uAutomate Server
Data Acquisition Web Server
Client Developer's Manual

Version: v1.0.0.0

HTML, JavaScript, and Ajax to Serial Port Web Server and Gateway

Developed by: Boden Automation, LLC

Table of Contents

1. Introduction 3
2. Including the JavaScript API 4
   2.1. Relative Reference to API on Server 4
   2.2. Absolute Referencing to API located on Server 4
   2.3. Relative Referencing to the API Copied Locally 4
   2.4. Absolute Referencing to the API Copied Locally 4
3. Calling API Functions 5
   3.1. Setting TCP Address of Server 5
   3.2. Login to the Server 5
   3.3. Creating the SerialPort Object on the Server 6
   3.4. Bringing it all Together 6
4. HTML Events 8
   4.1. onload 8
   4.2. setTimeout and Timers 8
5. uAutomateServer.js Reference 9
6. SerialPort Class Reference 15
7. Use in HTA Files 16
8. Samples 17
9. Table Of Figures 21
10. Revision History 22

1. Introduction

uAutomate Server comes with a small JavaScript API to allow HTML Pages that support standard JavaScript and AJAX to communicate with the server and computer’s serial ports. By including this .js file the client developer can be sending and receiving serial port data in a matter of minutes.

Cross Browser Support for accessing uAutomate Server and Serial Ports. Web Browsers viewing client pages served up from uAutomate Server will have access to the same functions the local computer would have. This allows you to write pages that can be viewed by compatible web browsers on other computers, Android devices/ tablets, and Iphone. This can give you access to the computer's COM Ports from these remote devices. This software package has not been tested on every version of web browser that is in the market. If the browser you wish to use supports scripting along with AJAX or remote HTTP requests then this package will work for you.

2. Including the JavaScript API

In order to use the API in your code, you will need to include it in your header of the HTML page.

The name of the js file is uAutomateServer.js and can be linked to in a few ways.

2.1.Relative Reference to API on Server

If your web page is served up from the server itself, then you can use relative referencing as shown below. This is the best way to accomplish the linking. This will work no matter where the pages are served up from.

```
<head>
  <script src="/Scripts/uAutomateServer.js" type="text/javascript"></script>
</head>
```

2.2.Absolute Referencing to API located on Server

Absolute referencing can primarily come in useful if you have created a local HTML file and need to point to the remote server file.

```
<head>
  <script src="http://127.0.0.1:4260/Scripts/uAutomateServer.js"
type="text/javascript"></script>
</head>
```

2.3.Relative Referencing to the API Copied Locally

The uAutomateServer.js file can be copied locally and used from within your local file system. For simplicity the js file should be copied into the same folder as your HTML file. If the js file is in the same folder as the HTML file then you would reference it as below.

```
<head>
  <script src="uAutomateServer.js" type="text/javascript"></script>
</head>
```

2.4.Absolute Referencing to the API Copied Locally

The uAutomateServer.js file can be copied locally and used from within your local file system. If copied locally, you can link to the file by using the entire path of the js file. Below is a sample of how the client can link to the js file if it is in the default install location.

```
<head>
  <script src=”file:/// C:\Program Files
(x86)\uAutomate\uAutomateServer\System\Scripts\uAutomateServer.js"
type="text/javascript"></script>
</head>
```

3. Calling API Functions

Now that you have properly linked to the uAutomateServer.js file, you can start to call functions that are defined inside of it. The js file is not very large and can be opened by your HTML/JS editor and examined. The help reference for the API is included with this manual.

To access functions there is a uAutomateServer JavaScript Class built within the js file. Below is a sample of how the developer can create an instance of this class

```
var SerialPort;

SerialPort = new uAutomateServer();
```

Now that you have an instance of the class stored in your newly created variable “SerialPort”, you can begin to call properties and methods of that class. Below is an example of calling a uAutomateServer function.

```
SerialPort.Call("PortName = 'COM1'");
```

The two functions that get most frequently used in the uAutomateServer class are the Call and Get functions. “Call” will call a function or set a value without returning a value. “Get” will return a value.

```
SerialPort.Call("PortName = 'COM1'");
s = SerialPort.Get("ReadAllLines()");
```

3.1.Setting TCP Address of Server

In order to properly communicate with the uAutomate server you will need to set the TCP address. TCP_Address is a function that is part of the uAutomateServer class. This function will allow you to set the TCP/IP address of the server.

