iTach Flex HTTP API Specification
Version 1.1

INTRODUCTION

The iTach Flex is capable of using a newly implemented RESTful HTTP API (REST, or Representational State Transfer, refers to the style of programming specifically designed for distributed systems and used in web APIs). The REST interface allows for web based control implementations built on the HTTP standard and allows for stateless communications with the iTach Flex (no persistent socket connection needs to be opened). To accomplish this, unit resources have been designated URL based addresses, and all payload data is serialized into JSON (JavaScript Object Notation).

Below is the HTTP request used to request the version of the target unit:

```
HTTP/1.1 GET
<Base URL>/version
        where:
                <Base URL> is "http://<Unit IP Address>/api/v1"
```

In this string, the "api" tag indicates the request is a command rather than a request for a file (such as index.htm). The packet below is returned in response to the above request:

```
HTTP/1.1 (200 OK)
{
        "make":"GlobalCache",
        "model":"iTachFlexWiFi",
        "host":"Flex",
        "firmwareVersion":"710-2000-01",
        "userName":"",
        "password":""
}
```

NETWORK RELATED POLLS

Once communication with the device type is known, you can apply or request network related settings via the "network" namespace as shown below. The following is the response for a WiFi model iTach Flex:

```
HTTP/1.1 GET
<Base URL>/network

{

        "dhcp":"<DHCP Value>",
        "gateway":"<Gateway IP>",
        "ipAddress":"<IP Address>",
        "subnetMask":"<Mask Address>",
        "primaryDNSServer":"<Primary DNS>",
        "secondaryDNSServer":"<Secondary DNS>",
        "netBIOSName":"<NetBios String>",
        "networkType":"<Network Type>",
```


160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

```
    "ssid":"<Flex SSID>",
    "securityMode":"<Security Type>",
    "securityKey":"<Passphrase>"
}
```

where:

<DHCP Value> can be true or false
<Gateway IP>, <IP Address>, <Mask Address>, <Primary DNS> and <Secondary DNS> are IP Addresses
<NetBios String> is "ITACHFLEX" plus the last six characters of the unit's MAC address by default; limited to 16 characters
<Network Type> is 1 for infrastructure and 2 for adhoc
<Flex SSID> is limited to 32 characters
<Security Type> can be one of the following values:

0 // Open Security
1 // WEP
2 // WPA
3 // WPA2

<Passphrase> is the security key for the current SSID

To update the network settings, simply issue an HTTP PUT, rather than an HTTP GET passing the modified JSON Array in the payload of the HTTP Request.

While setting up a unit you can request all SSIDs available to the unit. This provides the ability for a user interface to list all available SSIDs and enable users to pick from the list with the request below:

```
HTTP/1.1 GET
<Base URL>/network/ssids
```

Sent in response to the above request:

```
HTTP/1.1 (200 OK)
[
    {
        "ssid":"ExampleNetwork",
        "networkType":"<TypeValue>",
        "config":"<ConfigValue>",
        "channel":"<WirelessChannel>"
    },
    {...}
]
```

where:

<TypeValue> is 0 for adhoc, 1 for infrastructure
<WirelessChannel> is the wireless channel used by this network
<ConfigValue> is 0 for Open Security, 1 for WEP, 2 for WPA, and 3 for WPA2

160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

CONNECTOR RELATED POLLS

Once the necessary information to communicate with and configure the unit is available, access the configuration information for the connectors with the following query:

```
HTTP/1.1 GET
<Base URL>/connectors
```

The array below is sent in response to the above request, and contains as many entries as there are connectors:

```
HTTP/1.1 (200 OK)
[
        {
                "address":"connectors/<Number Value1>",
                "configuration":"<Mode Namespace>/<Number Value2>",
                "type":"<Current Mode>"
        },
        {...}
]
```

where:

<Number Value1> is the physical connector number (numbered from left to right)
<Mode Namespace> is a grouping of connectors with the same configuration
|serialports|irports
<Number Value2> is the number representing this connector within this mode namespace. "serialports/1" would represent the first connector configured as a serial port starting from the left
<Current Mode> is the current configuration of the connector
IR|IRBlaster|IRTriPort|IRTriPortBlaster|Serial

After ascertaining configuration URLs for the connector(s) to configure, use that URL to apply settings to the connector through one of the two scenarios below:

```
HTTP/1.1 PUT
<Base URL>/<Configuration>
{
        "type":"<Type Value>"
}
```

where:

<Configuration> is the URL section provided in the above request (i.e. "connectors/1")
<Type Value> can be IR|IRBlaster |IRTriPort|IRTriPortBlaster|Serial

The response below is returned in response from the above request or a GET to the same URL:

```
HTTP/1.1 (200 OK)
{
```


160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

```
        "type":"<Type Value>"
}
```

An example of applying serial settings is below.

```
HTTP/1.1 PUT
<Base URL>/serialports/1
{
        "baudrate":"<BaudValue>",
        "parity":"<ParityValue>",
        "flowcontrol":"<HandshakeValue>",
        "databits":"<DatabitsValue>",
        "stopbits":"<StopbitsValue>",
        "crossover":"false"
}
```

where:

<BaudValue> can be 300|600|1200|2400|4800|9600|14400|19200|38400|
<ParityValue> can be 0|1|2 for None, Even or Odd
<HandshakeValue> can be 0|1 for No Handshake or Hardware Handshake (CTS/RTS)
<DatabitsValue> currently can only be set to 8
<StopbitsValue> currently can only be set to 1

