Videndum Production Solutions

Autoscript Winplus-IP Automation Interface Specification

Version:	1.0
Date:	21 March 2024
Author:	Chris Deas
Copyright:	Videndum Production Solutions

Preface

This document is very much still in draft form, it is a working protocol specification for a planned automation interface for Winplus-IP. It outlines the main operational messages that are intended for V1 of the interface.

NOTE: That this is a “live” document and updates will continue to be made during implementation of the API, hence change is very likely.

Revision History

Date	Author	Version	Description
05/07/2018	C Deas	0.1	Initial early draft
10/10/2018	C Deas	0.2	Re written as REST web service
15/11/2018	D Devereux	0.3	Updated System Devices
23/11/2018	D Devereux	0.4	Updated Run Order Operations
28/11/2018	D Devereux	0.5	Updated notes on redundancy pairing
09/07/2021	D Devereux	0.6	Added information on configurations, status, logging. Extended redundancy methods.
15/12/2021	D Devereux	0.65	Added information on run order navigation
07/04/2022	D Devereux	0.7	Added current configuration query endpoint
21/06/2022	D Devereux	0.8	Added Commands
05/07/2022	D Devereux	0.9	Added “blank” to load configuration. Added paired peer information to redundancy status query. Added information on new endpoint to get current state of loaded configurations.
20/10/2023	D Devereux	1.0	Added closed caption control end point. System device and device discoverer end points now support WC-IP, WB-IP and Shuttle XPRESS and Shuttle PRO. Fixed 6.1 Discovered devices not returning the correct device type

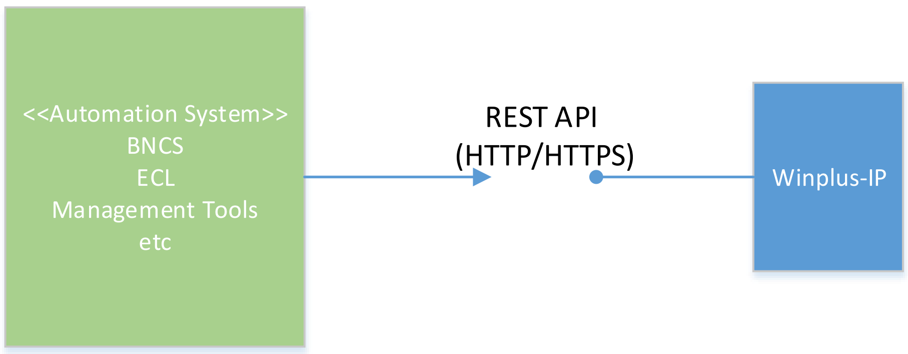
Winplus-IP version supporting automation interface revision

Automation Interface Version	Minimum version of WP-IP that supports that level
0.6	1.11.0.25
0.65	1.12.0.100
0.7	1.12.0.120
0.8	1.12.1.101
0.9	1.12.1.112
1.0	1.13.1.TBA

Contents

1. Introduction

The Winplus-IP automation interface (WPA) is a REST API that allows an external automation system access to certain configuration and system operations of the prompter system. Such as discovering, assigning, and removal of prompter devices (monitors, controllers etc) from Winplus, querying the prompting status, enabling prompter output and runorder management.



2. REST API Specifics

2.1. API Configuration

The API can be configured to use either HTTP or HTTPS. In both case the server port is configurable. The default port will be set to 8080.

2.2. Message Encoding/Data Format

- Character encoding = UTF-8
- Data format = JSON

2.3. Authentication

WPA uses standard OAuth 2 authentication with a grant type of Resource Owner Password Credentials (ROPC). Only a single user ID is supported “default”, with a configurable password which defaults to “winplus”.

Request

```
POST /Token
```

2.4. HTTP Methods

WPA uses appropriate HTTP verbs for every action.

- GET Used for retrieving resources.
- POST Used for creating resources and performing resource actions.
- PUT Used for updating resources.
- DELETE Used for deleting resources.

2.5. Status Codes and Error Reporting

WPA uses standard HTTP status codes to indicate success or failure of a request.

API calls generally return status code 200 to indicate success and status code 400 to indicate an error.

A success or failure, of all API calls return the following JSON message structure first in the HTTP response body.

```
Content-Type:application/json;charset=UTF-8

{

    “code”: integer error code 0 for success,
    "message": "detailed error description",
}
```

The code field is reserved for more granular error reporting from an API call. The message field is a string field describing an error. It will be set to "`success`" if no error occurred.