Below is an example of how to call the TCP_Address function.

```
SerialPort.TCP_Address("127.0.0.1");
```

***NOTE: You only need to set the TCP Address of the server IF you are NOT serving up your html files from the server. By default if the TCP Address has NOT been set the API will default to using the server TCP address of where the files were served from. This means that if you are using uAutomate Server to host your html files then the API will reference back to that server when making calls back and forth.

3.2.Login to the Server

Another vital step in calling commands from the API is the need to login to the server. If user ids and passwords have been setup on the server then you will need to call the “Login” function of the uAutomateServer class. This function requires a user id and password to be passed to it. Upon successful login, the function will return a KeyCode. This KeyCode must be passed to the “KeyCode” function to be stored for future calls into the uAutomate Server. If the client is not properly logged in, errors will appear on subsequent calls to the API.

```
KeyCode = SerialPort.Login("admin", "admin");
SerialPort.KeyCode(KeyCode);
```

It is possible that in the Settings.ini file of the server that LocalAccessGranted could be enabled. If local access is granted and you are calling to and from the local server “127.0.0.1”, then you will not need to login.

3.3.Creating the SerialPort Object on the Server

The last step if the process before calling functions is to create the remote object on the server. The function to use is “CreateRemoteTag”. Calling this function will add an object running on the server to receive and process the commands you send to it. Below is a sample of how to call this function.

```
SerialPort.CreateRemoteTag("COM1", "uAutomateServer.SerialPort");
```

3.4.Bringing it all Together

Below is a sample of what it looks like when all of the previous setup steps are complete.

```
<head>
  <script src="/Scripts/uAutomateServer.js" type="text/javascript"></script>
</head>

<script type="text/jscript">
    var SerialPort;
    var SerialPortOpen = false;
    var KeyCode;

    function Loaded() {

        try {

            SerialPort = new uAutomateServer();

            //SerialPort.TCP_Address("127.0.0.1");

            KeyCode = SerialPort.Login("admin", "admin");
            SerialPort.KeyCode(KeyCode);

            SerialPort.CreateRemoteTag("COM1", "uAutomateServer.SerialPort");

            SerialPort.Call("PortName = 'COM1'");
            SerialPort.Call("BaudRate = 38400");
            SerialPort.Call("DataBits = 8");
            SerialPort.Call("StopBits = 1");
            SerialPort.Call("Parity = 0");
            SerialPort.Call("Handshake = 0");
            SerialPort.Call("ReadTimeout = 250");
            SerialPort.Call("Open()");
            SerialPortOpen = true;
        }
        catch (err) {
            SetError(err);
        }
        finally {
        }
    }
</script>
```

4. HTML Events

This section covers some very useful events that can be used from your HTML pages in the web browser.

4.1.onload

The Body Element's onload event gets trigger upon loading of the HTML page and is very useful for setting up initial conditions and getting things going. This event is where you can configure your SerialPort and get the timer events initiated.

Below is a sample how you can structure your code to handle the onload event.

```
<body onload="return Loaded();">

<script type="text/jscript">

    function Loaded() {


    }
</script>
```

4.2.setTimeout and Timers

The setTimeout function can be used to create a timer type of event. This can be used to update the screen every xx milliseconds or to send a set point out to a device after an interval has passed. The two parameters for setTimeout are a pointer to the function to be called and how long to wait in milliseconds before calling the function. More information can be found on the operation of this function online. The setTimeout function will only trigger the event once. If you wish for the event to trigger every xx milliseconds, you will need to recall the function from within your event function that you called.

