

Panel Builder Scripting Manual

Version 1.0

Table of Contents

A.	Introduction p.2
B.	Controllers p.3

SVSi 2014

A. Introduction

The Panel Builder scripting engine now supports Javascript syntax for writing functional code and executing commands. To use Javascript in an SVSi button script or in the project's setup script (which can be accessed from the project panel), simply use a script open tag, '<script>', to begin writing Javascript and a '</script>' tag to close Javascript execution.

A sample script using Javascript syntax could look like the following:

```
myVariable = '127.0.0.1';
myFunction(myVariable);
```

All standard Javascript syntax can be utilized inside of script tags, and all variables defined inside the script tags are assigned to the global scope. A number of helpful functions are also included in the global Javascript scope including the following:

nCmd: a function to send a TCP message to another machine,
ex: nCmd("tcpclient 192.168.1.1 50001 "+asciiToHex("getStatus"));

nCmdAsync: a function to send a TCP message to another machine asynchronously,
ex: nCmdAsync("tcpclient 192.168.1.1 50001 "+asciiToHex("getStatus"),
"myResponse", function(){ doThisOnComplete(); });

require: a function that is passed the string name of the filepath of a Controller,
ex: require("js/controllers/SVSi/N_Series.js");

project: an object that represents the current Panel Builder project and its panels,
ex: project.currentpanel.buttons[0].element.style.backgroundColor = 'black';

panel: an object that represents the currently visible panel,
ex: panel.element.style.backgroundImage = 'url(http://mysite/myimage.jpg)';

buttons: an array holding all the buttons on the currently visible panel,
ex: buttons[1].textnode.innerHTML = "The text on the button";

createElement: a function that allows creation of DOM nodes like document.createElement,
ex: myImage = createElement("img");

gotopanel: a function that moves to a new panel,
ex: gotopanel(2);

responses: an object where network responses from nCmd may be stored,
ex: console.log(responses["myResponse"]);

tweencss: a function to animate CSS transformations and other values,
ex: tweencss({object: buttons[5].element, time: 0.9, props: {rotationZ: 0}, to:
{rotationZ: 1.57}});

getButton: pass this function the id of a button and it will return the button object,
ex: mybutton = getButton('Button-1');

asciiToHex: a function that converts an ASCII string to a HEX string,
ex: asciiToHex("sometext");

hexToAscii: a function that converts a HEX string to an ASCII string,
ex: HexToAscii("fe43b204");

subPanel: given the index of a panel, this creates a sub-panel of it on the current panel,
ex: subPanel(2, width, height, left, top);

self: an object representing the button or widget that is calling the script
ex: self.element.style.width = '120px';

B. Controllers

Controllers are prewritten functions that can be called in the Javascript areas provided they be linked to first by a require call in the project setup script like so:

```
require('js/controllers/SVSi/N_Series.js');
```

There are helpful dropdown menu options in the script editor for adding require statements to your setup script which will list all the available Controllers by their manufacturer and fill in the necessary file link for you.

Here is a simple example of what would be in the N_Series Controller Javascript file that would be included if the above require function call was added to your project setup script:

```
function N_Series(ip, port, onError){

    var obj = {};

    obj.ip = ip;
    obj.port = !undef(port) ? port : '50001';

    obj.getStatus = function(async){

        if (async)
            nCmdAsync('tcpclient '+obj.ip+' '+obj.port+' '+asciiToHex('getStatus'),
                function(data){
                    obj.status = data;
                });
        else obj.status =
            nCmd('tcpclient '+obj.ip+' '+obj.port+' '+asciiToHex('getStatus'));

    }

    obj.switch = function(stream){
        nCmd('tcpclient '+obj._ip+' '+asciiToHex('set:'+stream));
    }

    obj.print = function(){
        console.log(obj);
    }

    return obj;

}

if (!Controllers.N_Series) Controllers.N_Series = N_Series;
```

This function takes an IP address of a machine and a port number and sets up a Javascript

object for interacting with that machine. After the Javascript file is included by the require function, the object could then be created and used like so in a script on a button or in the project setup script:

```
myN_SeriesObj = N_Series('192.168.1.1', '50001');  
myN_SeriesObj.switch(251);
```

Normally you would declare the object in the project setup script and then only call its functions in a button's script.