//6X//8D/gf//pf//gf
'HWCg#{}=1://pf//A/4H//+l//4H/A///6X//A/4H///pf/4H/A///+l//g/8H///6X/+D/wf///pf/4H/A///+l//wP+B///6X//gf8D///pf//A/4H//+l//
'HWCg#{}=2:8D//+l///A/4H//6X//+B/wP//pf//8H/g//+l///4f/D//6X///x/+P//pP///////8if///////kMAAAAAAAAAAwP///////8AAAAAAAAAAA
```

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