

APOLLO HTTP API REFERENCE

V 1.1

MAY 2010

1. Overview

1.1 Description

This document specifies the external HTTP-based application programming interface of Apollo Devices with firmware version 1.7.0 and above.

The HTTP-based interface provides the functionality for retrieving system information and getting/setting configuration parameters. The CGI-requests are handled by the device's built-in web server.

The API is a simple HTTP API that can be accessed from any language that supports creation of HTTP requests. The API returns [JSON](#) encoded data. You should use a JSON library to decode the data returned. Some good JSON libraries are, [JSON.simple](#) (Java), [Json.NET](#) (.NET), [simplejson](#) (Python), [json](#) (Ruby).

1.2 URL Syntax

In URL syntax and in descriptions of CGI arguments, text in italics within angle brackets denotes content that should be replaced with either a value or a string. When replacing the text string, the angle brackets must also be replaced. For example, the IP of the Apollo device is denoted by *<device_ip>* in the URL syntax description. In the URL syntax examples *<device_ip>* is replaced by the IP address 192.168.1.42.

The general format for HTTP-GET based requests is:

```
http://<device_ip>:80/cgi-bin/webgui/webgui.cgi?command=<command>&
param1=<param1>&param2=<param2>&param3=<param3>&...&
```

The available commands/parameters are documented in section 2 of this document.

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_network_config&
dhcpc=0&ip=192.168.1.42&gw=0.0.0.0&mask=255.255.255.0&
```

Important Notice: Care must be taken in order to supply ALL required parameters for a specific command. For instance, in the previous example ALL three parameters (dhcpc, gw and mask) are necessary in order for the command to be valid.

1.3 Return Data

Upon successful completion of the request, the device returns an HTTP 200 OK response with a JSON-encoded body. The syntax for the JSON-encoded return value for each command is documented in section 2 of this document. For example, upon successful completion of the `set_network_config` command, the device returns an HTTP 200 OK response with the following body:

```
{ "dhcpc":0, "ip":"192.168.1.42", "gw":"0.0.0.0", "mask":"255.255.255.0"}
```

Upon error, the device returns an appropriate HTTP Error Code (500 Internal Server Error, 400 Bad Request, 501 Not Implemented or 503 Service Unavailable), with an optional JSON-encoded body of the form:

```
{  
  error: "<string:error_message>"  
}
```

For example, the following command:

```
http://192.168.1.42/cgi-bin/webgui/webgui.cgi?  
command=set_password&new_pass=small
```

produces an HTTP 500 Internal Server Error response with the following body:

```
{  
  "error": "Too small password(at least 8 characters)."  
}
```

1.4 Definitions

Ptp Point To Point Firmware

Pmp Point To Multipoint Firmware

BS Base Station, i.e. the Apollo Access Point for Pmp Firmware/Topology

SS Subscriber Station, i.e. the Apollo Access Point Client for Pmp
Firmware/Topology

Master The Master Endpoint for a Ptp link

Slave The Slave Endpoint for a Ptp link

1.5 Reference .NET Implementation

Antcor provides a complete reference implementation of the HTTP Api in source (C# code) and binary forms (ApolloInterface.dll .NET assembly for use in .NET projects). Please consult the source code documentation for more implementation details.

2. Command/Parameter Reference

2.1 get_system_info

Description: Retrieves the current system information of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Types: All (Ptp, Pmp)

Return Data on success:

```
{
    "type": <string>, // "Ptp" or "Pmp" depending on firmware type
    "version": <string>,
    "build": <string>,
    "api_version": <string>,
    "uptime": <string>,
    "cpu_usage_perc": <integer>,
    "free_mem_kb": <integer>,
}
```

Example Request:

```
http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_system_info
```

Example Response:

```
{
    "type": "Ptp",
    "version": "Apollo v.1.7.0-MULTIRF",
    "build": "Build 233 (Mon Dec 21 18:32:31 EET 2009)",
    "api_version": "1.0",
    "uptime": "0d/0h/25m/4s",
    "cpu_usage_perc": 99,
    "free_mem_kb": 28820
}
```

2.2 get_apollo_config

Description: Retrieves the current Apollo Protocol Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: BS, Master

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
"arq_enabled":<int>, // 0 or 1, Indicates if Arq is enabled
"arq_maxtry":<int>, // Max ARQ retries for Apollo Data Packets
"pack_enabled":<int>, // 0 or 1, Indicates if packing is enabled
"tcpopt_enabled":<int>, // 0 or 1, Indicates if TCP Optimizer is enabled
"act_man":<int>, // 0 or 1, Indicates if Activity Management is
enabled
"basic_timeout":<int>, // The basic timeout for Apollo Protocol in msec
"reg_timeout":<int>, // The registration timeout for Apollo Protocol in
msec
"reg_interval":<int>, // The registration interval for Apollo Protocol in
mse
"max_retries":<int>, // Maximum retries for an Apollo Protocol Packe
"max_failed_polls":<int> // Number of failed polls before disassociating a
station
}
```

