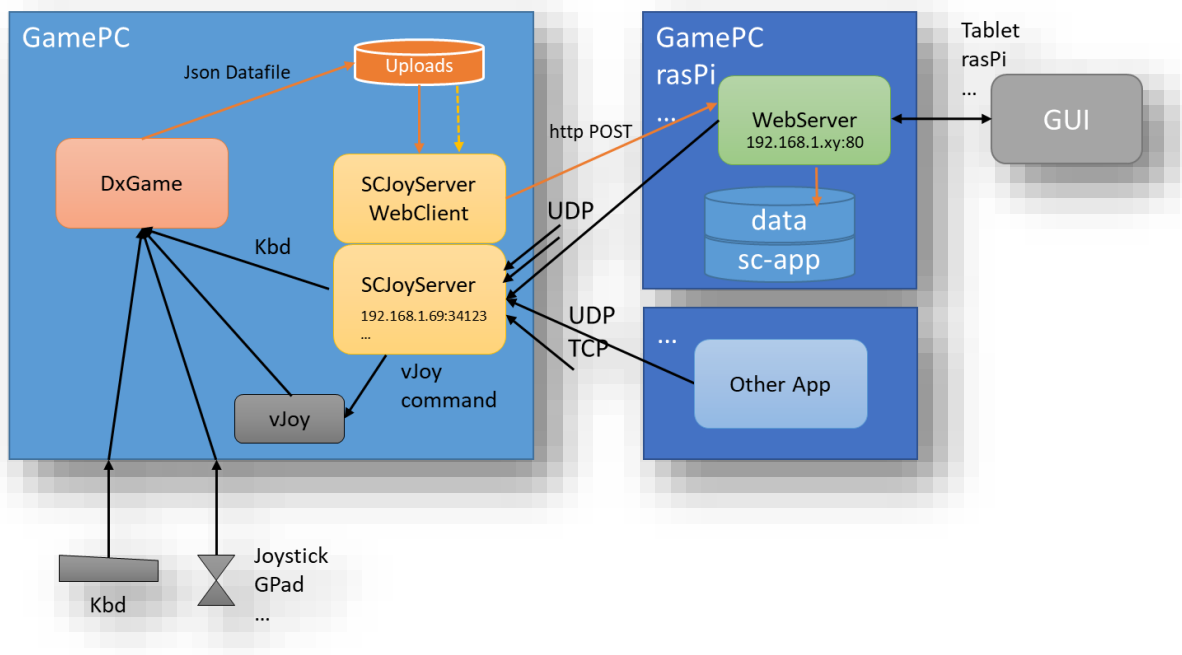


SC Remote Server – How To ...

V 2.0 20190815/Cassini

General Overview



There four Parts involved here:

- A game (application) that wants to get keyboard and/or Joystick commands to act on them. The game may optionally provide an interface to capture real-time data to show it on the remote device.
- The SCJoyServer which provides keyboard and optionally vJoy commands to the receiving application e.g. a game and waits for triggers to do it on a network address/port.
- An application such as a Web Server that hosts the sc-app and other game hosts (SCRemoteServer) which sends commands to trigger the SCJoyServer via an UDP protocol.
- An upload facility in SCJoyServer to upload game data in real-time to display it in the GUI

The SCJoyServer listens on its network address and port for incoming TCP or UDP connects.

Where UDB is just a one off message sent over the network that triggers a single command for the game. The Server supports more than one Joystick through multiple ports opened for receiving commands for the respective Joystick.

SCJoyServer can also monitor a local directory for files to be uploaded to the WebServer.

The WebServer serves a GUI for a client such as a tablet or other device with a screen and input means. The clicks, touches are then initiating the command sent via network to SCJoyServer and from there it is injecting vJoy or keyboard input for the game application.

If there is game data to display a Json data file can be uploaded to the WebServer in order to show real time data from it in the connected GUI.

While this doc describes how to setup the SCJoyServer and WebServer on the GamePC and how to create your own game host web app; it would be possible to trigger the SCJoyServer from further applications running on either the GamePC or preferably any other computer that is able to connect to the GamePC and equipped to send meaningful triggers to the SCJoyServer on the GamePC.

Infrastructure Setup

WebServer

Update V2.0 – rework and integrate web server

The package here contains a tiny web server built from Node.JS (a server side Javascript environment)

So there is no need to install and configure a 3rd party web server.

Install Node.JS on the computer that will serve the game hosts from (<https://nodejs.org/en/download/>)

Download and copy the SCRemoteServer package with all sub folders into a preferred folder.

Open e.g. PowerShell on Win, change to your install folder and issue:

```
$ npm install
```

This should load all needed modules and install them in a new folder `node_modules`.

The web server Port can be changed in `main.js` (default is :9042)

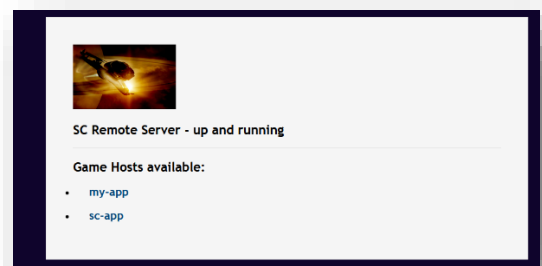
START the server from the shell by issuing:

```
$npm run start
```

Target your browser to `webServerIP:9042` (or the port you defined)

You find the default page that was supplied with the package.

Note: You may run the remote site on any computer able to install node.js e.g. also on a small rasPi.



