uAutomate Server
Data Acquisition Web Server

Version: v1.0.0.0

Running Scripts on uAutomateServer Using HTTP Commands and XML

Developed by: Boden Automation, LLC

09/01/2014

Table of Contents

1. Introduction

uAutomateServer is a light weight Web Server developed to serve as a gateway between various software platforms, data acquisition devices and communications protocols.

Many applications do not have the ability to communicate with certain devices such as COM ports or USB devices etc… uAutomate Server was designed to bridge that gap and allow these applications to work with these devices using more common protocols such as AJAX.

The underlying theory behind the uAutomate Server is that it consists of a script evaluator. Script commands are created on the client applications then sent over to the server to process and return values if applicable. A collection of different data acquisition devices or protocols are exposed to the scripts using a "Tags" collection. So scripts written can now properly access these devices and protocols.

The main reason for developing this application, and the perfect example of how it is intended to be used was the need to be able to read/write to an RS232 Serial port from HTML and Javascript.

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.

This document describes the underlying scripting support built into the uAutomate server and how to access the uAutomate Web server using HTML URLs or XML/Soap Commands using AJAX.

2. uAutomate Server Scripting

The underlying theory behind the uAutomate Server is that it consists of a script evaluator. Script commands are created on the client applications then sent over to the server to process and to return values if applicable. A collection of different data acquisition devices or protocols are exposed to the scripts using a “Tags” collection. So scripts written properly can now access these devices and protocols.

This section describes the different scripting languages and syntax along with how to call into the “Tags” collection of objects and devices.

** NOTE: In order for script commands to be executed or evaluated on the server the current user must be logged in or have allow local access turned on. See logging in section on how to log into the server before sending commands.

2.1.Scripting Languages

The uAutomate Server can support a few different scripting languages to allow for easier use for different programming languages. Seeing as how the script commands are very short and simple there is not much difference between the commands that would be sent. Some reasons to use one language over another would be speed of the script evaluator, for example the IronPython interpreter is 50 times faster than the VB.NET evaluator.

2.1.1. IronPython

Open source version of Python that can be implemented into .NET languages. This scripting language is very fast.

Samples:

Tags[‘MyComPort’]. PortName = 'COM1'

Tags[‘MyComPort’]. BaudRate = 38400

Value = 1 + 3 * 5

See Also: http://ironpython.net/

2.1.2. VB.NET

This is actual VB.NET code snippets. This scripting language is very slow.

Samples:

Tags(“MyComPort’”). PortName = “COM1”

Tags(“MyComPort’”). BaudRate = 38400

Value = 1 + 3 * 5

See Also: http://msdn.microsoft.com/en-us/library/aa712050(v=vs.71).aspx

2.2.Executing Scripts - Tags.Command Function

All Scripts are carried out by calling the Tags.Command function on the uAutomate server. This function will execute and evaluate the scripts on the server and return values if needed.

The tags collection object has a built in "Command" function that is used to execute the client scripts on the server. Please understand that this function is not called directly from your client code. Each client type has its own mechanism form calling into the Command function. For example HTML or AJAX calls the Command.htm file with either URL parameters OR XML posted data.

Not only do you send the scripts you wish to execute but you will also pass the proper user Keycode that was generated for security.

To understand how to execute scripts on the server please read the later section on HTML and AJAX commands.

2.3.Tags Collection

The Tags collection is how the server shares and exposes the different types of data acquisition to the scripts being processed. The collection has some default objects built into such as the Login object. To call methods, functions, and set/get properties of an object in the Tags collection just follow the samples below.

IronPython:

```
Tags['MyComPort']. PortName = 'COM1'
```

VB.NET:

```
Tags("MyComPort"). PortName = "COM1"
```

In the above sample "MyComPort" is the name of the object that you added to the collection. PortName is a property of the underlying object.

2.4.Tags.AddObjectByTypeName

Before objects can be used in the collection they first need to be added to the collection if they are not by default part of the collection. To add an object to the collection you must use the AddObjectByTypeName function built into the Tags object. Because you can have many of them SerialPort objects are not by default part of Tags collection. Below is a sample of how you can add a "MyComPort" tag to the Tags collection and have its base object be a SerialPort Class.

IronPython:

```
Tags.AddObjectByTypeName(' MyComPort ', 'uAutomateServer.SerialPort ')

Tags['MyComPort']. PortName = 'COM1'
```

VB.NET:

Tags.AddObjectByTypeName(”MyComPort “, “uAutomateServer.SerialPort “)

Tags(“MyComPort”). PortName = “COM1”

You can see in the above samples that once the tags are created you can access the underlying objects functions, methods, properties, etc…

2.5.Returning Values

Scripts can return values back to the client. To do this all you need to do is have your script execute its code but return the data in a Value variable. Once the script is evaluated the server will read the value that is stored in the “Value” property and return that back to the client. You can see how data is returned to the client with XML in a later chapter in this manual.

Below is an example of filling in the Value variable with data.

```
Value = Tags[' MyComPort’].ReadAllLines()
```

2.6.Logging In and User Access

In order for any commands to be executed on the server the client must be logged in properly. Once logged in the server sends back a KeyCode for the client to use with subsequent calls to the server to allow access. Once the client has the KeyCode they can retain that and use repeatedly as needed. If the client loses the KeyCode they can simply log back in again with the same user id and password to get the KeyCode resent.

The only way around the use of a password when sending commands to the server is to set the “LocalAccessGranted” equal to 1 in the Settings.ini file. This then allows only local users on the same computer to access the uAutomate Server.

To login to the server the client must call the Login function from the Login object that is part of the Tags collection. The syntax is as follows.

```
KeyCode = Tags['Login'].Login('userid’,'password')"
```

‘Login’ is the Login object in the tags collection, ‘userid’ is the first parameter passed and ‘password’ is the second parameter passed. Upon successful login the KeyCode is returned to be used in other locations in your client code.

