Application Programming Interface (API)

The API section is a comprehensive resource that provides detailed documentation and guidance on utilizing the FileWave API. Whether you're a developer or an IT professional looking to integrate FileWave functionality into your own applications or workflows, this section offers valuable insights. Discover comprehensive API reference documentation, code samples, and tutorials that walk you through the process of leveraging the FileWave API. By leveraging the API, you can automate tasks, streamline workflows, and integrate FileWave functionality seamlessly into your existing systems. Unlock the full potential of FileWave by harnessing the power of the API and customizing your FileWave deployment to suit your specific needs.

What is an API?

What

Application Programming Interface (API). APIs programmatically provide the reading or writing of information to one or more services.

FileWave has a rich set of APIs to allow customers to build connections. Some vendors also produce integrations, utilising FileWave API, without needing to work with the API directly.

When/Why

APIs extend the capability and integration of FileWave with other Systems.

How

The articles linked to this article are a good starting point. Professional Services can also help with learning and using APIs.

- Command Line API (v1)
- Anywhere API (v2)
- How do I export the results of an Inventory query?
- Using the RESTful API to limit, sort, and offset values returned

An important note on URLs

FileWave has 2 APIs

History

For inventory purposes, FileWave has a Command Line API, designed to interact with Inventory Queries. This API has existed for years and can work on either port 20443 or port 20445.

Since the introduction of the FileWave web admin, 'FileWave Anywhere', a new API was created. FileWave Anywhere interacts with the FileWave Server using this new API. Unless configured otherwise, FileWave Anywhere works over port 443.

The Web API has its own paths, when compared with the Command Line version (note the extra /api at the beginning of the path. Below is an example, which creates a new Inventory Query, using both APIs.

TCP port 443 (default port if not specified in URL). Notice that it begins with /api/inv/api/

```
curl -s -H "Authorization: e2M2sssYjIwLTxxx1hMzdiLTFmyyyGIwYTdjOH0="
https://myserver.filewave.net/api/inv/api/v1/query/ \
    --header "Content-Type: application/json" -X POST -d @ios16_incompatible.json
```

TCP port 20445 (could also use port 20443). Notice that it begins /inv/api/

```
curl -s -H "Authorization: e2M2sssYjIwLTxxx1hMzdiLTFmyyyGIwYTdjOH0="
https://myserver.filewave.net:20445/inv/api/v1/query/ \
    --header "Content-Type: application/json" -X POST -d @ios16_incompatible.json
```

Since the Web API is used for all server interaction, strictly speaking anything that can be achieved in FileWave Anywhere may also be programmed. As such, the scope of the Web API is broader in the main than that of the Command Line version.

The Web API has both public and private URLs. Private URLs can be changed, between FileWave versions, at any time without warning or notification and as such should be avoided in scripts and other systems interacting with FileWave. They are easily recognisable by the word 'internal' in their URL paths, as per below:

```
https://${server_dns}/filewave/api/devices/internal/devices/${filewave_id}/details/custom_fields/fields
```

Working With APIs

Getting Started

The purpose of this guides is two fold:

- First, to provide an introduction to those unfamiliar with using the FileWave API
- Second, to be a reference for commands that can be used within the API

Command Line API refers to the original RESTful API. Recognisable both by the port used and URL paths commencing:

- /inv/

FileWave Anywhere API v2 refers to the newer web admin, FileWave Anywhere, API and recognisable by paths commencing:

- /api/

Purpose of an API

As described, APIs are designed to communicate with systems. As such, FileWave may be leveraged by other systems, e.g. SCCM data engines, in-house databases, etc. To maintain security, there must be an authentication method to allow such communication to take place. The API provides this kind of ability to provide in-depth integration on an as-needed basis.

Basic, non-specific command line examples:

macOS shell

```
curl -s -H "Authorization: $auth" \
    https://$server_dns:20445/inv/api/v1/query_result/ \
    --data $query \
    -H "Content-Type: application/json"
```

Windows PowerShell

```
$header = @{Authorization="$auth"}

Invoke-RestMethod -Method POST \
    -Headers $header \
    -ContentType application/json \
    -Uri https://$server_dns:20445/inv/api/v1/query_result/ \
    -Body $query
```

The macOS 'Authorization' or Window's 'Headers' make use of base64 tokens. From the above code, $auth would need to be set as one of these tokens.

> Each user generated will have their own unique base64 token automatically generated. This token can be revoked and new tokens created. Each user may have multiple tokens.