SCJoyServer

Get the package here:

<https://github.com/SCToolsfactory/SCJoyServer/releases>

Extract into an empty directory on your Game PC.

Run `SCJoyServer.exe`

Configure your Game PC IP and the port to use (must be a free one)

IANA would suggest:

Dynamic and/or Private Ports are those from 49152 through 65535.

But for your in house use the default one is usually good enough.

You may see if a vJoy device is found – it is not required but then you can only use keyboard commands. Check the Joysticks to use if there are any.

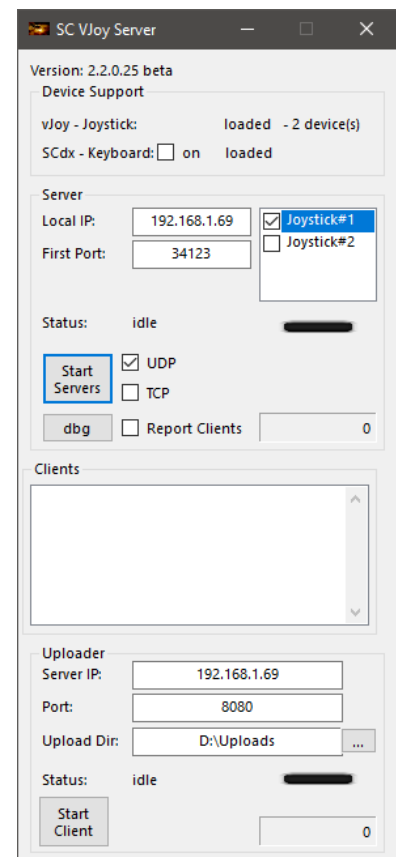
Start the Server once you want to receive commands.

→ Beware keystroke commands are supplied to your active window on that PC i.e. you may feed keys into the wrong application – some Window shortcuts such as Alt F4 (Close App) have unexpected results...

→ The program starts with keyboard input disabled (checkbox unchecked)

The Server reports commands received by incrementing the number at Report Clients.

`dbg` opens a window that may help to find issues.



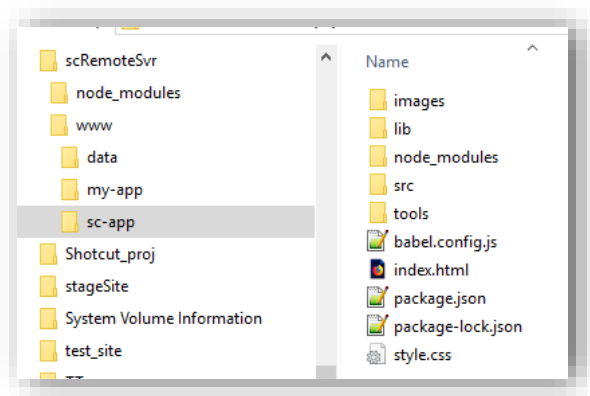
vJoy Driver

is located here:

<http://vjoystick.sourceforge.net/site/index.php/download-a-install/download>

Setup of the SCRemoteServer Web Site

You should find all of the provided demo game hosts: `sc-app` and `my-app` in the Web root directory (`www`), then also the demo display update content in `data` which uses all files to run the Display Demo.



Should then look as above.

Any directory added to `www` is exposed by the web server and can be accessed using:

`http://webServerIP:9042/directory/`

e.g. our new site as `http://webServerIP:9042/sc-app/`

Note `webServerIP` is something like: `192.168.1.68` i.e. the IP address of the machine where the web server is running.

If you now start the provided web server it will serve a tiny homepage as seen on page 2.

To test the site you may need to change the `SCJoyServer` address and port to your game PC.

Edits should be done in the `src` folder and in `pages.js` (unless you want extend features and functions)

Edit `pages.js` in the `src` folder with Notepad

Find:

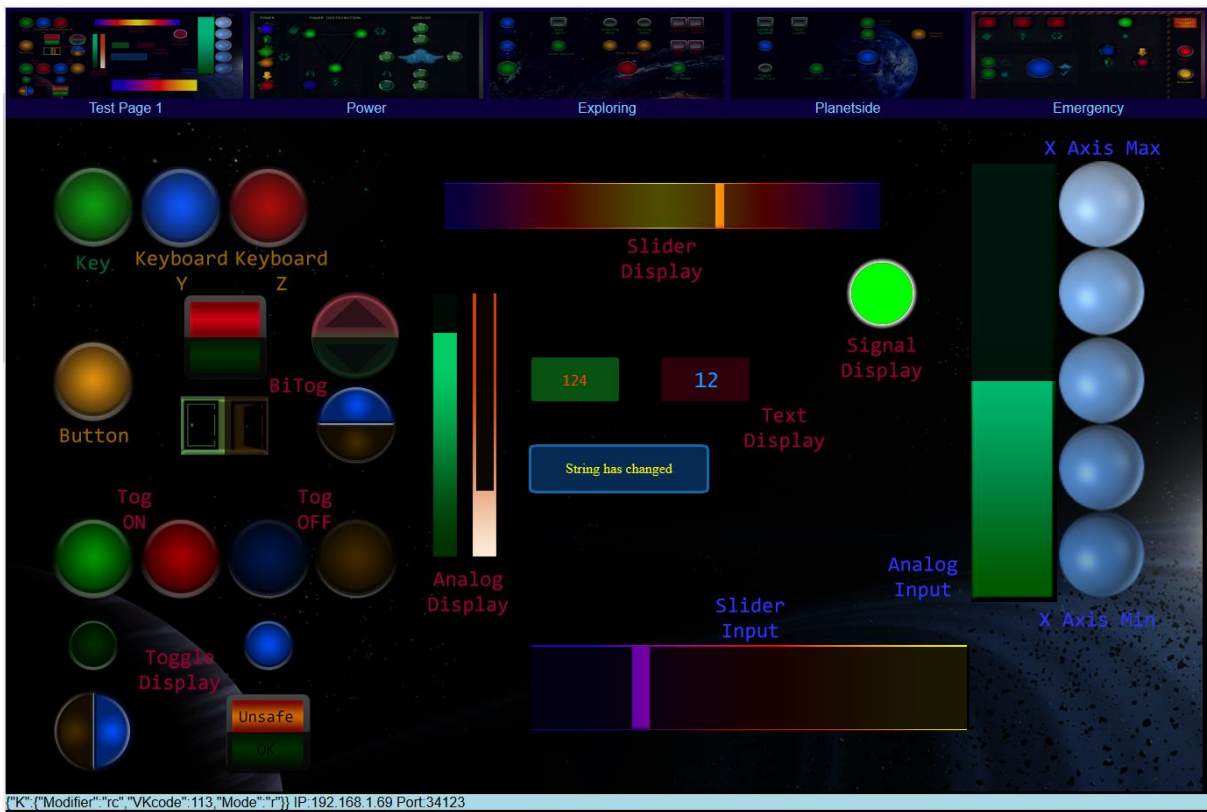
```
// the vJoy Command Server IP
Dom.IP = '192.168.1.69';
// the vJoy Command Server First Port number (UDP protocol)
Dom.PORT = 34123;
// data file to capture content from, located in and loaded from webRoot/data (empty
name disables polling)
Dom.DATAFILE = 'sc-app.json';
```

| Server | |
|-------------|--------------|
| Local IP: | 192.168.1.69 |
| First Port: | 34123 |

Change it to the values you set in the `SCJoyServer`.

The `DATAFILE` name fits the provided data files. See also Display Items section.

Now you should be able to navigate to `http://webServerIP:9042/sc-app/` and your browser should show the following (see next page)



The page should update the Display items randomly at a pace of about 2 seconds.

Click the Tabs (top row thumbnails) to change the page.

The DEBUG information is showing what command was issued.

If you click and hold the mouse on the buttons it should issue a Press command, Releasing issues the release to the key or button. i.e. it presses as long as you click or touch the item.

Clicking 'Key' shows:

```
{\"K\":{\"Modifier\":\"rc\",\"VKcode\":113,\"Mode\":\"r\"}} IP:192.168.1.69 Port:34123
```

You may see Mode: “p” while pressing and “r” once released

The Keycode was 113 – which is F2

Modifier is “rc” which is Right Ctrl – so the key sent was RCtrl+F2.

Now you are ready to play with your own customization..

Customization

The workflow is:

- Design one up to 5 pages (images) size is currently fixed to 1366x768 pixels
- Edit `pages.js` in the `src` folder to setup your hit targets and display items.

That's it.

The site consist of 5 pages (tabs) where the background image contains all drawn visuals.

For every page one has to define hit targets. Those serve as interaction spots where a click or touch is recognized and then the according command is sent to the SCjoyServer.

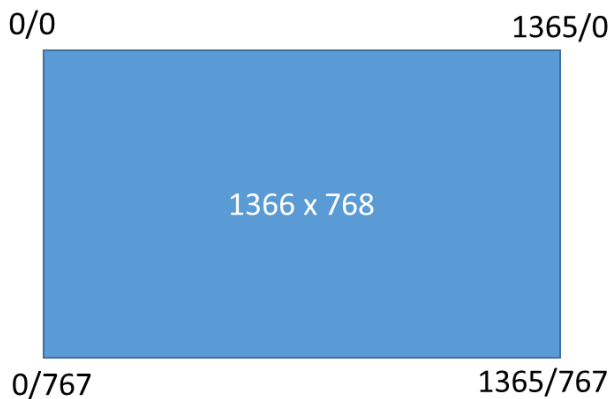
Targets can be of different modes and shapes. The target layer will respond with visual clues depending on the mode of activation. There is either a visual feedback on a hit, a toggle state or an analog slider setting.

Drawing background images

Format can be either PNG or JPG, Size is 1366x768 pixels.

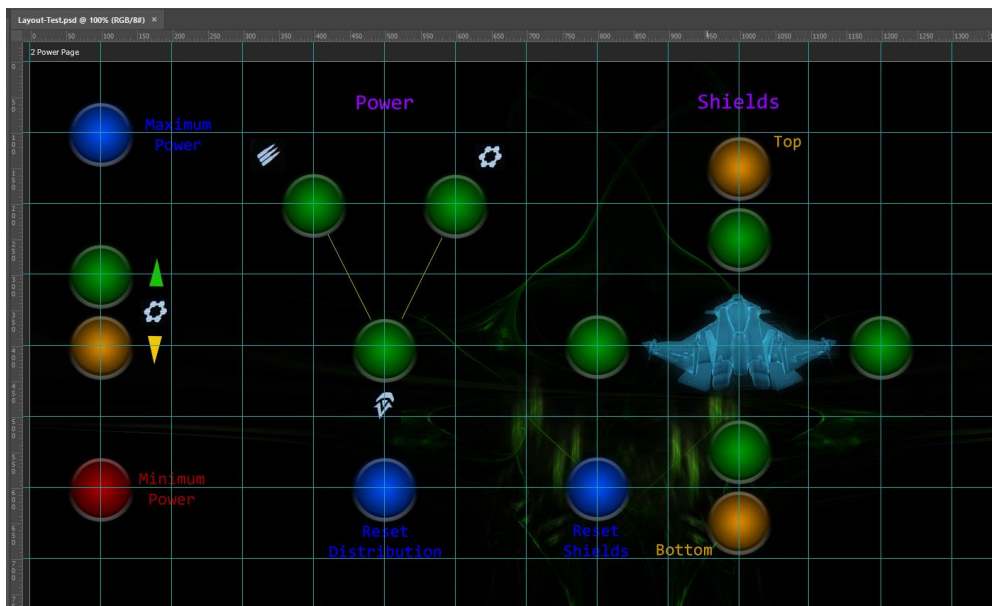
Images go into the `images` directory.