2.6. URI's

The base URI is /`api/v{version}/`

`e.g. /api/v1/`

2.7. Long Polling

Certain status methods implement long polling. When you attempt to 'get' the value from those methods they will not return straight away. They will only return once whatever you are subscribed to has changed. This is useful for more of an event driven architecture. Timeouts will occur if whatever you are subscribed to does not change very often so you will need to detect the timeouts and re-attach.

3. System Devices

`URI: /api/v1/sysdevices`

All devices within the system have a unique id "`dev_id`" (a GUID) that is generated by Winplus-IP when the device is first added. The device ID is then used in other API calls to reference a given device.

A device also has a fixed type id "`dev_type".` Currently the following strings can be used for the `dev_type` field:

```
"EVO-IP"
"XBOX-IP",
"XBOX-USB",
"EVO-IPS",
"HC-IP",
"FC-IP",
"WHC-IP",
"IEVO",
"HC1",
"MFC",
"RAT",
"SCB",
"AFC",
"WB-IP",
"WC-IP",
"SHUT-XPRESS"
```

3.1. List (All/One) Assigned Device(s)

Gets a description of all prompter devices (ipmonitors, Xboxes, ipcontrollers etc) that are currently assigned to Winplus-IP or a specified device.

Request

```
GET api/v1/sysdevices
```

Or `GET /api/v1/sysdevices/{dev_id}`

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

    "code": 0,

    "message": "success",

    "devices": [

        {

            "dev_id": "212F0CEA-7DAC-4FA6-BD6C-BC41F23B9E4E",

            "dev_type": "EVO-IP",

            "dev_name": "Prompter1",

            "endpoint": "A4ADB80099F2",

            "ipaddress": "10.0.0.11",

            "dev_status": "connected",

            "dev_status_extra": "{}"

        },

        {….},

        {….}

    ]

}
```

Notes:

"dev_name" the configured friendly name of the device

"endpoint" the device's communications endpoint

"ipaddress" the resolved ip address of the device. Note that this field maybe empty if winplus has not yet resolved the ip address from the device endpoint.

"dev_status" is the device status and can be one of the follow string values:

```
        Unknown,
        Disconnected,
        Connecting,
        FailedToConnect,
        Connected,
        ConnectionLost,
        InUse,
        IncompatibleDevice,
        IncompatibleVersion,
        Disabled
```

"dev_status_extra" will only appear for devices that support extra status information such as the WC-IP. It is a JSON encoded string that will list various properties such as battery level

3.2. Assign a Device

Adds a device to Winplus.

Request

```
POST /api/v1/sysdevices

Content-Type:application/json;charset=UTF-8

{
    "dev_type": "EVO-IP",
    "endpoint": "A4ADB80099F2",
    "dev_name": "optional name"
    "dev_id": "id required for some devices"
}
```

Notes:

"endpoint" can be the hardware address of the device, a static ip address or the USB device path.

"dev_id" is only required for certain devices (WC-IP) and is a combination of the parent's id and a unique identifier (e.g. 49459870-af5b-4d20-a7ec-4f1f662ec4b5:181066035556046). The Discovered Devices endpoint dev_id field should be used to find out what that value is for a given device.

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    "code": 0,
    "message": "success",
    "dev_id": "212F0CEA-7DAC-4FA6-BD6C-BC41F23B9E4E",
    "dev_type": "EVO-IP",
    "dev_name": "Prompter1",
    "endpoint": "A4ADB80099F2",
}
```

Notes:

The GUID returned in the “dev_id” field is used in subsequent api calls to reference the given device.

3.3. Delete a device

Removes a device from Winplus.

Request

```
DELETE /api/v1/sysdevices/{dev_id}
```

Response

Standard response.

3.4. Device Configuration

Access to device specific configuration.

```
URI: /api/v1/sysdevices/{dev_id}/config/
```

3.4.1. Get Device’s Current Configuration

List all configuration parameters and their current values for a given device.