Example Request:

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_apollo_config

Example Response:

```
{
  "arq_enabled":1,
  "arq_maxtry":4,
  "pack_enabled":1,
  "tcpopt_enabled":0,
  "act_man":0,
  "basic_timeout":20,
  "reg_timeout":20,
  "reg_interval":20,
  "max_retries":8,
  "max_failed_polls":64
}
```

2.3 set_apollo_config

Description: Changes the Apollo Protocol Configuration of the target device

Request Method: GET

Required Parameters:

```
arq_enabled:<int>, // 0 or 1, Indicates if Arq is enabled
arq_maxtry:<int>, // Max ARQ retries for Apollo Data Packets
pack_enabled:<int>, // 0 or 1, Indicates if packing is enabled
```

```
tcpopt_enabled:<int>, // 0 or 1, Indicates if TCP Optimizer is enabled
act_man:<int>, // 0 or 1, Indicates if Activity Management is enabled
basic_timeout:<int>, // The basic timeout for Apollo Protocol in msec
reg_timeout:<int>, // The registration timeout for Apollo Protocol in msec
reg_interval:<int>, // The registration interval for Apollo Protocol in msec
max_retries:<int>, // Maximum retries for an Apollo Protocol Packe
max_failed_polls:<int> // Number of failed polls before disassociating a
station
```

Applicable Opmodes: BS, Master

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
"arq_enabled":<int>, // 0 or 1, Indicates if Arq is enabled
"arq_maxtry":<int>, // Max ARQ retries for Apollo Data Packets
"pack_enabled":<int>, // 0 or 1, Indicates if packing is enabled
"tcpopt_enabled":<int>, // 0 or 1, Indicates if TCP Optimizer is enabled
"act_man":<int>, // 0 or 1, Indicates if Activity Management is enabled
"basic_timeout":<int>, // The basic timeout for Apollo Protocol in msec
"reg_timeout":<int>, // The registration timeout for Apollo Protocol in msec
"reg_interval":<int>, // The registration interval for Apollo Protocol in
msec
"max_retries":<int>, // Maximum retries for an Apollo Protocol Packe
"max_failed_polls":<int> // Number of failed polls before disassociating a
station
}
```

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_apollo_config&
arq_enabled=1&arq_maxtry=4&pack_enabled=1&tcpopt_enabled=0&act_man=0&basic_t
imeout=20&reg_timeout=20&reg_interval=20&max_retries=8&max_failed_polls=64&
```

Example Response:

```
{
    "arq_enabled":1,
    "arq_maxtry":4,
    "pack_enabled":1,
    "tcpopt_enabled":0,
    "act_man":0,
    "basic_timeout":20,
    "reg_timeout":20,
    "reg_interval":20,
    "max_retries":8,
    "max_failed_polls":64
}
```

2.4 get_apollo_qos

Description: Retrieves the current Apollo QoS Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
"rt_delay": <string>, // Desired delay for Real Time traffic in seconds
                        // (only valid for Base Station/Master Devices)
"rules": <JSON array of rules>
}
// Each rule is a JSON object of the following form:
{
"apoll_real_time_protocol":<string>, // Packet's network protocol ("tcp"
                                     // , "udp" or "icmp"
"Real_Time_sport":<int>, // Packet's source port (0 for any)
"Real_Time_dport":<int> // Packet's destination port (0 for any)
}
```

Example Request:

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_apollo_qos

Example Response (2 QoS Rules):

```
{
  "rt_delay": "0.06",
  "rules": [
    {
      "apoll_real_time_protocol": "icmp",
      "Real_Time_sport": 0,
      "Real_Time_dport": 0
    },
    {
      "apoll_real_time_protocol": "tcp",
      "Real_Time_sport": 1234,
      "Real_Time_dport": 0 }
  ]
}
```

2.5 set_apollo_delay

Description: Changes the desired Real Time delay of the target device

Request Method: GET

Required Parameters:

```
rt_delay:<double>, // Desired RT delay in seconds (decimal point = .)
```

Applicable Opmodes: BS, Master

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_apollo_qos'

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_apollo_delay&
rt_delay=0.08&
```

Example Response:

```
{
  "rt_delay": "0.08",
  "rules": [
    {
      "apoll_real_time_protocol": "icmp",
      "Real_Time_sport": 0,
      "Real_Time_dport": 0
    },
    {
      "apoll_real_time_protocol": "tcp",
      "Real_Time_sport": 1234,
      "Real_Time_dport": 0 }
  ]
}
```