To identify hit targets you must record the X and Y pixel location and the size of the hit target. Locations are top – down and left right oriented. I.e. top left corner is 0/0.



The example buttons are about 100x100 pixels which is an acceptable touch target size on an iPad 2.

Using guides and place targets on the grid makes it rather easy – below a 100x100 grid is used.



Editing pages.js

- ➔ Use only notepad, MS Code or a similar code editor (never Wordpad or Word or any other word processor) - Make a backup copy before you edit...
- ➔ Once edited and saved you have to reload the page in your browser to make it active (F5)

The file consist of 5 similar parts for each of the 5 supported pages:

```
// BEGIN PAGE CONSTRUCTION
// overwrite the provided one with our own definitions
page_base.myPages_InitPages = function()
{
    // PAGE 1 Construction
    this.page 1 obj = new Page_proto_obj(
        "Test Page 1",
        "images/page_1.png",
        [
            // Hit / Touch Targets
            new Target("my1", 100, 100, 90, 0, Item.ModePR, new Cmd(Item.TypeKey, VK.F2, Item.KModRCtrl, 0, 0)),
            new Target("my2", 100, 300, 90, 0, Item.ModePR, new Cmd(Item.TypeButton, 3, Item.KModNone, 0, 0)),
            ...
            new Target("sl1", 850, 650, 500, 94, Item.ModeSlider, new Cmd( Item.TypeYaxis, 250, Item.KModNone, 0, 0))
        ],
        // Data Display Items
        [
            new Display("dItm1", 700, 400, 190, 90, Disp.ModeTxt, Disp.CenterAlign, Disp.Middle, Cust.Font16, Cust.D_Yellow, "section1", "item1"),
            new Display("dItm2", 650, 300, 100, 50, Disp.ModeTxt, Disp.CenterAlign, Disp.Middle, Cust.Font16m, Cust.D_Red, "section1", "item2"),
            ...
        ]
    );
};
```

The **page name** – it is also shown below the Tab for easy navigation.

The **image path and filename** – it refers to the image to load as background image.

You may change them according to your need and image file naming.

Then there are as many **Target definitions** as hit areas you want to use on that page.

Last are as many **Data Display definitions** as data areas you want to use on that page.

The Test Page has some buttons, toggles and axis parts for the hit targets.

For the data display items there are 9 on this Test Page.

Target Items

Name e.g. "my1" this must be a unique name within the page but serves only internal purposes.

Note: Using the same name across pages will maintain only one value state and therefore synchronize toggles, analog and slider controls when tabbing the pages.

X, Y, DW, H Center of the hit target (X/Y) and for circles diameter (DW) and height=0 (H), for rectangles use width (DW) and height (H) of the shape.

The hit area of my 100x100 buttons get a circle with a diameter of 90 pixels in order to show properly. Hit targets are marked with a semi-transparent circle that gets whiter when pressed (alpha 0.05 vs. 0.3 – to be changed in pages.js – function myPages_InitCust ()), or a black shape for toggle mode items (they black out if OFF).

Mode – the mode of activation (Press/Release, Toggle, Tap, Value, Analog, Slider)

Type – the type of control (key, button, axis, rotAxis, slider)

Value – a value or index related to the control

KMod –for keystrokes only a key modifier such as Left or Right - Ctrl, Alt, Shift

TogInit – defines the toggle init state (use 0 to start with Off)

JsIndex – Joystick index - selects the UDP channel for the SCvJoyServer

Data Display Items:

Name e.g. "my1" this must be a unique name within the page but serves only internal purposes.

X, Y, DW, H Center of the display item (X/Y) and for circles diameter (DW) and height=0 (H), for rectangles use width (DW) and height (H) of the shape.

Mode – the mode of display (Text, Toggle, Signal, Analog, Slider)

Alignment, Baseline and Font – for text display, you may use the labels provided in `const.js` and `custom.js`

Color – for text and signal display, you may use the `Cust.D_Color` labels provided in `custom.js`

Section and Item – defines the data source item in the json datafile, two strings to match the data item

Target Items for click and touch

Key and Button 'Press – Release' activation

```
new Target("my1", 100, 100, 90, 0, Item.ModePR, new Cmd(Item.TypeKey, VK.F2, Item.KModRCtrl, 0, 0)),  
new Target("my2", 100, 300, 90, 0, Item.ModePR, new Cmd(Item.TypeButton, 3, Item.KModNone, 0, 0)),
```

The **Mode** is `Item.ModePR` (press, release)

A **Type** whether a Key or a Button is triggered (`Item.TypeKey`, `Item.TypeButton` are valid here)

→ The key/button remains down as long as you click/touch.