Request

```
GET /api/v1/sysdevices/{dev_id}/config/
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

    “code”: 0,

    “message”: “success”,

    “dev_id”: “a9e3c45b-ad6e-4b2d-b2a7-1a090e786ba9",

    "dev_serial_number": "12345",

    "dev_type": "HC-IP",

    "dev_configs": [

        {

            "group": "IScrollController",

            "property": "ButtonMap",

            "value":
"BJumpNext:JumpNextStory\r\nBJumpPrev:JumpPrevStory\r\nBJumpTop:JumpToTop\r\nBF
unc1:ToggleLivePrompt\r\nBFunc2:BlankScreen\r\n"

        },

        {

            "group": "IScrollController",
            "property": "CustomProfiler.Exponent",

            "value": "2"

        },

       . . .

    ]

}
```

Notes:

`“dev_id” – is the device id`

`“dev_configs” – is the list of configs currently set on the device, there will be one entry per group/property/value triplet. The group is the area or interface that the config belongs to, examples are IScrollController for handling scrolling and IClockModule for handling of the clock on Prompters. The property item is the config item in that group and value is the current value of that property. All values will be returned as strings but may represent complex types.`

3.4.2. Set a Device’s Configuration

Sets configuration parameters for a given device.

Request

```
POST /api/v1/sysdevices/{dev_id}/config/

Content-Type:application/json;charset=UTF-8

{

    “dev_id”: “a9e3c45b-ad6e-4b2d-b2a7-1a090e786ba9",

    "dev_serial_number": "12345",

    "dev_type": "HC-IP",

    "dev_configs": [

        {

            "group": "IScrollController",

            "property": "ButtonMap",

            "value":
"BJumpNext:JumpNextStory\r\nBJumpPrev:JumpPrevStory\r\nBJumpTop:JumpToTop\r\nBF
unc1:ToggleLivePrompt\r\nBFunc2:BlankScreen\r\n"

        },

        {

            "group": "IScrollController",

            "property": "CustomProfiler.Exponent",

            "value": "2"

        },

        . . .

    ]
}
```

Response

Standard response.

Notes:

`“dev_id” - is the device id for the item being set, this must match in the url/payload`

`“dev_serial_number” & “dev_type” - you can also supply a serial number/device type which will also be matched and updated`

`“dev_configs” - is the list of configs to set on the device, this is as per 3.4.1 Get Device’s Current Configuration. Not all items need to be present for a device, any missing items will be left as currently set.`

3.4.3. Video Source Switching

Not currently implemented. Selects an input video source for a prompting device.

config uri = `video/src/`

Request

```
POST /api/v1/sysdevices/{dev_id}/config/video/src/

Content-Type:application/json;charset=UTF-8

{

    “video_source”: “IP”

}
```

Notes:

```
“video_src” can be one of the follow string id’s:

      IP – ip input

      SDI1 – SDI input 1
```

Response

Standard response.

3.5. Take control of devices

Allows you to take control of an individual device or all devices simultaneously

3.5.1. Take control of a specific device

Request

```
POST /api/v1/sysdevices/{dev_id}/takecontrol
```

Response

Standard response.

Notes:

`“dev_id” the device id which Winplus-Ip should take control of`

3.5.2. Take control of all devices

Request

```
POST /api/v1/sysdevices/takecontrol
```

Response

Standard response.

4. Prompting Operations

```
URI: /api/v1/prompt
```

4.1. Get Prompting Status

Returns the current prompt status

Request

```
GET /api/v1/prompt
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

    “code”: 0,

    “message”: “success”,

    “prompting”: 1,

    “captioning”: 1,

    “story_id”: “10293”

}
```

Notes:

```
“prompting” – 1 means prompt output is currently live, 0 means not live

“captioning” – 1 = captioning on, 0 = captioning off, -1 not available (either
no captioner available or prompting is off)

“story_id” – indicates the id of the story where the cue marker is.
```

4.2. Enable/Disable Prompt Output

Switch prompt output on/off.

Request

```
POST /api/v1/prompt

Content-Type:application/json;charset=UTF-8

{

     “prompt”: 1 or 0

     “jump_to_top”: 1 or 0
```

```
    "jump_to_story": "story id"
}
```

Response

Standard Response

Notes:

"prompt" integer 1 or 0 to enable/disable the prompt output

"jump_to_top" integer 1 or 0 to make the prompt output jump to the top of the run order.

"jump_to_story" a string story id instructing the prompter to jump to the top of a story.

4.3. Control Prompting Actions

Perform actions similar to controls on the prompter for example moving between stories or blanking the prompter.