Below is a sample of how this mechanism can work. This sample will call the UpdateSend function every 1 second.

```
<body onload="return Loaded();">

<script type="text/jscript">

    function Loaded() {
        setTimeout(UpdateSend, 1000);
    }


    function UpdateSend() {
        setTimeout(UpdateSend, 1000);
    }
</script>

</body>
```

5. uAutomateServer.js Reference

A reference guild has been created for the functions contained within the uAutomateServer.js file. To view this reference follow the link below.

uAutomateServer.js Reference\index.html

Class uAutomateServer

Defined in: uAutomateServer.js.

Class Summary	
	uAutomateServer() uAutomateServer v1.0.0.0 | (c) 2014 Boden Automation, LLC.

Method Summary	
	Call(Command, callback) Calls a function or set property on the remote server Tag and returns the value.
	CodeType(val) Code syntax type of the commands being sent to the uAutomateServer.
	CommandTime() Time in milliseconds that it takes for the execution and return of the command.
	CreateRemoteTag(Tag, RemoteType) Creates the object with Tag on uAutomateServer.
	ExecuteMethod(Code) Executes code on the uAutomateServer.
	ExecuteMethodAsync(Code, CallBack) Executes code on the uAutomateServer Asynchronously.
	Get(Command, callback) Calls a function or property on the remote server Tag and returns the value.
	KeyCode(val) Security code returned from the uAutomateServer after logging in.
	Login(UserName, Password) Function to login the user the the uAutomateServer.

Method Summary

	Tag(val) Remote Tag name to create for this link on the remote uAutomateServer.
	TCP_Address(val) TCP Address of the uAutomateServer.
	TCP_Port(val) TCP Port used by the uAutomateServer.
	TCP_TimeOut(val) TCP Timeout is how long to wait for responses from the uAutomateServer.

Class Detail

uAutomateServer()

uAutomateServer v1.0.0.0 | (c) 2014 Boden Automation, LLC. uAutomateServer is a JavaScript used to connect and communicate with uAutomateServer.exe web service.

Method Detail

{String} **Call**(Command, callback)

Calls a function or set property on the remote server Tag and returns the value.

Parameters:

{string} **Command**

- Code to call on the remote server.

{callback} **callback**

- The callback function to call upon completion.

Returns:

{String} Value returned by remote code call.

{String} **CodeType**(val)

Code syntax type of the commands being sent to the uAutomateServer.

Parameters:

{string} **val** *Optional, Default: PYTHON*

- Code Type PYTHON, VB.NET. Default is PYTHON.

Returns:

{String} Code Type PYTHON, VB.NET. Default is PYTHON.

`{number}` **`CommandTime`**`()`

Time in milliseconds that it takes for the execution and return of the command.

Returns:

{number} Execution time in milliseconds.

`{String}` **`CreateRemoteTag`**`(Tag, RemoteType)`

Creates the object with Tag on uAutomateServer.

Parameters:

{string} **Tag**

- Remote Tag name to create for this link on the remote uAutomateServer.

{string} **RemoteType**

- Remote Tag type to create for this link on the remote uAutomateServer.

Returns:

{String} Remote Tag name to create for this link on the remote uAutomateServer.

`{String}` **`ExecuteMethod`**`(Code)`

Executes code on the uAutomateServer.

Parameters:

{string} **Code**

- Code to run on the uAutomateServer.

Returns:

{String} Result - The results of the code execution. To return a value the code must set a Value= variable.

`{String}` **`ExecuteMethodAsync`**`(Code, CallBack)`

Executes code on the uAutomateServer Asynchronously.

Parameters:

{string} **Code**

- Code to run on the uAutomateServer.

{CallBack} **CallBack**

- Callback function to call upon completion.

Returns:

{String} Result - The results of the code execution. To return a value the code must set a Value= variable.

`{String} Get(Command, callback)`

Calls a function or property on the remote server Tag and returns the value.

Parameters:

{string} **Command**

- Code to call on the remote server.

{callback} **callback**

- The callback function to call upon completion.