The **Value** as Key Code or button index (1.. max button).

The last one is a **Key modifier** such as `RightCtrl` + Key.

For **keys** you may use the symbols defined in the file `command.js`. Those are `VK.something`. E.g. `VK.A` is an A pressed, `VK.D3` is the 3 key one on the main keyboard, 3 on the num pad would be `VK.NUMPAD3`.

Note: if you are using a non US (QWERTY) keyboard/language setting you are punished with the same issues that the game gets the input from the key location rather than the imprint on the key. I.e. QWERTZ keyboards issue a Y Key but the game sees it as Z input. Here you may use the same letter key assignment as on the keyboard imprint.

For special chars use the ones you really need e.g. `VK_SEMICOLON` issues really a semicolon.

You may need to check what arrives in e.g. notepad as receiving active window.

Valid key modifiers are:

```
Item.KModNone, Item.KModLCtrl, Item.KModRCtrl, Item.KModLAlt, Item.KModRAlt, Item.KModLShift, Item.KModRShift
```

Once edited and saved you have to reload the page in your browser to make it active (F5)

Key and Button 'Short Tap' activation

Sometimes you may want to issue just a short tap.

```
new Target("my4", 300, 100, 90, 0, Item.ModeTap, new Cmd(Item.TypeKey, VK.Z, Item.KModNone, 0, 0)),
```

The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Mode**, **Type**, Value, kMod

The **Mode** is then `Item.ModeTap` which equals a single short key or button strike independent of the length of the key/button click or touch.

This is for **Type** key or button, `Item.TypeKey`, `Item.TypeButton` are valid here.

Key and Button 'Toggle' activation

Toggles might get it sometimes wrong either because an input is lost by the game or you pressed the keyboard and the App would not know about that one.

Toggles described here are either showing or masking the defined area.

```
new Target("tg1", 100, 500, 90, 0, Item.ModeTog, new Cmd(Item.TypeKey, VK.V, Item.KModNone, 1, 0)),  
new Target("tg2", 200, 500, 90, 0, Item.ModeTog, new Cmd(Item.TypeButton, 4, Item.KModNone, 1, 0)),  
new Target("tg3", 300, 500, 90, 0, Item.ModeTog, new Cmd(Item.TypeKey, VK.W, Item.KModNone, 0, 0)),  
new Target("tg4", 400, 500, 90, 0, Item.ModeTog, new Cmd(Item.TypeButton, 5, Item.KModNone, 0, 0)),
```

The **Mode** is `Item.ModeTog` with **TogInit**=1 (ON at startup) or `Item.ModeTog` with **TogInit**=0 (OFF at startup).

A **Type** selector whether a Key or a Button is triggered (`Item.TypeKey`, `Item.TypeButton` are valid here)

→ It issues a single stroke when clicked/touched and changes the visual toggle state from one to the other.

The **Value** as Key Code or button index (1.. max button).

The last one is a **Key modifier** such as `RightCtrl` + Key.

Valid key modifiers are again:

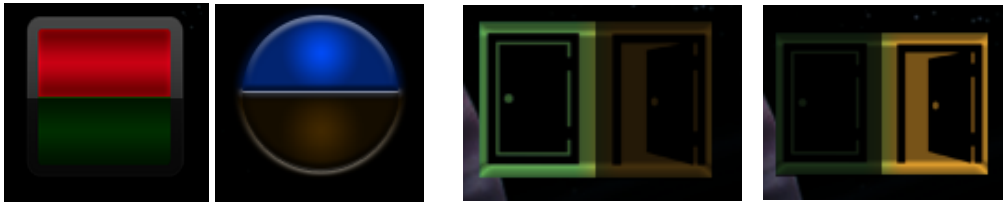
```
Item.KModNone, Item.KModLCtrl, Item.KModRCtrl, Item.KModLAlt, Item.KModRAlt, Item.KModLShift, Item.KModRShift
```

NOTE: if you use it on your PC where the `SCJoyServer` runs with Key sending enabled you just issue the keys into your active window – **the browser** – and you may see strange effects when using Function keys.. or other browser shortcuts. Alternatively use a tablet to test the web client and make the active window on your PC an application that does not bother with keys sent – e.g. an empty notepad window.

Key and Button 'BiColorToggle' activation

Toggles may also appear as 'BiColor' i.e. have two states visualized rather than only on and off as the above does. The hit target is the full defined area. BiColor Toggles are always OFF at startup.

BiColor toggles are round or rectangle target areas which are divided either horizontally or vertically in two parts. One half is masked and the other is fully visible, clicking will change the state and will therefore cover the formerly unmasked part where the masked one gets fully visible.



It can be used with colors and/or icons as shown above.

Off state will show the left or upper part (red, blue, closed door above). The yellow door is the toggled door state in the example above.

```
new Target("tg7", 400, 250, 100, 0, Item.ModeBiTogUD, new Cmd(Item.TypeButton, 8, Item.KModNone, 0, 0)),
new Target("tg8", 250, 350, 100, 70, Item.ModeBiTogLR, new Cmd(Item.TypeButton, 9, Item.KModNone, 0, 0)),
```

The **Mode** is either `Item.ModeBiTogUD` (up, down orientation) or `Item.ModeBiTogLR` (left, right orientation).