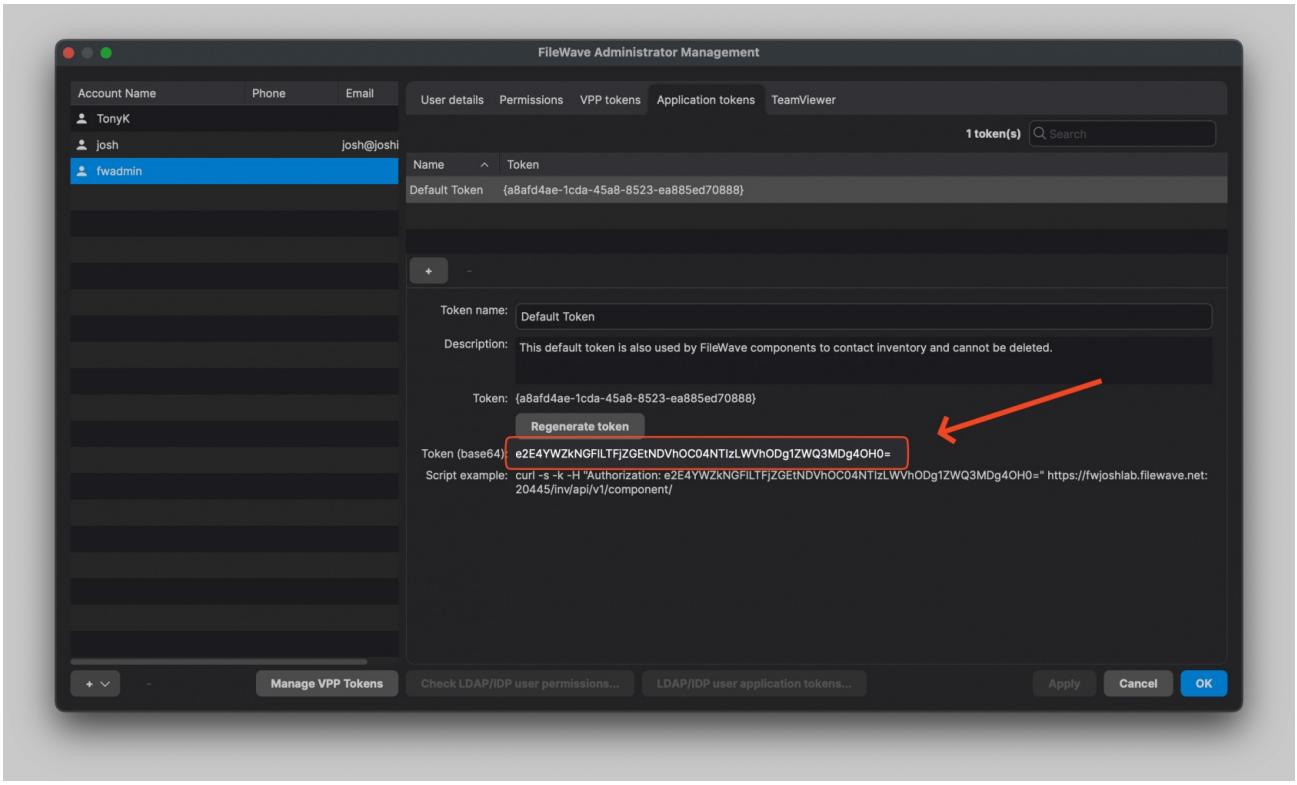
The Token is the Key

> The token allows access to any API calls from anything that has access to the FileWave Server. It should not be shared unnecessarily and otherwise should be kept secret.

Tokens may be viewed from FileWave Central App:

- Assistants > Manage Administrators

Select a desired Administrator and choose the 'Application Tokens' tab. The token value to copy is the 'Token (base64)'



Token value from above image:

```
e2E4YWZkNGFlLTFjZGEtNDVhOC04NTIzLWVhODg1ZWQ3MDg4OH0=
```

When choosing accounts and tokens for API interaction, consider making dedicated users for the task(s) required. Set the permissions of that user to limit their ability to the required task alone. If the token is compromised, this will limit not only the scope of how that token could be used by an attacker, but when revoking and generating a new token, minimal impact would be experienced.

For more information on the Application Tokens see the page: Managing FileWave Administrators (Application Tokens)

API Requests

Requests could be one of:

Command Type	Description
GET	Returns a resource value
PUT	Replaces a resource
PATCH	Updates a resource
POST	Creates a resource
DELETE	Removes a resource

Data sent with requests are in the form of a JSON, as are the responses from those requests.