Returns:

{String} Value returned by remote code call.

`<inner> getText(el)`

Parameters:

el

`{String} KeyCode(val)`

Security code returned from the uAutomateServer after logging in. This code is used by the uAutomateServer to give access to the commands sent from this script.

Parameters:

{string} **val** *Optional*

- Code returned from the uAutomateServer.

Returns:

{String} Code returned from the uAutomateServer.

`{String} Login(UserName, Password)`

Function to login the user the the uAutomateServer.

Parameters:

{string} **UserName**

- >Username to login as.

{string} **Password**

- Password for the specified Username.

Returns:

{String} Result - Key used by preceding functions to allow interaction with uAutomateServer.

`{string}` **`Tag`**`(val)`

Remote Tag name to create for this link on the remote uAutomateServer.

Parameters:

{string} **val** *Optional*

- Tag name.

Returns:

{string} Tag name.

`{String}` **`TCP_Address`**`(val)`

TCP Address of the uAutomateServer. If left blank will use same domain as served from.

Parameters:

{string} **val** *Optional*

- The TCP Adress fo the uAutomateServer.

Returns:

{String} The TCP Adress fo the uAutomateServer.

`{String}` **`TCP_Port`**`(val)`

TCP Port used by the uAutomateServer.

Parameters:

{string} **val** *Optional*

- The TCP Port fo the uAutomateServer. Default is 4260.

Returns:

{String} The TCP Port fo the uAutomateServer. Default is 4260.

`{number}` **`TCP_TimeOut`**`(val)`

TCP Timeout is how long to wait for responses from the uAutomateServer.

Parameters:

{number} **val** *Optional*

- The TCP TimeOut fo the uAutomateServer. Default is 8000.

Returns:

{number} The TCP TimeOut fo the uAutomateServer. Default is 8000.

Documentation generated by JsDoc Toolkit 2.4.0 on Sat Sep 20 2014 15:40:47 GMT-0500 (CDT)

6. SerialPort Class Reference

The SerialPort class that is created in the uAutomateServer, and can be linked to from your client, is the same System.IO.Ports.SerialPort object built into the .NET framework. If you wish to understand more about the .NET SerialPort class you can review the MSDN document included with this manual at SerialPort.pdf.

Seeing as how the functions and properties of the SerialPort class are being called remotely and from various scripting languages, not all functions are supported. Because this is coming from a remote client Events of the SerialPort object are not handled at all. Below is the list of Properties and Methods supported by uAutomate Server's implementation of this class.

Properties

Name	Description
BaudRate	Gets or sets the serial baud rate.
BreakState	Gets or sets the break signal state.
BytesToRead	Gets the number of bytes of data in the receive buffer.
BytesToWrite	Gets the number of bytes of data in the send buffer.
CDHolding	Gets the state of the Carrier Detect line for the port.
CtsHolding	Gets the state of the Clear-to-Send line.
DataBits	Gets or sets the standard length of data bits per byte.
DiscardNull	Gets or sets a value indicating whether null bytes are ignored when transmitted between the port and the receive buffer.
DsrHolding	Gets the state of the Data Set Ready (DSR) signal.
DtrEnable	Gets or sets a value that enables the Data Terminal Ready (DTR) signal during serial communication.
Handshake	Gets or sets the handshaking protocol for serial port transmission of data.
IsOpen	Gets a value indicating the open or closed status of the SerialPort object.
NewLine	Gets or sets the value used to interpret the end of a call to the ReadLine and WriteLine methods.
Parity	Gets or sets the parity-checking protocol.
ParityReplace	Gets or sets the byte that replaces invalid bytes in a data stream when a parity error occurs.
PortName	Gets or sets the port for communications, including but not limited to all available COM ports.
ReadBufferSize	Gets or sets the size of the SerialPort input buffer.
ReadTimeout	Gets or sets the number of milliseconds before a time-out occurs when a read operation does not finish.
ReceivedBytesThreshold	Gets or sets the number of bytes in the internal input buffer before a DataReceived event occurs.
RtsEnable	Gets or sets a value indicating whether the Request to Send (RTS) signal is enabled during serial communication.
StopBits	Gets or sets the standard number of stopbits per byte.
WriteBufferSize	Gets or sets the size of the serial port output buffer.
WriteTimeout	Gets or sets the number of milliseconds before a time-out occurs when a write operation does not finish.