A **Type** selector whether a Key or a Button is triggered (`Item.TypeKey`, `Item.TypeButton` are valid here)

→ It issues a single stroke when clicked/touched and changes the visual toggle state from one to the other.

The **Value** as Key Code or button index (1.. max button).

The last one is a **Key modifier** such as `RightCtrl + Key`.

Valid key modifiers are again:

```
Item.KModNone, Item.KModLCtrl, Item.KModRCtrl, Item.KModLAlt, Item.KModRAlt, Item.KModLShift, Item.KModRShift
```

Axis and Slider activation by Value

You may also issue axis and slider actions with a value.

Value range is always 0...1000 (min ... max), 500 is center.

An X axis single click/touch command with a value of 500 (center point) looks like:

```
new Target("ax2", 1250, 300, 90, 0, Item.ModeVal, new Cmd(Item.TypeXaxis, 500, Item.KModNone, 0, 0)),
```

The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Type**, **Mode**, **Value**, kMod (where kMod is ignored)

The **Mode** is `Item.ModeVal` as you give a Value to set in the command.

The **Type** for the axis and slider are:

```
Item.TypeXaxis, Item.TypeYaxis, Item.TypeZaxis,
Item.TypeRXaxis, Item.TypeRYaxis, Item.TypeRZaxis,
Item.TypeSL1, Item.TypeSL2
```

Value is the set point 0...1000

You may use e.g. 5 targets to cover 0, 250, 500, 750, 1000 as in the example page 1.

Axis and Slider activation by analog or slider control

If you wish to use a slider with analog behavior use the following

```
new Target("a11", 1150, 300, 100, 500, Item.ModeAnalog, new Cmd(Item.TypeXaxis, 500, Item.KModNone, 0, 0)),
new Target("s11", 850, 650, 500, 94, Item.ModeSlider, new Cmd( Item.TypeYaxis, 250, Item.KModNone, 0, 0)),
```

The items here are:

Name, CenterX, CenterY, **Width, Height, Type, Mode, Value**, kMod (where kMod is ignored)

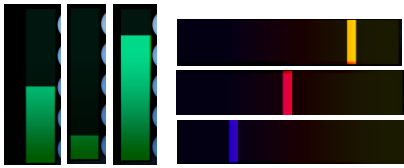
The **Mode** is Item.ModeAnalog or Item.ModeSlider. **Value** is the initial set point value 0..1000.

The **Type** for the axis and slider are:

```
Item.TypeXaxis, Item.TypeYaxis, Item.TypeZaxis,
Item.TypeRXaxis, Item.TypeRYaxis, Item.TypeRZaxis,
Item.TypeSL1, Item.TypeSL2
```

The orientation of the slider is determined by the larger extent of dw or h.

In the above example **dw**=100 and **h**=500 → **h** > **dw** defines a vertical orientation of the sliding effect.



Analog and Slider at different positions. Zero is left or bottom, max is right or top.

The analog/slider target should match the area that is to be hidden or revealed depending on the control set point. The effect covers and therefore hides the area right or above the current set point with a black mask of almost zero transparency. The color and alpha used for this effect can be changed in page.js - function myPages_InitCust() (A_SliderCol, A_Alpha).

Note: as there is no feedback from the receiving application the control just shows what you clicked or touched but not what the device setting is. I.e. two of the same controls e.g. Y-axis on the same or different tabs are not tracking each other.

Multi Joystick support

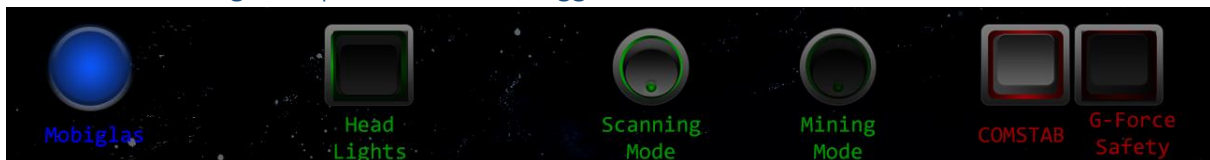
To use more than one Joystick use the optional parameter jsIndex when defining Targets.

```
new Target("my99", 100, 700, 90, 0, Item.ModePR, new Cmd(Item.TypeButton, 17, Item.KModNone, 0, 1)),
```

The last parameter is an index starting with 0 to whatever the SCvJoyServer is setup.

The default parameter 0 selects the first checked in the list, 1 the second checked etc.

Circle and Rectangle Shapes and PR and Toggle items



```
new Target("hli", 400, 100, 80, 75, Item.ModeTog, new Cmd(Item.TypeKey, VK.T, Item.KModLAlt, 0, 0)),
```

Above is the HeadLights target definition, a square with center of 400/100 and a size of 80x75.

For analog sliders only a rectangle is supported, it needs both width and height.

For On/Off toggles you may use red and green logic.

The green ones are meant to be OFF when dark i.e. Lights are off, Scanning Mode is ON, Mining OFF.

The red ones are meant to be OFF when light i.e. COMSTAB is off and G-Force is ON (hence dark because it's safe – red would be unsafe)

Using the same name across pages (e.g. "hli") will maintain one state and synch all Head-Lights toggle in all pages with the same toggle state. Valid for toggles, analog and slider targets.

→ But it is up to you to make this as helpful as possible...

Display Items

Display items show data from an uploaded file.