Request

```
POST /api/v1/prompt/action

Content-Type:application/json;charset=UTF-8

{

    "action": "top"

}
```

Response

Standard Response

Notes:

"action" one of the following actions:

- "top" to jump to the top/first story
- "next" to jump to the next story
- "prev" to jump to the previous story
- "blank" to toggle the blanking of the prompter
- "caption" to toggle captioning on/off if available, you will need a captioner attached and prompting will need to be live to toggle

5. Runorder Operations

URI: /api/v1/ro

5.1. List available Runorder sources

List all available sources of runorders in the system.

Request

```
GET /api/v1/ro/
```

Response

```
Content-Type:application/json;charset=UTF-8
{
    “code”: 0,
    “message”: “success”,
     “sources” : [
          {
               “source_name”: “Newsroom name”
               “source_type” : “INEWS”
               “source_id”: “212F0CEA-7DAC-4FA6-BD6C-BC41F23B9E4E”
               “source_active”: “1”
          }
          {….}
     ]
}
```

Notes:

```
“source_type” can be one of the following string values:
     INEWS
     MOS
     ENPSSHAREDDIRECTORY
“source_name” string name of the newsroom.
“source_id” string ID of the newsroom.
“source_active” int, 1 if active and available, 0 if unavailable
```

5.2. List available Runorders

List available runorders for a particular source and location. Some sources are flat and will provide all runorders available when examining the root node of the source (see source path in notes). Other sources are tree like and will provide either runorders our further nodes than can be further explored.

Request

```
GET /api/v1/ro/{source_id}/{source_path}
```

Response

```
Content-Type:application/json;charset=UTF-8

{

     "code": 0,

     "message": "success",

      "runorders" : [

           {

                "ro_id": "854BE1AC-74C9-4EEB-A160-6783E4475DBA"

                "ro_name" : "SHOW.10PM.RUNDOWN"

                "ro_type" : "RUNORDER"

           }

           {

                "ro_id": "123BE1AC-74C9-4FF3-A160-6783E4475AAC"

                "ro_name" : "SHOW.10PM.MORE"

                "ro_type" : "DIRECTORY"

           }

           {….}

      ]

}
```

Notes:

"source_id" Mandatory string ID of the newsroom.

"source_path" Optional string. If not present it returns the available items in the root of the runorder source.

"ro_id" string id of the runorder.

"ro_name" string name of the rundown.

"ro_type" string value denoting type of resource, one of the following:

    DIRECTORY

    RUNORDER

5.3. Load a Runorder

Loads a runorder into the prompter.

Request

```
POST /api/v1/ro/

Content-Type:application/json;charset=UTF-8

{

      "ro_id": "854BE1AC-74C9-4EEB-A160-6783E4475DBA"

}
```

Response

```
Standard Response
```

6. Discovering Devices

Auto discoverable prompt devices that a Winplus machine can find on the network can be reported vai the API. The amount of devices found and their details can change over time as devices connect and disconnect.

6.1. List All Discovered Devices

Request

```
GET api/v1/discovered/devices/
```

Response

```
HTTP/1.1 200 OK
Content-Type:application/json;charset=UTF-8
{
    “code”: 0,
    “message”: “success”,
    “devices”: [
        {
            “dev_id”: null,
            “dev_type”: “EVO-IP”,
            “dev_name”: “Prompter1”,
            “endpoint”: “A4ADB80099F2”,
            “ipaddress”: “10.0.0.11”
        },
        {….},
        {….}
    ]
}
```

7. System Configuration

Access to additional system wide configuration.

7.1. Redundancy Pairing

This call takes at least five seconds to complete and will list all other instances of Winplus-IP that are also looking for redundancy peers. All instances that wish to auto-discover must be performing this call at the same time for them to find each other.

```
URI: /api/v1/sysconfig/pairing
```

To setup, and query redundancy pairing.

7.1.1. Query the Available Peer Machines

Request

```
GET /api/v1/sysconfig/pairing/peers
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

    "code": 0,

    "message": "success",

    "peers" : [

           { "peer_id" : "5FA272B5-67A3-4AC9-BC71-44568BF05D58" }

           {..}

           {..}

      ]

}
```

7.1.2. Pair Winplus

To pair Winplus with a peer machine to setup redundancy. The peer Id to connect to can be obtained either through auto-discoverery as documented in section 7.1.1 or more simply by maintaining lists of connected node Ids through the use of status request in section 7.1.3. To unpair, leave peer_id blank.