Methods

Name	Description
Close	Closes the port connection, sets the IsOpen property to **false**, and disposes of the internal Stream object.
DiscardInBuffer	Discards data from the serial driver's receive buffer.
DiscardOutBuffer	Discards data from the serial driver's transmit buffer.
GetPortNames	Gets an array of serial port names for the current computer.
Open	Opens a new serial port connection.
ReadByte	Synchronously reads one byte from the SerialPort input buffer.
ReadChar	Synchronously reads one character from the SerialPort input buffer.
ReadExisting	Reads all immediately available bytes, based on the encoding, in both the stream and the input buffer of the SerialPort object.
ReadLine	Reads up to the NewLine value in the input buffer.
ReadTo	Reads a string up to the specified *value* in the input buffer.
Write(String)	Writes the specified string to the serial port.
Write(Byte[], Int32, Int32)	Writes a specified number of bytes to the serial port using data from a buffer.
Write(Char[], Int32, Int32)	Writes a specified number of characters to the serial port using data from a buffer.
WriteLine	Writes the specified string and the NewLine value to the output buffer.

7. Use in HTA Files

HTA files are local HTML files that can be used on a local computer with no web server serving up the files. This can be useful for writing custom forms and scripts where you want to bypass some of the security restrictions that are enforced on pages coming from a remote tcp address. If an HTML file is named with the extension HTA, it can be run on a local computer by double clicking it. After activating the HTA file, it will by default be loaded into a trimmed down web browser HTA viewer window.

When using local HTA files, you will need to include the uAutomateServer.js file in a manner that works. This would be one of the absolutes including referencing described in an earlier chapter.

8. Samples

Samples demonstrating how to develop a HTML and JavaScript client are installed along with the uAutomate Server. These samples can be found in the uAutomate support folder which is by default installed in the "C:\uAutomateFiles\Samples\Serial" folder. Along with samples on how to send and receive serial data there is also included is a sample of a HTML and JavaScript based HyperTerminal type application.

By default the samples are installed in the "C:\uAutomateFiles\Samples\Serial" folder. These can be accessed through the web server at the address http://127.0.0.1:4260/Samples/Serial/.

Samples included:

http://127.0.0.1:4260/Samples/Serial/sendreceive.htm a simple send and receive between two COM ports.

http://127.0.0.1:4260/Samples/Serial/SendReceive%20Request.htm a simple send and receive between two COM ports where one COM port waits for a request then returns data back to the other COM port.

http://127.0.0.1:4260/Samples/Serial/Terminal.htm a simple web based HyperTerminal type application.