2.6 add_apollo_rule

Description: Adds an extra QoS classification rule to the target device

Request Method: GET

Required Parameters:

```
apoll_real_time_protocol: string, // Packet's network protocol ("tcp"
                                // , "udp" or "icmp"
Real_Time_sport: int, // Packet's source port (0 for any)
Real_Time_dport: int, // Packet's destination port (0 for any)
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_apollo_qos'

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=add_apollo_rule&
apoll_real_time_protocol=udp&Real_Time_sport=567&Real_Time_dport=0&
```

Example Response:

```
{
  "rt_delay": "0.08",
```

```

"rules": [
{
    "apoll_real_time_protocol": "icmp",
    "Real_Time_sport": 0,
    "Real_Time_dport": 0
},
{
    "apoll_real_time_protocol": "tcp",
    "Real_Time_sport": 1234,
    "Real_Time_dport": 0
},
{
    "apoll_real_time_protocol": "udp",
    "Real_Time_sport": 567,
    "Real_Time_dport": 0
}
]
}

```

2.7 delete_apollo_rule

Description: Adds an extra QoS classification rule to the target device

Request Method: GET

Required Parameters:

index: int, // The index of the rule that will be deleted (First=0)

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_apollo_qos'

Example Request:

```

http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=delete_apollo_rule&
index=2

```

Example Response:

```

{
    "rt_delay":"0.08",
    "rules": [
    {
        "apoll_real_time_protocol": "icmp",
        "Real_Time_sport": 0,
        "Real_Time_dport": 0
    },
    {
        "apoll_real_time_protocol": "tcp",
        "Real_Time_sport": 1234,

```

```

        "Real_Time_dport": 0
    }
]
}

```

2.8 get_ptp_wireless_config

Description: Retrieves the Wireless Configuration of a Point to Point device

Request Method: GET

Required Parameters: None

Applicable Opmodes: Master, Slave

Applicable Firmware Type: Ptp

Return Data on success:

```

{
"opmode": "<integer>", // Operational Mode "3"=Master, "2"=Slave
"ieee_mode": <integer>, // IEEE 802.11 mode 1=802.11a, 3=802.11bg
"rate_0": "<integer>", // Physical data rate (common for both primary/
// secondary interfaces), use 'get_wireless_rates'
// in order to get the available rates
"rate_auto": <integer>, // 0 or 1, Disable/Enable Auto-Rate operation (Not
// supported in v1.7.0)
"multirf_enabled": <integer>, // 1/0, Enables/Disables the Secondary
// Interface (only valid for Master Devices)
"distance": "<integer>", // Link distance in meters
"wpa_key": "<string>", // WPA Security Key
"security": <integer>, // 1/0 Enables/Disables WPA Security
"apmac_0": "<mac>", // MAC Address of the Primary Wireless Interface
// of the peer device
"apmac_1": "<mac>", // MAC Address of the Secondary Wireless Interface
// of the peer device
"frequency_0": "<integer>", // Center frequency of the Primary Wireless
// Interface in KHz (use 'get_wireless_channels'
// in order to get the available channels)
"frequency_1": "<integer>", // Center frequency of the Secondary Wireless
// Interface in KHz (use 'get_wireless_channels' in
// order to get the available channels)
"txpower_0": "<integer>", // Transmit Power for the Primary Wireless
// Interface (Valid values are 5, 10, 15, 20, 25
// and 30)
"txpower_1": "<integer>", // Transmit Power for the Secondary Wireless
// Interface (Valid values are 5, 10, 15, 20,
// 25 and 30)
"IkrAntenna_0": <integer>, // Antenna Selection for the Primary Wireless
// Interface 0=Aux, 1=Main (Valid only if

```

```

        // Diversity is off)
"IkrAntenna_1":<integer>, // Antenna Selection for the Secondary Wireless
        // Interface 0=Aux, 1=Main (Valid only if
        // Diversity is off)
"IkrDiversity_0":<integer>, // 1/0, Enables/Disables diversity for the
        // Primary Wireless Interface
"IkrDiversity_1":<integer> // 1/0, Enables/Disables diversity for the
        // Secondary Wireless Interface
}

```

Example Request:

```

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=
get_ptp_wireless_config