Request

```
POST /api/v1/sysconfig/pairing

Content-Type:application/json;charset=UTF-8

{

     "peer_id" : "5FA272B5-67A3-4AC9-BC71-44568BF05D58"

}
```

Response

Standard Response.

7.1.3. Get Pairing/Redundancy Status

Adding /long to the uri is optional and gives the long polling variant.

Request

```
GET /api/v1/sysconfig/pairing{/long}
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    "code": 0,
    "message": "success",
    "peer_id" : "5FA272B5-67A3-4AC9-BC71-44568BF05D58"
    "paired_peer_id" : "AE522B5-67A3-4AC9-BC71-44568BFFF000"
    "paired_ip_or_hostname" : "192.168.1.100"
    "connection_status":
    "standby_status":
}
```

Notes:

"paired_peer_id" is the id of the machine that this machine is paired to. It will be empty if unpaired.

"paired_ip_or_hostname" is the ip address of the paired peer if known or the hostname otherwise.

"connection_status" one of the following string values:

    Unpaired
    Connected
    Connecting
    Error

"standby_status" one of the following string values:

    Unpaired
    Active
    Passive
    Swap
    Error

7.1.4. Take Control / Become the active machine

This will cause a passive Winplus-IP to try and become the active machine of the pair by taking control. If the machine is already active it will return a success.

Request

```
POST /api/v1/sysconfig/pairing/takecontrol
```

Response

Standard Response.

8. Configurations

Methods for viewing and applying Winplus-IP configurations

8.1. Querying Configurations

These methods can be used to query information about the available configurations within Winplus-IP

8.1.1. Getting all configurations

Request

```
GET /api/v1/configuration
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    “configurations” : [
            {
                 “conf_id” : “5FA272B5-67A3-4AC9-BC71-44568BF05D58”,
                 “conf_parent_id” : null,
                 “conf_name” : “Parent”
            },
            {
                 “conf_id” : “7d74724e-0c8c-41ae-884a-bdb4623bb5f6”,
                 “conf_parent_id” : “5FA272B5-67A3-4AC9-BC71-44568BF05D58”,
                 “conf_name” : “Child”
            }
      ],
    “code”: 0,
    “message”: “success”,
}
```

Notes:

“conf_id” the configuration id

“conf_parent_id” the id of the parent configuration, if null then it is a top-level configuration

“conf_name” the name of the configuration

8.1.2. Getting a singular configuration

Request

```
GET /api/v1/configuration/{conf_id}
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    “configurations” : [
            {
                 “conf_id” : “7d74724e-0c8c-41ae-884a-bdb4623bb5f6”,
                 “conf_parent_id” : “5FA272B5-67A3-4AC9-BC71-44568BF05D58”,
                 “conf_name” : “Child”
            }
      ],
    "code": 0,
    "message": "success",
}
```

Notes:

```
"conf_id" the configuration id that you wish to query / is queried
```

```
"conf_parent_id" the id of the parent configuration, if null then it is a top
level configuration
```

```
"conf_name" the name of the configuration
```

8.1.3. Getting history of applied configurations

Use the optional /long in the URI to use the long poll form. In this case it will only return if the overall configuration changes.

Request

```
GET /api/v1/configuration/active{/long}
```

Response

```
HTTP/1.1 200 OK
Content-Type:application/json;charset=UTF-8
{
    "configuration" :
            {
                 "conf_id" : "5FA272B5-67A3-4AC9-BC71-44568BF05D58",
                 "conf_parent_id" : null,
                 "conf_name" : "Parent"
            }
    "addedremoved" : [
            {
                 "conf_id" : "1114724e-0c8c-41ae-884a-bdb4623bb222",
                 "conf_parent_id" : "5FA272B5-67A3-4AC9-BC71-44568BF05D58",
                 "conf_name" : "Room A"
                 "action" : "added"
            }
      ],
    "code": 0,
    "message": "success",
}
```

Notes:

`“conf_id” the configuration id`

`“conf_parent_id” the id of the parent configuration, if null then it is a top-level configuration`

`“conf_name” the name of the configuration`

`“configuration” is the configuration that was last loaded - either fully or partially`

`“addedremoved” is a list of configurations where the devices from that configuration have either been added or removed to/from the current configuration. There are in time sequence.`

`“action” is either “added” or “removed” depending on whether the devices from the configuration were added or removed.`

8.1.4. Getting currently applied configurations

Use the optional /long in the URI to use the long poll form. In this case it will only return if the overall configuration changes.

