Introduction

The **Panel Builder** scripting engine now supports JavaScript syntax for writing functional code and executing commands.

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

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

All standard JavaScript syntax can be used and all defined variables are assigned to the global scope as long as you do not prefix them with the var declaration (i.e., `var k = 0;`). The var declaration in JavaScript is used to attach a variable to a local scope and that variable will no longer be available after that function or script ends/returns. A number of helpful functions are also included in the global JavaScript scope (see the following table):

Option	Description
Global Objects	
project	Object that represents the current **Panel Builder** project and its panels.
panel	Object that represents the currently visible panel.
self	Object representing the button or widget that is calling the script.
buttons	Array holding all of the buttons on the currently visible panel.
widgets	Array holding all of the widgets on the currently visible panel.
Server	An object containing information about the server like `Server.name` and `Server.ip`. Optionally server warnings, errors, and notices in the console can be turned off by setting `Server.errors = false`, etc.
Client	An object containing information about the client.
$/jQuery	A jQuery object.
Global Functions	
noop	An empty function that does nothing. Can be used in the place of a callback.
sleep	Pauses execution for specified amount of milliseconds.
setTimeout	Allows execution of function after specified amount of milliseconds.
setInterval	Allows execution of function in an interval of specified amount of milliseconds.
asciiToHex	Function used to convert an ASCII string to a HEX string.
hexToAscii	Function used to convert a HEX string to an ASCII string.
get	Pass this function the ID of a button, widget, or panel and it returns the object.
require	Function that is passed the string name of the filepath of a Module.
gotopanel	Function that moves to a new panel.
tcpclient	Function to send TCP message to a client.
udpclient	Function to send UDP message to a client.
telnetclient	Function to send a telnet message to a client.
int	Function to convert a string or number to an integer.
float	Function to convert a string or number to a float value.
string	Function to convert an object or number to a string.

Option	Description
time	Function to return UNIX time in seconds.
millitime	Function to return UNIX time in milliseconds.
openLink	Function to open a browser tab or window to a network or internet address.
setServerVar	Function to save a variable to the server.
getServerVar	Function to retrieve a variable from the server.
setProjectVar	Function to save a variable to the project's directory on the server.
getProjectVar	Function to retrieve a variable from the project's directory on the server.
tweencss	Function used to animate CSS transformations and other values.
createElement	Function that allows creation of DOM nodes (e.g., `document.createElement`).
nCmd	Function used to send a TCP message to another machine.
nCmdAsync	Function used to send a TCP message to another machine asynchronously.
ajaxCmd	Function used to send scripting messages through AJAX.
wsCmd	Function used to send scripting messages through the websocket.

Modules

Modules are pre-written functions that can be called in the JavaScript areas provided they are linked to first by a require call in the project setup script (as shown below):

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

There are helpful drop-down menu options in the script editor for adding require statements to your setup script which will list all the available modules 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 Module JavaScript file that would be included if the above require function call was added to your project setup script:

```
modules.SVSi.N_Series = function (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.setStream = function(stream){
    nCmd('tcpclient '+obj._ip+' '+asciiToHex('set:'+stream));
    }
    obj.print = function(){
    console.log(obj);
    }
    return obj;
}
```

This function takes an IP address of a machine and a port number and sets up a JavaScript object for interacting with that machine. As shown below, after the JavaScript file is included by the require function, the object could then be created and used in a script on a button or in the project setup script:

```
myN_SeriesObj = SVSi.N_Series('192.0.0.1', '50010');
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.

3000 RESEARCH DRIVE, RICHARDSON, TX 75082 AMX.com | 800.222.0193 | 469.624.8000 | +1.469.624.7400 | fax 469.624.7153
AMX (UK) LTD, AMX by HARMAN - Unit C, Auster Road, Clifton Moor, York, YO30 4GD United Kingdom • +44 1904-343-100 • www.amx.com/eu/