By default there is a default admin userid and password of userid=admin and password=admin.

3. HTTP Commands and Scripts as Part of URL – Command.htm

It is not possible to call the Command function directly, but one way to call it is to make a call to the Command.htm file. When the server sees a request for the Command.htm file it looks for parameters being passed to it using the "?" and "&" symbols.

Below is an example of calling a script on the web server by passing parameters to the Command.htm file. With the local uAutomate Server running from your favorite browser enter the below in the navigation bar.

http://127.0.0.1:4260/Command.htm?Code=Value = 1 + 2

http://127.0.0.1:4260/Command.htm?Code=Value = 1 + 2&CodeType=Python

The returned data from this command will show up as XML in the browser.

```
<?xml version="1.0"?>

<Return>

    <Status>success</Status>

    <Value>3</Value>

</Return>
```

Parameters that can be passed to the Command.htm file are…

Code: Script code to execute.

CodeType (Optional): Type of scripting language being used. Python for IronPython or VB.NET for VB.NET. Default value is Python.

KeyCode (Optional): Keycode returned from a previous call to Tags['Login'].Login(userid,password). This parameter is only optional if "LocalAccessGranted" is set to 1 in the Settings.ini file and it is a local client.

4. Sending Script Commands Using XML / SOAP and AJAX

The best way to send script commands to the uAutomate server is to pass some simple XML formatted data to the server using AJAX. The server file to post the data to is still the Command.htm file but instead the client must POST data XML data.

URL = http://127.0.0.1:4260/Command.htm

The structure of the XML data is below.

```
<Call>
    <KeyCode>'KeyCode '</KeyCode>
    <Code>' Code'</Code>
    <CodeType>'CodeType '</CodeType>
</Call>
```

Parameters that can be passed to the Command.htm file from within this XML structure are…

Code: Script code to execute.

CodeType (Optional): Type of scripting language being used. Python for IronPython or VB.NET for VB.NET. Default value is Python.

KeyCode (Optional): Keycode returned from a previous call to Tags['Login'].Login(userid,password). This parameter is only optional if "LocalAccessGranted" is set to 1 in the Settings.ini file and it is a local client.

Sample XLM Content: The below sample will duplicate what is done in the url sample in the previous topic.

```
<Call>
    <KeyCode>''</KeyCode>
    <Code>'Value = 1 + 2'</Code>
    <CodeType>'Python '</CodeType>
</Call>
```

Below is Sample JavaScript with AJAX to pass the data and return the value.

```
    var xmlHttp = false;
    /* IE7, Firefox, Safari, Opera...  */
    if (!xmlHttp) try { xmlHttp = new XMLHttpRequest(); } catch (e) { xmlHttp = false; }
    /* IE6 */
    if (typeof ActiveXObject != "undefined") {
        if (!xmlHttp) try { xmlHttp = new ActiveXObject("MSXML2.XMLHTTP"); } catch (e)
{ xmlHttp = false; }
```

```
        if (!xmlHttp) try { xmlHttp = new ActiveXObject("Microsoft.XMLHTTP"); } catch (e)
{ xmlHttp = false; }
    }

        var Message = '<Call><KeyCode>'Value = 1 +
2'</KeyCode><Code>'Python'</Code><CodeType>''</CodeType></Call>';

        try {
                xmlHttp.open("POST", "http://127.0.0.1:4260/Command.htm", false);
        } catch (err) {
            throw "On open tcp Error." + err;
        }

        var err;
            try {
                err = "";
                xmlHttp.send(Message);
                break;
            } catch (err) {
                alert("On send tcp Error." + err + Message);
            }

        var ret;
        ret = xmlHttp.responseText;

        var xmldoc;
        xmldoc = xmlHttp.responseXML;

        var Status = xmldoc.getElementsByTagName("Status")[0];
        if (Status == null)
            Status = "";

        if (Status.toUpperCase() == "ERROR") {
            var e;
            e = xmldoc.getElementsByTagName("StatusCode")[0];
            e = e + ":" + xmldoc.getElementsByTagName("StatusDescription")[0];
            e = e + ":" + xmldoc.getElementsByTagName("Code")[0];
            throw e;
        } else {
            ret = xmldoc.getElementsByTagName("Value")[0];
        }
```

5. Returned Status and Values – XML Structure

The uAutomate server returns data to the client in XML format. Depending on the result of the script evaluation there are two versions of XML data that can get returned.

On successful completion on script the server will return the following XML.

```
<Return>
        <Status>" success"</Status>
        <Value>"Return Value of Script if applicable"</Value>
</Return>
```

If script does not execute successfully the server will return the following XML.

```
<Return>
        <Status>"error"</Status>
        <StatusCode>"Error Number"</StatusCode>
        <StatusDescription>"Error  Message"</StatusDescription>
        <Code>"Code"</Code>
        <CodeType>"CodeType"</CodeType>
</Return>
```

Returned Values XML Tags

Status: Status of the script execution will be either “success” or “error”.

Value: Upon successful execution of script on server the server will return the result of the equation in this tag. In order to have a script return a value you will need to use the syntax described above where you set the Value variable equal to the value you want returned, for example Value = 1 + 2.

StatusCode: If error occurs while executing the script the server will return a unique value in this number.

StatusDescription: If error occurs while executing the script the server will return the error description in this tag.

Code: If error occurs while executing the script the server will return the original script that caused the error in this tag.

CodeType: If error occurs while executing the script the server will return the CodeType that was passed with the code to be executed.

6. Table Of Figures

No table of figures entries found.

7. Revision History

Date	Version	Description	Function
5/25/2013	1.0.0.0	Initial Release	NA