```

Example Response:

```

{
  "opmode":"3",
  "ieee_mode":1,
  "rate_0":"36",
  "rate_auto":0,
  "multirf_enabled":1,
  "distance":"3300",
  "wpa_key":"1234567890",
  "security":1,
  "apmac_0":"00:80:48:41:2F:2E",
  "apmac_1":"00:80:48:45:16:8E",
  "frequency_0":"5240",
  "frequency_1":"5785",
  "txpower_0":"15",
  "txpower_1":"15",
  "IkrAntenna_0":1,
  "IkrAntenna_1":1,
  "IkrDiversity_0":0,
  "IkrDiversity_1":0
}

```

2.9 set_ptp_wireless_config

Description: Changes the Wireless Configuration of a Point to Point device

Request Method: GET

Required Parameters:

```

opmode:<integer>, // Operational Mode 3=Master, 2=Slave
ieee_mode:<integer>, // IEEE 802.11 mode 1=802.11a, 3=802.11bg
rate_0:<integer>, // Physical data rate (common for both primary/
// secondary interfaces), use 'get_wireless_rates' in
// order to get the available rates

```

```

rate_auto:<integer>, // 0 or 1, Disable/Enable Auto-Rate operation (Not
                    // supported in v1.7.0)
multirf_enabled:<integer>, // 1/0, Enables/Disables the Secondary Interface
                        //(only valid for Master Devices)
distance:<integer>, // Link distance in meters
wpa_key:<string>, // WPA Security Key
security:<integer>, // 1/0 Enables/Disables WPA Security
apmac_0:<mac>, // MAC Address of the Primary Wireless Interface of the
            // peer device
apmac_1:<mac>, // MAC Address of the Secondary Wireless Interface of
            // the peer device
frequency_0:<integer>, // Center frequency of the Primary Wireless Interface
                    // in KHz (use 'get_wireless_channels' in order to
                    //get the available channels)
frequency_1:<integer>, // Center frequency of the Secondary Wireless
                    // interface in KHz (use 'get_wireless_channels' in
                    //order to get the available channels)
txpower_0:<integer>, // Transmit Power for the Primary Wireless Interface
                    // (Valid values are 5, 10, 15, 20, 25 and 30)
txpower_1:<integer>, // Transmit Power for the Secondary Wireless Interface
                    // (Valid values are 5, 10, 15, 20, 25 and 30)
IkrAntenna_0:<integer>, // Antenna Selection for the Primary Wireless
                    // Interface 0=Aux, 1=Main (Valid only if Diversity
                    // is off)
IkrAntenna_1:<integer>, // Antenna Selection for the Secondary Wireless
                    // Interface 0=Aux, 1=Main (Valid only if Diversity
                    // is off)
IkrDiversity_0:<integer>, // 1/0, Enables/Disables diversity for the Primary
                    // Wireless Interface
IkrDiversity_1:<integer> // 1/0, Enables/Disables diversity for the
                    // Secondary Wireless Interface

```

Applicable Opmodes: Master, Slave

Applicable Firmware Type: Ptp

Return Data on success: Same as 'get_ptp_wireless_config'

Example Request:

```

http://192.168.1.42:80/cgi-
bin/webgui/webgui.cgi?command=set_ptp_wireless_config&
opmode=3&ieee_mode=1&rate_0=36&rate_auto=0&multirf_enabled=1&distance=3300&w
pa_key=1234567890&security=1&apmac_0=00:80:48:41:2F:2E&apmac_1=00:80:48:45:1
6:8E&frequency_0=5240&frequency_1=5785&txpower_0=15&txpower_1=15&IkrAntenna_
0=1&IkrAntenna_1=1&IkrDiversity_0=0&IkrDiversity_1=0&

```

Example Response: Same as 'get_ptp_wireless_config'

2.10 get_pmp_wireless_config

Description: Retrieves the Wireless Configuration of a Point to Multipoint device

Request Method: GET

Required Parameters: None

Applicable Opmodes: BS, SS

Applicable Firmware Type: Pmp

Return Data on success:

```
{
    "opmode":"<integer>", // Operational Mode "3"=BS, "2"=SS
    "ieee_mode":"<string>", // IEEE 802.11 mode "a"=802.11a, "bg"=802.11bg
    (NOTE:This is different than the ieee_mode parameter of ptp_wireless_config)
    "rate":"<integer>", // Physical data rate, use 'get_wireless_rates' in
order to get the available rates
    "rate_auto":<integer>, // 0 or 1, Disable/Enable Auto-Rate operation
(Not supported in v1.7.0)
    "distance":"<integer>", // Link distance in meters
    "wpa_key":"<string>", // WPA Security Key
    "security":<integer>, // 1/0 Enables/Disables WPA Security
    "apmac":"<mac>", // MAC Address of the Base Station (applies to SS
only, not valid for BS)
    "frequency":"<integer>", // Center frequency of the Wireless Interface
in KHz (use 'get_wireless_channels' in order to get the available channels)
    "txpower":"<integer>", // Transmit Power for the Wireless Interface
(Valid values are 5, 10, 15, 20, 25 and 30)
    "IkrAntenna":<integer>, // Antenna Selection for the Wireless
Interface 0=Aux, 1=Main (Valid only if Diversity is off)
    "IkrDiversity":<integer>, // 1/0, Enables/Disables diversity
}
```