In addition to changing the settings with a PUT, the same URL can be polled with a GET to ask for the current settings. The information below is sent in response to the above message, or to a GET request made to the same URL.

```
HTTP/1.1 (200 OK)
{
        "baudrate":"19200",
        "parity":"0",
        "flowcontrol":"0",
        "databits":"8",
        "stopbits":"1"
}
```

IR COMMAND TRANSMISSION

When sending IR commands to a Flex unit via HTTP, an HTTP POST is used to send the JSON packet as shown below.

```
HTTP/1.1 POST
<Base URL>/irports/<IR port number>/sendir
```

Header Values:

"crossDomain" must be true if issuing this command from the web page of a different Flex unit

```
{
```


160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

```
"frequency":"<FrequencyString>",
"preamble":"<PreamblePulseTrain>",
"irCode":"<IRCodePulseTrain>",
"repeat":"<RepeatCount>"

}
```

where:

<FrequencyString> is the frequency of the command in hertz (Hz)
<PreamblePulseTrain> is a comma separated list of numeric values signifying the first portion of the IR command that is only fired once. This portion is not repeated when using a repeat count.
<IRCodePulseTrain> is a comma separated list of numeric values signifying the full IR command when not using a preamble. This portion of the code is repeated when using a repeat count.
<RepeatCount> is the number of times that the <IRCodePulseTrain> is to be repeated after sending the preamble of the command in the case that it is used.

IR LEARNING

IR learning can be initiated via HTTP with the following GET request. This will facilitate capture of a single command, and any subsequent captures will require a repeated request.

```
HTTP/1.1 GET
<Base URL>/irlearn
```

Once the above request is received, the unit will issue a "HTTP/1.1 200 OK" response and will then wait until an IR command is received by the IR learner. At this point, the unit will return a packet with a JSON payload which uses our HTTP IR format, allowing for immediate retransmission for testing purposes or storage for later use.

```
HTTP/1.1 (200 OK)
{

"frequency":"<FrequencyString>",
"preamble":"<PreamblePulseTrain>",
"irCode":"<IRCodePulseTrain>",
"repeat":"<RepeatCount>"
 }
```

ONE WAY SERIAL COMMUNICATION

The iTach Flex allows for raw communications to send one way serial transmissions. Due to necessary logic required to parse incoming serial data, there is currently no HTTP method to receive serial data. Serial transmissions use the structure below

160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

HTTP/1.1 POST
<Base URL>/serialports/1/sendserial

Header Values:

"crossDomain" must be true if issuing this command from the web page of a different Flex unit
"contentType" must be 'text/plain; charset=utf-8' in order to send strings of raw data with no alteration or encoding

By using this URL and these header values, simply fill the HTTP payload with the data to be sent through the connected Flex Link Serial Cable.

FILE SYSTEM

The iTach Flex contains a FAT style read/write file system which holds the web configuration files, as well as the users files that can be added using HTTP POST. Note that all files should be compressed (gzip'd) before being uploaded or the iTach Flex will return the HTTP header of *Content-Encoding:gzip*. This is to save space in the file system and to reduce the amount of traffic sent from the iTach Flex. Traffic can be quite substantial as HTML, JS, and CSS files can typically compress down to a third of their original size.

A GET to the file system will return all contained files and their attributes in a JSON array as shown in the example below:

HTTP/1.1 GET
<Base URL>/files

Sent in response to the above request:

HTTP/1.1 (200 OK)
[
{"name":"filename.ext","size":"32615","attributes":"32","timestamp":"29306"},
{...}
]

The Attributes are a set of bit flags:

READ_ONLY	0x01
HIDDEN	0x02
DIRECTORY	0x10
ARCHIVE	0x20

By using the POST directive, files can be added to the file system for later use, including, but not limited to, web page files, images, or even storage of variable values. An example is shown below:

HTTP/1.1 POST
<Base URL>/files/<fileName>
<Payload>

where:

160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808

<fileName> is the name of the file being added to the file system (including file extension)
<Payload> is the payload data contents of the file being added

CODE EXAMPLES

Below are AJAX code examples for IR learning and retransmission, as well as serial output from a browser:

```
<body>
    <script src="Scripts/jquery-2.0.3.js"></script>
    <script>
        (function ($) {
            $.ajax({
                url: 'http://<FLEX IP ADDRESS>/api/v1/irlearn',
                type: 'GET',
                crossDomain: true
            }).done(function (result) {
                $.ajax({
                    url: 'http://<FLEX IP ADDRESS>/api/v1/irports/1/sendir',
                    type: 'POST',
                    contentType: 'application/json',
                    data:  JSON.stringify(result),
                    crossDomain: true

                });
            });
        })(jQuery);
    </script>
</body>
```

```
<body>
    <script src="Scripts/jquery-2.0.3.js"></script>
    <script>
        (function ($) {
            $.ajax({
                url: 'http://<FLEX IP ADDRESS>/api/v1/serialports/1/sendserial',
                type: 'POST',
                contentType: 'text/plain; charset=utf-8',
                data: 'Hello World',
                crossDomain: true
            });
        })(jQuery);
    </script>
</body>
```


160 East California Street, PO Box 1659
Jacksonville, Oregon 97530
Phone: 541-899-4800
Fax: 541-899-4808