Data file format

The expected file format is Json with the following 2 level structure:

```
{
  "section":{"item":value ... }
  ...
}
```

e.g.

```
{
  "section1":{"item1":"UTF-8 实例 - may be ??", "item2": 130.1}
  , "section2":{"item33":"Other String", "item44": 42, "item55":true}
  , "section3":{"tog1":false, "tog2": true, "sig1":true, "alog1": 22, "alog2": 22, "sli1":22 }
}
```

Data reference in the Display item is then “section”, “item” to retrieve the data item.

Above ... , “section1”, “item2” would return the numeric value 130.1

Data types of items can be string, numeric value (NO quotes!!) or boolean (true, false NO quotes !!).

Section names, Item names and strings must be quoted.

Data File

The files are expected to be in the `www(root)/data` directory and of type json.

The data file to be used must be defined in `page.js` as:

```
// data file to capture content from, located in and loaded from webRoot/data (empty name disables polling)
Dom.DATAFILE = 'sc-app.json';
```

here `www/data/sc-app.json` is loaded to show display items.

There can be only one data file per site. You may however use different filenames for different sites.

If you provide an empty string then the data capture and display update is turned off (recommended if not used)

Data / GUI Refresh

As default the GUI is refreshed once per second. i.e. the data file is reloaded and all items are updated with the current values and the page is redrawn.

The update pace can be changed in `page.js myPages_InitCust` function. E.g. to slow down.

```
Cust.DisplayPerSec = 0.5; // slow down for Demo - 2sec update pace
```

Or to speed up 1..10.

The fastest one allowed is 10 (every 100msec) but in general 1..4 should be good enough to not overload the browser and the server.

Display Text

Text is shown as delivered by the file, UTF-8 is supported. The alignment is around the X,Y coordinate given and the text is boxed within the given boundaries and get squeezed if too long for the regular font spacing.

Text requires a rectangular box i.e. use width AND height.

```
new Display("dItm1", 700, 400, 190, 90, Disp.ModeTxt, Disp.CenterAlign, Disp.Middle, Cust.Font16, Cust.D_Yellow, "section1", "item1"),
```

The items here are as above:

Name, CenterX, CenterY, Width, Height, **Mode**, **Align**, **Baseline**, **Font**, **Color**, **Section**, **Item**

The **Mode** is `Disp.ModeTxt` to show a text.

Align is how the text is aligned with respect to X, you may use: `Disp.LeftAlign`, `Disp.CenterAlign`, `Disp.RightAlign` where Center uses X as center of the show text, Left, Right accordingly.

Baseline defines the vertical alignment with respect to Y, you may use: `Disp.Bottom`, `Disp.Middle`, `Disp.Top` where Middle uses Y as middle of the character box for this font, Top and Bottom accordingly.

Font defines the character font, use any CSS compliant font specification here, some prefabs are: `Cust.Font16`, `Cust.Font24`, `Cust.Font32` as Serif ones, `Cust.Font16s` would be SansSerif (Arial) and `Cust.Font16m` is MonoSpace. See `page_base.js` for the available ones. You may supply a string here with your own spec or substitute the prefaps in the `page.js myPages_InitCust` function with your preferred font specs and then use the prefabs.

Color defines the text color, use any CSS compliant color specification here, some prefabs are: `Cust.D_White`, `Cust.D_Black`, `Cust.D_Red`, `Cust.D_Blue`, `Cust.D_Green`, `Cust.D_Yellow`, `Cust.D_Orange`. Same here, substitute or define new ones in `page.js myPages_InitCust` function and use labels rather than strings as parameters.

Section is the data source section part (see above), a string.

Item is the data source item part (see above), a string.

Display Signals

Signals will show the background drawing under the defined shape (rectangle or circle) or cover it with a shape of defined color.

A Signal needs a data item of type Boolean (true, false) else it will fail.

Signals show the background as default (when false), true enables the color shape drawn.

```
// Signal - uses the given color as shape color (shown when true)
```

```
new Display("dItm14", 1000, 200, 70, 0, Disp.ModeSig, Gen.Dummy, Gen.Dummy, Gen.Dummy, Cust.D_Green, "section3", "sig1"),
```

The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Mode**, , , **Color** , **Section**, **Item**

The **Mode** is `Disp.ModeSig` to visualize a signal state.

Color defines the text color, use any CSS compliant color specification here, some prefabs are: `Cust.D_White`, `Cust.D_Black`, `Cust.D_Red`, `Cust.D_Blue`, `Cust.D_Green`, `Cust.D_Yellow`, `Cust.D_Orange`.

Section is the data source section part (see above), a string.

Item is the data source item part (see above), a string.

In general for the unused parameters use `Gen.Dummy` this is safe.


Display Toggles

Toggles will show and hide (mask) the background drawing under the defined shape (rectangle or circle).

A Toggle needs a data item of type Boolean (true, false) else it will fail.

Toggles are either default (when false) ON (background is shown) or OFF (background is masked)


```
// Toggles
new Display("dItm11", 100, 600, 50, 0, Disp.ModeTog, Gen.Dummy, Gen.Dummy, Gen.Dummy, 1, "section3", "tog1"),
new Display("dItm12", 300, 600, 50, 0, Disp.ModeTog, Gen.Dummy, Gen.Dummy, Gen.Dummy, 0, "section3", "tog1"),
```



The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Mode**, , , , **Section**, **Item**

The **Mode** is `Disp.ModeTog` with a **TogInit** value of either 0 (OFF) or 1 (ON) to visualize a toggle state.



Section is the data source section part (see above), a string.

Item is the data source item part (see above), a string.