Example Request:

```
http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=
get_pmp_wireless_config
```

Example Response:

```
{
    "opmode":"3",
    "ieee_mode":"a",
    "rate":"36",
    "rate_auto":0,
    "distance":"3300",
    "wpa_key":"",
    "security":0,
```

```

    "apmac":"00:00:00:00:00:00",
    "frequency":"5240",
    "txpower":"15",
    "IkrAntenna":1,
    "IkrDiversity":0,
}

```

2.11 set_pmp_wireless_config

Description: Changes the Wireless Configuration of a Point to Multipoint device

Request Method: GET

Required Parameters:

```

    opmode:<integer>, // Operational Mode "3"=BS, "2"=SS
    ieee_mode:<integer>, // IEEE 802.11 mode "a"=802.11a, "bg"=802.11bg
(NOTE:This is different than the ieee_mode parameter of ptp_wireless_config)
    rate:<integer>, // Physical data rate (use 'get_wireless_rates' in
order to get the available rates)
    rate_auto:<integer>, // 0 or 1, Disable/Enable Auto-Rate operation
(Not supported in v1.7.0)
    distance:<integer>, // Link distance in meters
    wpa_key:<string>, // WPA Security Key
    security:<integer>, // 1/0 Enables/Disables WPA Security
    apmac:<mac>, // MAC Address of the Base Station (applies to SS only,
not valid for BS)
    frequency:<integer>, // Center frequency of the Wireless Interface in
KHz (use 'get_wireless_channels' in order to get the available channels)
    txpower:<integer>, // Transmit Power for the Wireless Interface (Valid
values are 5, 10, 15, 20, 25 and 30)
    IkrAntenna:<integer>, // Antenna Selection for the Wireless Interface
0=Aux, 1=Main (Valid only if Diversity is off)
    IkrDiversity:<integer>, // 1/0, Enables/Disables diversity for the
Wireless Interface
    IkrDiversity_1:<integer> // 1/0, Enables/Disables diversity for the
Secondary Wireless Interface

```

Applicable Opmodes: BS, SS

Applicable Firmware Type: Pmp

Return Data on success: Same as 'get_pmp_wireless_config'

Example Request:

```

http://192.168.1.42:80/cgi-
bin/webgui/webgui.cgi?command=set_pmp_wireless_config&

```

```
opmode=3&ieee_mode=a&rate=36&rate_auto=0&distance=3300&wpa_key=&security=0&a
pmac=00:00:00:00:00:00&frequency=5240&txpower=15&IkrAntenna=1&IkrDiversity=0
&
```

Example Response: Same as 'get_pmp_wireless_config'

2.12 get_acl_config

Description: Retrieves the ACL Configuration of a Point to Multipoint BS

Request Method: GET

Required Parameters: None

Applicable Opmodes: BS

Applicable Firmware Type: Pmp

Return Data on success:

```
{
  "macs": <JSON array of MAC Addresses>
}
```

Example Request:

```
http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_acl_config
```

Example Response (2 ACL Entries):

```
{
  "macs": [
    "00:01:02:03:04:05",
    "00:80:48:41:2f:2e"
  ]
}
```

2.13 add_acl_entry

Description: Adds an extra entry in the ACL configuration of a Point to Multipoint

Base Station

Request Method: GET

Required Parameters:

```
newAclMacAddress: mac, // The mac address that will be added
```

Applicable Opmodes: BS

Applicable Firmware Type: Pmp

Return Data on success: Same as 'get_acl_config'

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=add_acl_entry&
```

newAclMacAddress=00:de:ad:de:ad:de

Example Response:

```
{
  "macs": [
    "00:01:02:03:04:05",
    "00:80:48:41:2f:2e",
    "00:de:ad:de:ad:de"
  ]
}
```

2.14 delete_acl_entry

Description: Deletes an ACL entry from the ACL of a Point to Multipoint Base Station

Request Method: GET

Required Parameters:

index: int, // The index of the entry that will be deleted (First=0)

Applicable Opmodes: BS

Applicable Firmware Type: Pmp

Return Data on success: Same as 'get_acl_config'

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=delete_acl_entry&
index=2
```

Example Response:

```
{
  "macs": [
    "00:01:02:03:04:05",
    "00:80:48:41:2f:2e"
  ]
}
```