Request

```
GET /api/v1/configuration/ activestate{/long}
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    “base_configuration” :
            {
                “conf_id” : “5FA272B5-67A3-4AC9-BC71-44568BF05D58”,
                “conf_parent_id” : null,
                “conf_name” : “Parent”
                “state” : “full”
            }
    “merged_configurations” : [
            {
                “conf_id” : “1114724e-0c8c-41ae-884a-bdb4623bb222”,
                “conf_parent_id” : “5FA272B5-67A3-4AC9-BC71-44568BF05D58”,
                “conf_name” : “Room A”
                “state” : “partial”
            }
     ],
    “code”: 0,
    “message”: “success”,
}
```

Notes:

"conf_id" the configuration id

"conf_parent_id" the id of the parent configuration, if null then it is a top-level configuration

"conf_name" the name of the configuration

"base_configuration" is the configuration that was last loaded - either fully or partially. If no configuration is loaded the various conf_* parameters will be empty and state is "empty"

"merged_configurations" is a list of configurations where the devices from that configuration have either been added to the current configuration. They will only report a state of either "full" or "partial".

"state" will be one of the following:

"full": when the configuration is completely present and loaded

"partial": when part of the configuration has been removed (i.e. somebody has manually removed a device

"empty": if no config is loaded (only for base_configuration – refer to that item).

8.2. Applying configurations

This method can be used to trigger Winplus-IP to load and apply the selected configuration.

Request

```
POST /api/v1/configuration/{conf_id}/apply

Content-Type:application/json;charset=UTF-8

{

      "partial": 1

}
```

Response

Standard Response.

Notes:

"conf_id" the configuration id to apply. Use a conf_id of "blank" (without quotes) with partial set to 0 to remove all loaded config.

"partial" If set to 0 then the full hierarchy of configurations will be applied with inheritance (items in child configurations will override settings in parent configurations) and any missing items will be reset to defaults. If "partial" is set to 1 then only that specific configuration will be loaded, and any missing configuration areas will be left as set before the command was initiated.

8.3. Merging and removing devices

These methods allow you to merge in the devices from a configuration or remove the devices in a configuration. This is different from applying a configuration where that will give you exactly the devices contained in that configuration. This method will add/merge (or remove) the devices in the selected configuration to whatever is currently active in Winplus-IP. If a device is already loaded and a configuration is merged in that contains that device, the device will be updated with the settings contained in the configuration.

8.3.1. Merging Devices

Request

```
POST /api/v1/configuration/{conf_id}/devices
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
    "devices" : [
            {
                 "dev_id" : "7d74724e-0c8c-41ae-884a-bdb4623bb5f6",
                 "success" : 0
            },
      ],
    "code": 0,
    "message": "success",
}
```

Notes:

```
"conf_id" the configuration id which contains devices that you wish to merge in
"dev_id" the device id of a device that is being added
"success" if 0 then failed to add, 1 means success
```

8.3.2. Removing Devices

This will remove the devices contained in a configuration. You will not receive a failure if a device is already not present.

Request

```
DELETE /api/v1/configuration/{conf_id}/devices
```

Response

Standard Response.

Notes:

```
"conf_id" the configuration id which contains devices that you wish to remove
```

9. Status

9.1. Overview of status

Use the optional /long in the URI to use the long poll form of status. In this case it will only return if the overall status changes.

Request

```
GET /api/v1/status{/long}
```

Response

```
HTTP/1.1 200 OK
```

```
Content-Type:application/json;charset=UTF-8

{

      "overall_status": "good"

      "components" : [

            {

                  "comp_name" : "prompters",

                  "comp_status" : "good"

            }

      ],

     "code": 0,

     "message": "success",

}
```

Notes:

`"overall_status"` an aggregate status of all components, can be good or bad, it will be bad if any component is reporting a bad status otherwise it will report good.

`"comp_name"` the name of the individual component reporting status. Supported components are (may be increased in the future):

```
      Prompters,

      Controllers,

      Captioners,

      Newsrooms,

      Redundancy,

      Autoprompt
```

`"comp_status"` one of either good, bad or none. If component is not in use or is disabled, then none will be reported otherwise it will report bad if any element in that component is not working. For example, a single failed to connect prompter will change the overall status to bad. If all is working as normal then the component will report good.