Display "BiColor" Toggles

See BiColor Activation Toggles above for the behavior.

A BiColor Toggle needs a data item of type Boolean (true, false) else it will fail.

BiColor Toggles will show the upper/left part when OFF and the lower/right part when ON (true).

```
// Bi color Toggles
new Display("dItm21", 100, 700, 85, 0, Disp.ModeBiTogLR, Gen.Dummy, Gen.Dummy, Gen.Dummy, Gen.Dummy, "section3", "tog1"),
new Display("dItm22", 300, 700, 95, 95, Disp.ModeBiTogUD, Gen.Dummy, Gen.Dummy, Gen.Dummy, Gen.Dummy, "section3", "tog2"),
```

The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Mode**, , , , **Section**, **Item**

The **Mode** is `Disp.ModeBiTogUD` or `Disp.ModeBiTogLR` (LR=left-right) to visualize a toggle state.

Section is the data source section part (see above), a string.

Item is the data source item part (see above), a string.

Display Analog (Bars) and Sliders

Analog (Bars) and Sliders will show or hide the background drawing under the defined shape (rectangle only) or mask it with a black shape of some transparency.

Analog and Slider needs a data item of type Numeric Value else it will fail.

The values are expected to be 0... 100.0 and will be reduced to fit that range.

Analog and Slider Display assume 0 to the left or bottom of the longer part of the shape.

The Slider uncovers only the part around the value where Analog provides a bar type display.

```
// Analogs (Bars)
new Display("dItm15", 500, 350, 40, 300, Disp.ModeAnalog, Gen.Dummy, Gen.Dummy, Gen.Dummy, Gen.Dummy, "section3", "alog1"),
// Slider
new Display("dItm17", 750, 100, 500, 50, Disp.ModeSlider, Gen.Dummy, Gen.Dummy, Gen.Dummy, Gen.Dummy, "section3", "sli1"),
```

The items here are as above:

Name, CenterX, CenterY, DiameterOrWidth, Height, **Mode**, , , , **Section**, **Item**

The **Mode** is `Disp.ModeAnalog` or `Disp.ModeSlider` to visualize a signal state.

Section is the data source section part (see above), a string.

Item is the data source item part (see above), a string.

Macro Items

New in V2.0

You may define and refer to Macro items when creating a Tap target. -> see `my-app/src/pages.js` for an example use.

Macros are lists of commands issued one after the other when the Target is tapped. So there is no feedback or other mean possible from macro command execution but sometime it is handy to issue a handful of keystrokes or setting the value of multiple axis with one tap.

```
// BEGIN MACRO DEFINITION
// overwrite the (empty) Init function with our own definitions
// Macros can be used as Activation Type in pages for Item.ModeTap only
page_base.myPages_InitMacros = function()
{
    var mac = null;

    // macros to choose Speed AND Accel Presets in page 1
    mac = new Macro("m_CfgFPwr", [ // set Speed to 100 and Accel to 100%
        new Cmd(Item.TypeRXaxis, Cust.A_Scale, Gen.Dummy, 0, 0), // Speed Sel 100%
        new Cmd(Item.TypeRYaxis, Cust.A_Scale, Gen.Dummy, 0, 0) // Accel Sel 100%
    ]);
    Macro.AddMacro(mac); // save it

    mac = new Macro("m_CfgCruise", [ // set Speed to SCM and Accel to 50%
        new Cmd(Item.TypeRXaxis, Cust.A_Scale/2, Gen.Dummy, 0, 0), // Speed Sel SCM = 50%
        new Cmd(Item.TypeRYaxis, Cust.A_Scale/2, Gen.Dummy, 0, 0) // Accel Sel 50%
    ]);
    Macro.AddMacro(mac); // save it
    ..
}
// END MACRO DEFINITION
```

In order to use macros one has to define them first.

The last section of the `pages.js` file will carry it.

A macro is a list of `Cmd()` items used exactly the same way as in Targets above – description of parameters can be seen above too.

In the example above there are two macros defined.

One has a name of `"m_CfgFPwr"` the second `"m_CfgCruise"`

`m_CfgFPwr` issues an Analog command for the rotaryX axis and sets it to a value of `Cust.A_Scale` (which is actually the max value of 1000 but using the label you must not know this number) as second action in the macro the rotaryY axis is set also to max (the effect in the SC game would be to put Speed and Accel to max with one touch)

`m_CfgCruise` does a similar thing but sets Speed and Accel in game to 50% (max / 2)

To use the macros we do in the page definition something like:

```
// PAGE 1 Construction
this.page_1_obj = new Page_proto_obj(
    'Flight',
    'images/page_1.png',
    // Target Items
    [
        new Target("FPwr", 99, 148, 76, 0, Item.ModeTap, new Cmd(Item.TypeMacro, "m_CfgFPwr", Gen.Dummy, 0, 0)),
        new Target("Cruise", 99, 249, 76, 0, Item.ModeTap, new Cmd(Item.TypeMacro, "m_CfgCruise", Gen.Dummy, 0, 0)),
    ]
);
```

For macro use you can only use `Item.ModeTap` as interaction behavior.

Then the command **Type** is `Item.TypeMacro` and the **Value** is the macro name as string e.g. `"m_CfgFPwr"` as defined in the macro section above.

Hitting this target (Full Power) will then set Speed AND Accel to 100%. (if this is mapped in game to rotX and rotY)

Note: As all Commands are synched a Slider that acts on the same axis will get its visual update within the page and across pages.