JSON data is broken into keys/value pairs, where values could even be lists inside of lists.

Examples

Below examples are using a python tool to reformat the returned response. Python must be installed to benefit from this. If not, remove the piped section of the command or instal Python.

For example, actioning a GET to list of all FileWave Server Inventory Queries, could return a JSON similar to the below:

List Existing Inventory Queries

1) Get all Queries

```
curl -s  -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
    https://myserver.company.org:20445/inv/api/v1/query/ | python3 -mjson.tool
```


```
[
    {
        "id": 1,
        "name": "All Windows",
        "favorite": true,
        "group": 1,
        "version": 1
    },
    {
        "id": 2,
        "name": "Mac OS X 10.7-10.11",
        "favorite": false,
        "group": 1,
        "version": 5
    },
...
    {
        "id": 103,
        "name": "All Computers to retire",
        "favorite": false,
        "group": 3,
        "version": 2
    }
]
```


Key	Value	Description
`id`	`integer`	The unique number for the query. To be used as reference
`name`	`string`	The name given to the query
`favorite`	`true/false`	Whether or not the query should show in the sidebar
`group`	`integer`	The group number given. built-in queries – for example – would be in the "Sample Queries" group, which is group 1. If the user made new groups
`version`	`integer`	The version for the query. How many times has the query been altered and saved, starting with 1

Return the definition of a chosen query

The query definition, not the results of the query. Using ID 1 as an example:

2) Get Query

```
curl -s -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
    https://myserver.company.org:20445/inv/api/v1/query/1 | python3 -mjson.tool
```


```
{
    "criteria": {
        "expressions": [
            {
                "column": "type",
                "component": "OperatingSystem",
                "operator": "=",
                "qualifier": "WIN"
            }
        ],
        "logic": "all"
    },
    "favorite": true,
    "fields": [
        {
            "column": "device_name",
            "component": "Client"
        },
```

```
        {
            "column": "filewave_client_name",
            "component": "Client"
        },
        {
            "column": "name",
            "component": "OperatingSystem"
        },
        {
            "column": "version",
            "component": "OperatingSystem"
        },
        {
            "column": "build",
            "component": "OperatingSystem"
        },
        {
            "column": "edition",
            "component": "OperatingSystem"
        }
    ],
    "main_component": "Client",
    "name": "All Windows",
    "id": 1,
    "version": 1,
    "group": 1
}
```

Key	Value	Description
`criteria`	`array`	Expressions and logic of query
	Criteria Expressions (Each entry will require all of the below. Add multiple entries to the array as required):	

Key	Value	Description
`column`	Multiple values e.g. 'version', 'device_id', etc.	Chosen search component (Figure 1.2 #1)
`component`	Multiple values e.g. 'Client', 'OperatingSystem', etc.	Group containing above component (Figure 1.2 #2)
`operator`	Multiple values e.g.'is', 'begins', etc.	Method of comparison (Figure 1.2 #3)
`qualifier`	Multiple values (Either a: String, Integer, Date or Boolean value)	Value for comparison (Figure 1.2 #4)

Logic:

Key	Value	Description
`logic`	Multiple values 'all', 'none' or 'one'	How Components should be logically considered for correct return of results (Figure 1.2 #5)

`favourite`	`true/false`	Show/Hide from FileWave Central sidebar Inventory Queries
`fields`	`array`	Which components will be shown (ordered first to last)
	Fields to display (Each entry will require all of the below. Add multiple entries to the array as required):	

Key	Value	Description
`column`	Multiple values e.g. 'device_name', 'name', etc.	Component to display (Figure 1.2 #1)

	`component`	Multiple values e.g. 'Client', 'OperatingSystem', etc.	Group containing above component (Figure 1.2 #2)
`main_component`	Selection Box	Important this is set correctly (Figure 1.2 #6)
`name`	`string`	Inventory Query name shown in FileWave Central
`id`	`integer`	Inventory Query unique number, not already in use. (Each query has a unique number, starting at 1 and incremented with each new query generated when actioned through FileWave)
`version`	`integer`	Increment by 1 for each alteration
`group`	`integer`	The Inventory Query group which the query should be displayed within. E.g. 'group' value of 1 would be within the 'Sample Queries' group