Below is a fully functional sample of a send and receive application between two COM Ports (COM1 and COM2). Sent and Received data is displayed in a HTML Table.

```
<!doctype html>

<html lang="en">

<head>
  <title>TCP Serial Send Receive Example</title>
  <meta charset="utf-8">
  <link rel="image icon" href="/Icon.ico"/>
  <link rel="shortcut icon" href="/Icon.ico"/>
  <script src="/Scripts/uAutomateServer.js" type="text/javascript"></script>
</head>

<body onload="return Loaded();">

<table border=1>
    <tr>
        <td>Send <input type="text" id="Send" rows=1 />Rec<input type="text" id="Rec"
rows=1 /></td>

    </tr>


    <tr>
        <td style="vertical-align: top;">
            Errors:<textarea id="Error" cols="30" rows="8"></textarea>
        </td>
    </tr>
</table>


<script type="text/jscript">
    var SerialPort1;
    var SerialPort2;
    var SerialPortOpen1 = false;
    var SerialPortOpen2 = false;
    var KeyCode;
    var UpdateInterval = 50;
    var OutputValue;
    var Async = true;

    function Loaded() {

        try {
            document.getElementById("Error").innerHTML = "";

            SerialPort1 = new uAutomateServer();
            SerialPort2 = new uAutomateServer();

            //SerialPort1.TCP_Address("127.0.0.1");
            //SerialPort2.TCP_Address("127.0.0.1");

            KeyCode = SerialPort1.Login("admin", "admin");
            SerialPort1.KeyCode(KeyCode);
            SerialPort2.KeyCode(KeyCode);

            SerialPort1.CreateRemoteTag("COM1", "uAutomateServer.SerialPort");
            SerialPort2.CreateRemoteTag("COM2", "uAutomateServer.SerialPort");

            SerialPort1.Call("PortName = 'COM1'");
            SerialPort1.Call("BaudRate = 38400");
            SerialPort1.Call("DataBits = 8");
            SerialPort1.Call("StopBits = 1");
            SerialPort1.Call("Parity = 0");
            SerialPort1.Call("Handshake = 0");
            SerialPort1.Call("ReadTimeout = 250");
            SerialPort1.Call("Open()");
            SerialPortOpen1 = true;

            SerialPort2.Call("PortName = 'COM2'");
            SerialPort2.Call("BaudRate = 38400");
            SerialPort2.Call("DataBits = 8");
            SerialPort2.Call("StopBits = 1");
            SerialPort2.Call("Parity = 0");
            SerialPort2.Call("Handshake = 0");
            SerialPort2.Call("ReadTimeout = 250");
            SerialPort2.Call("Open()");
            SerialPortOpen2 = true;
        }
        catch (err) {
            SetError(err);
        }
        finally {
        }
        OutputValue = 0;
        setTimeout(UpdateSend, UpdateInterval);
        setTimeout(UpdateReceive, UpdateInterval);
    }


    function DisConnect() {
        try {
            document.getElementById("Error").innerHTML = "";
            SerialPort1.Call("Close()");
            SerialPort2.Call("Close()");
        }
        catch (err) {
            SetError(err);
        }
        finally {
        }
    }

    function UpdateSend() {
        try {
            if (SerialPortOpen1 == true) {
                document.getElementById("Error").innerHTML = "";

                OutputValue = OutputValue + 1;
                if (OutputValue > 10) {
                    OutputValue = 0;
                }
                document.getElementById("Send").value = OutputValue;

                if (Async) {
                    SerialPort1.Call("Write('" + OutputValue + "')", SendCallBack);
                } else {
                    SerialPort1.Call("Write('" + OutputValue + "')");
                }
            }
        }
        catch (err) {
            SetError(err);
            SerialPortOpen1 = false;
        }
        finally {
        }

        setTimeout(UpdateSend, UpdateInterval);
    }


    function UpdateReceive() {

        if (SerialPortOpen2 == true) {
            try {
                if (Async) {
                    SerialPort2.Get("ReadAllLines()", ReceiveCallBack);

                } else {
                    var el;
                    el = document.getElementById("Rec");
                    var s;
                    s = SerialPort2.Get("ReadAllLines()");
                    s = s.replace(/"/g, '');
                    el.value = s;
                }
            }
            catch (err) {
                SetError(err);
                SerialPortOpen2 = false;
            }
        }

        setTimeout(UpdateReceive, UpdateInterval);
    }

    function SendCallBack(s) {
    }

    function ReceiveCallBack(s) {
        var el;
        el = document.getElementById("Rec");

        try {
            s = s.replace(/"/g, '');
            el.value = s;
        }
        catch (err) {
            SetError(err);
        }
    }


    function SetError(e) {
        var s;
        var el;
        el = document.getElementById("Error");
        s = e + el.value;
        el.value = s;
    }

</script>



</body>

</html>
```

9. Table Of Figures

No table of figures entries found.

10. Revision History

Date	Version	Description	Function
9/1/2014	1.0.0.0	Initial Release	NA