2.15 get_cc_config

Description: Retrieves Retrieves the Country Code Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
```

```
"cc": <int>, // Country Code, 356=India, 840=US, 1=All
"channelBW": <int>, // 5,10 or 20 (Bandwidth in MHz)
"channelSep": <int> // 5 (Seperation in MHz)
}
```

Example Request:

`http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_cc_config`

Example Response:

```
{
  "cc": 840,
  "channelBW": 20,
  "channelSep": 5
}
```

2.16 set_cc_config

Description: Changes Changes the Country Code Configuration of the target device

Request Method: GET

Required Parameters:

```
cc:<int>, // Country Code, 356=India, 840=US, 1=All
channelBW:<int>, // 5,10 or 20 (Bandwidth in MHz)
channelSep:<int>, // 5 (Seperation in MHz)
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_cc_config'

Example Request:

`http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_cc_config&cc=356&channelBw=20&channelSep=5&`

Example Response:

```
{
  "cc": 356,
  "channelBW": 20,
  "channelSep": 5
}
```

2.17 get_services_config

Description: Retrieves the Services Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
    "snmp_enable":<integer>, // 0/1 Indicates if SNMP is enabled
    "snmp_port":<integer>, // Port for SNMP service
    "snmp_community":<string>, // SNMP community
    "ssh_enable":<integer>, // 0/1 Indicates if SSH service is enabled
    "ssh_port":<integer> // SSH service port
}
```

Example Request:

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_services_config

Example Response:

```
{
    "snmp_enable":0,
    "snmp_port":161,
    "snmp_community":"public",
    "ssh_enable":1,
    "ssh_port":22
}
```

2.18 set_services_config

Description: Changes the Country Code Configuration of the target device

Request Method: GET

Required Parameters:

```
snmp_enable:<integer>, // 0/1 Indicates if SNMP is enabled
snmp_port:<integer>, // Port for SNMP service
snmp_community:<string>, // SNMP community
ssh_enable:<integer>, // 0/1 Indicates if SSH service is enabled
ssh_port:<integer> // SSH service port
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_services_config'

Example Request:

http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_services_config&snmp_enable=0&snmp_port=161&snmp_community=public&ssh_enable=1&ssh_port=22&

Example Response: Same as 'get_services_config'

2.19 get_wireless_channels

Description: Retrieves Retrieves the available channels for the target device **for the current Country Code settings.**

NOTE: The available channels are different depending on the Country Code Configuration of the device. You should retrieve a new list of available channels after a change in Country Code configuration

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
3: <Json array of channels>, // Available channels for IEEE 802.11bg mode
1: <Json array of channels> // Available channels for IEEE 802.11a mode
}
// Each channel is a JSON object of the following form:
{
"ieee_number": <integer>, // IEEE 802.11 number assigned to this channel
"freq": <integer> // Center Frequency in KHz
}
```

Example Request:

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_wireless_channels

Example Response:

```
{
  3: [
    { "ieee_number":1, "freq": 2412 } ,
    { "ieee_number":2, "freq": 2417 } ,
    { "ieee_number":3, "freq": 2422 } ,
    { "ieee_number":4, "freq": 2427 } ,
    { "ieee_number":5, "freq": 2432 } ,
    { "ieee_number":6, "freq": 2437 } ,
    { "ieee_number":7, "freq": 2442 } ,
    { "ieee_number":8, "freq": 2447 } ,
    { "ieee_number":9, "freq": 2452 } ,
    { "ieee_number":10, "freq": 2457 } ,
    { "ieee_number":11, "freq": 2462 }
  ] ,
  1: [
    { "ieee_number":36, "freq": 5180 } ,
    { "ieee_number":37, "freq": 5185 } ,
    { "ieee_number":38, "freq": 5190 } ,
    { "ieee_number":39, "freq": 5195 } ,
    { "ieee_number":40, "freq": 5200 } ,
    { "ieee_number":41, "freq": 5205 } ,
    { "ieee_number":42, "freq": 5210 } ,
    { "ieee_number":43, "freq": 5215 } ,
  ]
}
```