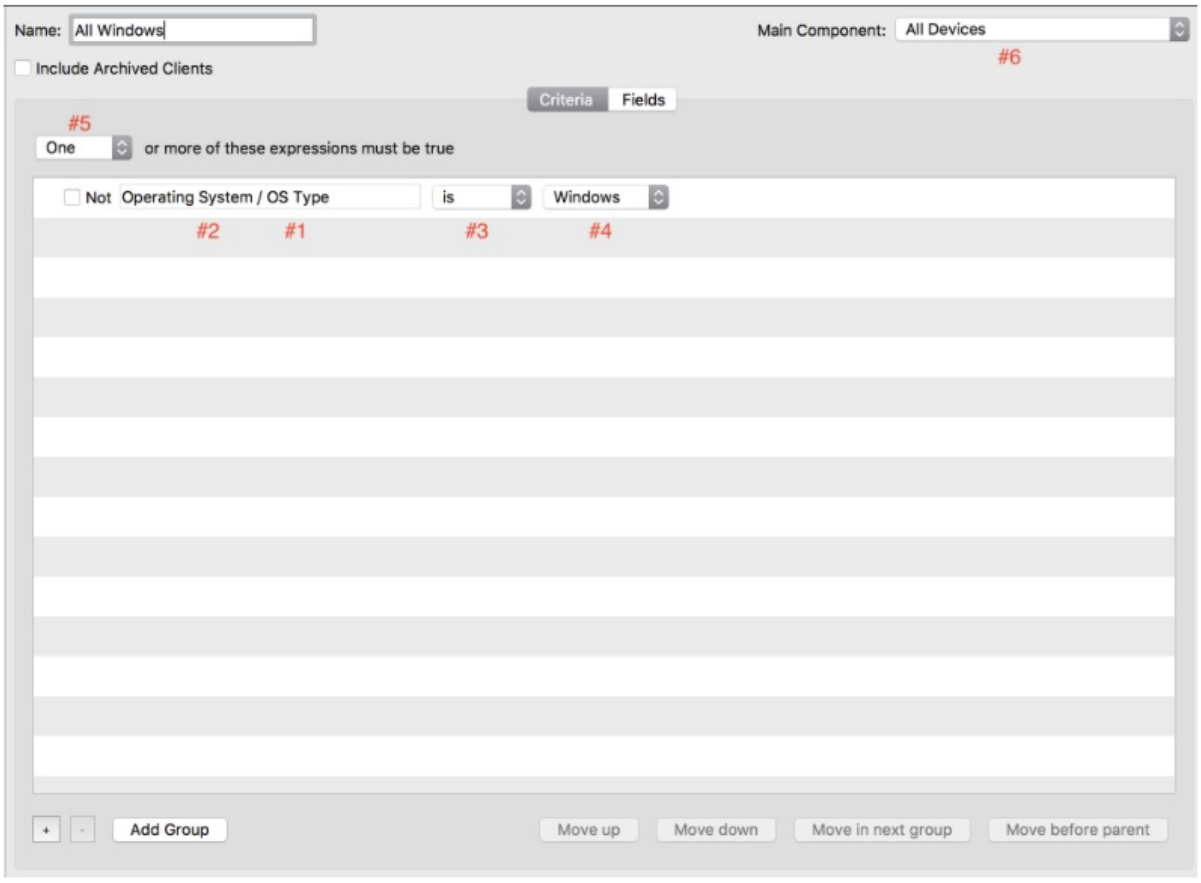


Figure 1.2 - Query Builder Criteria

Return the Inventory Query Results

3) Get Query Results

```
curl -s  -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
  https://myserver.company.org:20445/inv/api/v1/query_result/1 \
  | python3 -mjson.tool
```


```
{
    "total_results": 13,
    "filter_results": 13,
    "offset": 0,
    "values": [
        [
            "FW-BLUE-02",
```

```
            "FW-Blue-02",
            "Windows 10.0",
            "10.0.0",
            "10240",
            "Microsoft Windows 10 Home"
        ],
        [
            "LAPTOP-C6LLFGH6",
            "FH-History3",
            "Windows 10.0",
            "10.0.0",
            "14393",
            "Microsoft Windows 10 Home"
        ],
...
    ],
    "version": 3
}
```

Key	Value	Description
`total_results`	`integer`	Total count of results
`filter_results`	`integer`	
`offset`	`integer`	
`values`	`array`	The results. Repeated for each result. Items depends on what your specified in the fields
`version`	`integer`	The version for the query. How many times has the query been altered and saved, starting with 1

JSON

Verify JSON Formatting:

- https://jsonlint.com/

API Application

- Postman https://www.getpostman.com/ (macOS, Windows, and Linux)