9.2 Overview of status for a particular component

This is used to get the status for an individual component. As with before you can optionally append /long to the uri to get the long poll form

Request

```
GET /api/v1/status/{comp_name}{/long}
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

      "status": "good",
      "comp_name": "Prompters",
      "code": 0,
      "message": "success",
}
```

Notes:

`"status"` The status of the requested component

`"comp_name"` the name of the individual component reporting status. Supported components are (may be increased in the future):

```
      Prompters,
      Controllers,
      Captioners,
      Newsrooms,
      Redundancy,
      Autoprompt
```

10. Logging

10.1. Export logs

This will start an export of the Winplus-IP logs to the requested folder. While the operation is in effect Winplus-IP will show the message "Exporting logs…" in the status bar. The logs will not be zipped up. The copying process can take a while, the methods in 10.2 and 10.3 can be used to query their progress.

Request

```
POST /api/v1/logging/export
Content-Type:application/json;charset=UTF-8
{
      "export_to": "\\network\folder"
}
```

Response

```
HTTP/1.1 200 OK
Content-Type:application/json;charset=UTF-8
{
      "log_id": "7d74724e-0c8c-41ae-884a-bdb4623bb5f6",
      "code": 0,
      "message": "success",
}
```

Notes:

`"export_to"` should be either a local folder or a network directory

`"log_id"` is the id of the export process

10.2. Export logs progress

This method can be used to query the copying processes currently in progress. Completed and errored operations will not be listed.

```
Request

GET /api/v1/logging/export/

Response

HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
      "log_folders" : [
            {
                  "log_id" : "7d74724e-0c8c-41ae-884a-bdb4623bb5f6",
                  "log_folder" : "c:\local\folder",
                  "log_progress" : "In progress"
            },
      ],
      "code": 0,
      "message": "success",
}
```

Notes:

"log_id" is the id of the copying process

"log_folder" the location where the logs are being exported to

10.3. Export specific log progress

This method can be used to query the progressing of a specific export logs processes. As with before you can optionally append /long to the uri to get the long poll form

```
Request

GET /api/v1/logging/export/{log_id}{/long}

Response

HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{
      "log_folders" : [
            {
                  "log_id" : "7d74724e-0c8c-41ae-884a-bdb4623bb5f6",
                  "log_folder" : "\\network\folder",
                  "log_progress" : "In progress"
            },
      ],
      "code": 0,
      "message": "success",
}
```

Notes:

"log_id" is the id of the copying process

"log_folder" the location where the logs are being exported to

"log_progress" will either be: completed, in progress or if the export process fails, the reason for the failure

11. Commands

11.1. Getting available commands

Gets a list of all available commands and their requirements

Request

```
GET /api/v1/commands
```

Response

```
HTTP/1.1 200 OK
Content-Type:application/json;charset=UTF-8
{
      "commands" : [
            {
                  "name" : "CommandName",
                  "requires_parameter" : "true"
            }
      ],
    "code": 0,
    "message": "success",
}
```

Notes:

"name" the name of the command that can be executed

"requires_parameter" indicates whether the command requires a parameter when being triggered.

11.2. Executing a command

Executes a command

Request

```
POST /api/v1/commands
Content-Type:application/json;charset=UTF-8
{
      “name”: “CommandName”,

      “parameter”: “Optional parameter value”


}
```

Response

```
HTTP/1.1 200 OK

Content-Type:application/json;charset=UTF-8

{

      “code”: 0,

      “message”: “success”,

}
```

Notes:

“name” the name of the command that is to be executed

“parameter” an optional element that is only required for a subset of commands. You can use the get commands endpoint to find out if this is required or not. The parameter’s contents (if required) differs per command for example the load configuration commands require a configuration ID.

12. Error Codes

Code	Message	Description
0	Success	Success
1	Server Error	Generic error occurred
2	Winplus-IP is not ready	Winplus-IP has not finished starting up and requires more time to start up.
3	Item was not found	When performing an action on a specific item, this error denotes the situation where the specified item does not exist
4	Failed to do action	The commit of data to the internal Winplus-IP managers failed.
5	Message does not contain a body or body is not formatted correctly. See API documentation for expected body contents	See message, you will need to supply the correct JSON body to use that endpoint
6	Invalid item type	Using a non-support or invalid type when doing an operation e.g. using a device type of “NOT A DEVICE TYPE” when adding a device
7	Your licence does not support this action	Your Winplus-IP licence does not support the request you initiated. You will need to upgrade it to allow the call to succeed.