```

    { "ieee_number":44, "freq": 5220 } ,
    { "ieee_number":45, "freq": 5225 } ,
    { "ieee_number":46, "freq": 5230 } ,
    { "ieee_number":47, "freq": 5235 } ,
    { "ieee_number":48, "freq": 5240 } ,
    { "ieee_number":52, "freq": 5260 } ,
    { "ieee_number":53, "freq": 5265 } ,
    { "ieee_number":54, "freq": 5270 } ,
    { "ieee_number":55, "freq": 5275 } ,
    { "ieee_number":56, "freq": 5280 } ,
    { "ieee_number":57, "freq": 5285 } ,
    { "ieee_number":58, "freq": 5290 } ,
    { "ieee_number":59, "freq": 5295 } ,
    { "ieee_number":60, "freq": 5300 } ,
    { "ieee_number":61, "freq": 5305 } ,
    { "ieee_number":62, "freq": 5310 } ,
    { "ieee_number":63, "freq": 5315 } ,
    { "ieee_number":64, "freq": 5320 } ,
    { "ieee_number":149, "freq": 5745 } ,
    { "ieee_number":150, "freq": 5750 } ,
    { "ieee_number":151, "freq": 5755 } ,
    { "ieee_number":152, "freq": 5760 } ,
    { "ieee_number":153, "freq": 5765 } ,
    { "ieee_number":154, "freq": 5770 } ,
    { "ieee_number":155, "freq": 5775 } ,
    { "ieee_number":156, "freq": 5780 } ,
    { "ieee_number":157, "freq": 5785 } ,
    { "ieee_number":158, "freq": 5790 } ,
    { "ieee_number":159, "freq": 5795 } ,
    { "ieee_number":160, "freq": 5800 } ,
    { "ieee_number":161, "freq": 5805 } ,
    { "ieee_number":162, "freq": 5810 } ,
    { "ieee_number":163, "freq": 5815 } ,
    { "ieee_number":164, "freq": 5820 } ,
    { "ieee_number":165, "freq": 5825 }
  ]
}

```

2.20 get_wireless_rates

Description: Retrieves the available data rates for the target device, for every possible channel bandwidth setting

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
5: <Json array of rates as string>, // Available rates for 5MHz channel BW
10: <Json array of rates as string>, // Available rates for 10MHz channel BW
20: <Json array of rates as string> // Available rates for 20MHz channel BW
}
```

Example Request:

http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_wireless_rates

Example Response:

```
{
  5: [
    "1.5",
    "2.25",
    "3",
    "4.5",
    "6",
    "9",
    "12",
    "13.5"
  ],
  10: [
    "3",
    "4.5",
    "6",
    "9",
    "12",
    "18",
    "24",
    "27"
  ],
  20: [
    "6",
    "9",
    "12",
    "18",
    "24",
    "36",
    "48",
    "54"
  ]
}
```

2.21 get_network_config

Description: Retrieves the Network Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
    "dhcpc":<integer>, // 0/1 Indicates if the device obtains IP Address
                        // via DHCP
    "ip":"<ip>", // IP Address of the device
    "gw":"<ip>", // Default Gateway IP Address
    "mask":"<ip_subnet>" // Subnet mask
}
```

Example Request:

`http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_network_config`

Example Response:

```
{
    "dhcpc":0,
    "ip":"192.168.1.42",
    "gw":"0.0.0.0",
    "mask":"255.255.255.0"
}
```

2.22 set_network_config

Description: Changes the Network Configuration of the target device

Request Method: GET

Required Parameters:

```
dhcpc:<integer>, // 0/1 Indicates if the device obtains IP Address via
                // DHCP
ip:<ip>, // IP Address of the device
gw:<ip>, // Default Gateway IP Address
mask:<ip_subnet>" // Subnet mask
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_network_config'

Example Request:

`http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_network_config&dhcpc=0&ip=192.168.1.42&gw=0.0.0.0&mask=255.255.255.0&`

Example Response: Same as 'get_services_config'

2.23 get_vlan_config

Description: Retrieves the Vlan Configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success:

```
{
  "rules": <Json array of Vlan rules> // List of Vlan Rules
}
// Each Vlan Rule is a JSON object of the following form:
{
  "index":<integer>, // The rule's index.
  "tag":<integer>, // The Vlan tag for this rule
  "iface":<string>, // The VLAN interface name (Can be one of "eth0" or "br0")
  "bridge":<string>, // The name of the bridge that contains the Vlan interface (The
                      // bridge name is "br0" if iface is "eth0" and the Vlan interface
                      // is under a brige. It is empty on all other circumstances.
  "ip":<ip>, // The IP that has been assigned to a non-bridged Vlan interface
              // (This is only valid for non-bridged Vlan Rules, i.e. rules with
              // bridge="")
  "mask":<integer> // The Subnet Mask for the IP that will be assigned to a non-
                  // bridged Vlan interface (This is only valid for non-bridged Vlan
                  // Rules, i.e. rules with bridge="")
}
```

Example Request:

```
http://192.168.1.42/cgi-bin/webgui/webgui.cgi?command=get_vlan_config
```

Example Response: The following response shows a device configured with 2 Vlan rules:

- Rule 0 is a 'non-bridged' Vlan Rule with tag 55 for interface "br0". The Vlan interface (br0.55) is setup with the address 10.10.0.150/24.
- Rule 1 is a bridged Vlan Rule with tag 23 for interface "eth0". The Vlan interface is added in bridge "br0" and hence, it does not have IP settings of it's own (It inherits the bridge's settings).

```
{
  "rules": [
    {
      "index":0,
      "tag":55,
```

```

        "iface":"br0",
        "bridge":"",
        "ip":"10.10.0.150",
        "mask":24
    } ,
    {
        "index":1,
        "tag":23,
        "iface":"eth0",
        "bridge":"br0",
        "ip":"0.0.0.0",
        "mask":0
    }
]
}

```

2.24 set_vlan_config

Description: Changes the Vlan Configuration of the target device. (NOTE: The previously configured Vlan configuration will be lost.)

Request Method: GET

Required Parameters:

vlan_submit_iface=<iface1>.<iface2>.<iface3>.<ifaceN>.

Where <ifaceN> is the interface/bridge description for the N-th rule.

<ifaceN> is one of:

- eth0 or br0 for non-bridged Vlan
- eth0%2Fbr0 for bridged Vlan.

vlan_submit_ip=<ip1>%2B<ip2>%2B<ip3>%2B ... %2B<ipN>%2B

Where <ipN> is the ip address configuration for the N-th rule. The ip address is only valid for non-bridged Vlan interfaces and should be set to 0.0.0.0 for bridged Vlan interfaces.

vlan_submit_subnet=<mask1>%2B<mask2>%2B<mask3>%2B ... %2B<maskN>%2B

Where <maskN> is the subnet mask configuration for the N-th rule. The subnet mask is only valid for non-bridged Vlan interfaces and should be set to 24 for bridged Vlan interfaces.

vlan_submit_tag=<tag1>.<tag2>.<tag3>.<tagN>.

Where <tagN> is the Vlan tag for the N-th rule.

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: Same as 'get_vlan_config'

Example Request:

The following request configures the device with two Vlan Rules

- The first rule is a 'non-bridged' Vlan Rule with tag 55 for interface "br0". The Vlan interface (br0.55) is setup with the address 10.10.0.150/24.
- Rule 1 is a bridged Vlan Rule with tag 23 for interface "eth0". The Vlan interface is added in bridge "br0" and hence, it does not have IP settings of its own (It inherits the bridge's settings).

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_vlan_config&
vlan_submit_iface=br0.eth0%2Fbr0.&vlan_submit_ip=10.10.0.150%2B0.0.0.0%2B&vl
an_submit_subnet=24%2B24%2B&vlan_submit_tag=55.23.&
```

Example Response: Same as 'get_vlan_config'

2.25 system_reboot

Description: Reboots the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=system_reboot
```

Example Response: None

2.26 system_save

Description: Saves the configuration of the target device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=system_save
```

Example Response: None

2.27 system_save_and_reboot

Description: Saves the configuration of the target device and reboots

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?  
command=system_save_and_reboot
```

Example Response: None

2.28 system_defaults

Description: Restores the target device to factory defaults and reboots the device

Request Method: GET

Required Parameters: None

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=system_defaults
```

Example Response: None

2.29 set_password

Description: Sets the administrator password for the target device

Request Method: GET

Required Parameters:

```
new_pass: <string> // The new password to set
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
http://192.168.1.42:80/cgi-bin/webgui/webgui.cgi?command=set_password&  
new_pass=1234567890
```

Example Response: None

2.30 fwupgrade

Description: Performs firmware upgrade on the target device and reboots.

Request Method: **POST**

Required Parameters:

```
command=fwupgrade // The command "fwupgrade" should be present in the
                  // appropriate form-data multipart field named "command"
fwimage=<firmware image contents> // The file content, is provided in
                                   the HTTP body according to the format
                                   given in RFC 1867.
```

Applicable Opmodes: All (BS, SS, Master, Slave)

Applicable Firmware Type: All (Ptp, Pmp)

Return Data on success: HTTP 200 OK with empty body

Example Request:

```
POST /cgi-bin/webgui/webgui.cgi?fwupgrade HTTP/1.0
Content-Type: multipart/form-data; boundary=-----8cc513381221c32
Authorization: Basic YWRtaW46YWRtaW4=
Content-Length: 7098631
Connection: keep-alive
Host: 192.168.1.42

-----8cc513381221c32
Content-Disposition: form-data; name="command"

fwupgrade
-----8cc513381221c32
Content-Disposition: form-data; name="fwimage";
filename="ikarus_POINTRED_SENAO_apollo_upgrade.img"
Content-Type: application/octet-stream

<firmware image contents>
-----8cc513381221c32--
```

Example Response: None