Browser Extensions

- Mod-Header https://mod-header.appspot.com/ Will allow you to use the Google Chrome Browser to view and interact with the FileWave API

Commands

Remember: All URLs start with

```
https://myserver.company.org:20445/inv/api/v1/
```

Must include the authorization header

Below are the possible options:

URLs

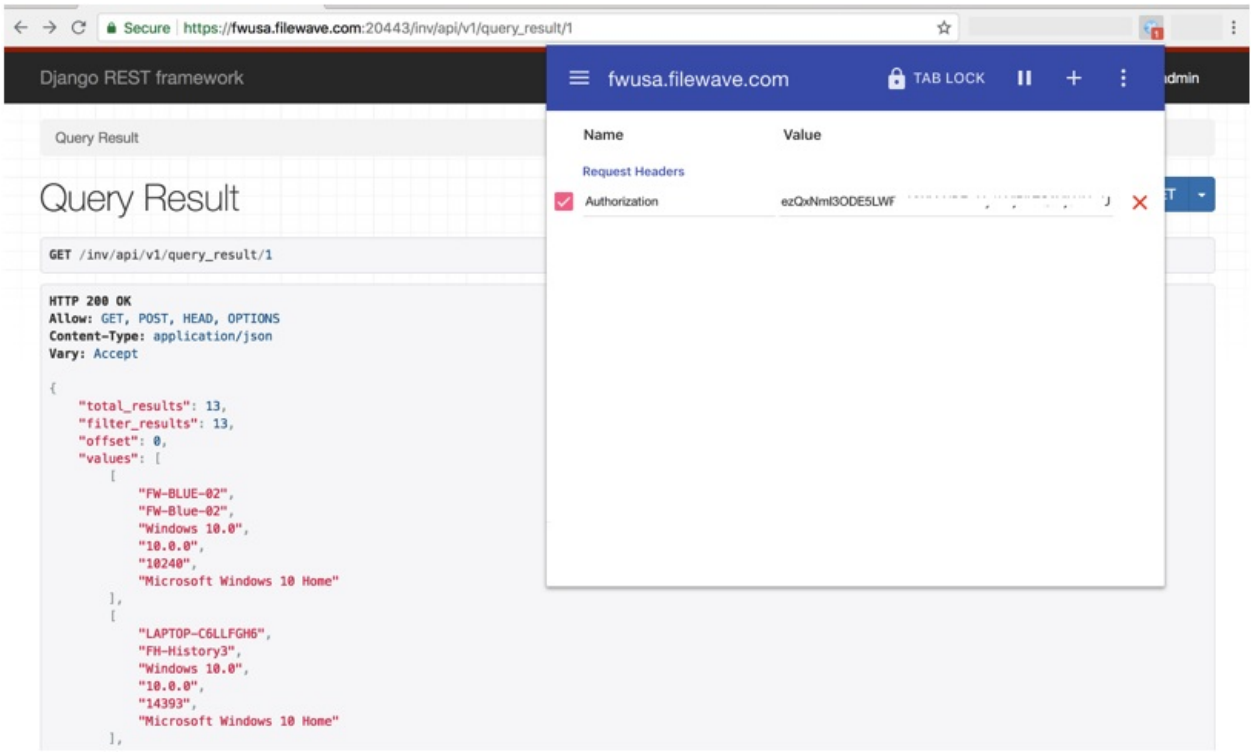
URL	Use	Options
	Inventory	
`query`	Show all queries	`GET POST`
`query/#`	Show information on a single query Where # is the query ID	`GET PUT DELETE`
`query_group/`	Show all query group	`GET POST`
`query_group/#`	Detail information on a single group Where # is the group ID	`GET PUT DELETE`
`query_result/#`	Show the results of one query	`GET POST`

	Where # is the query ID	
`query_count`		`POST`
`component`	Show all component options on your instance	`GET`
`field_type`	Show all fields on your instance	`GET`
License		
`license_definition`	Show all query	`GET`
`license_definition/#`	Show information on a single license Where # is the license ID	`GET`
Custom Fields		
`custom_field/`	Show all custom fields	
`custom_field/get_association`		`POST`
`custom_field/set_association`		`POST`
`custom_field/upload`		`POST`
`custom_field/usages/<Field_Name>`	Where <Field_Name> is the Internal Name (E.G "battery_cycle_count")	`GET`
`custom_field/values/`		`POST`
`custom_field/edit/`		`POST`

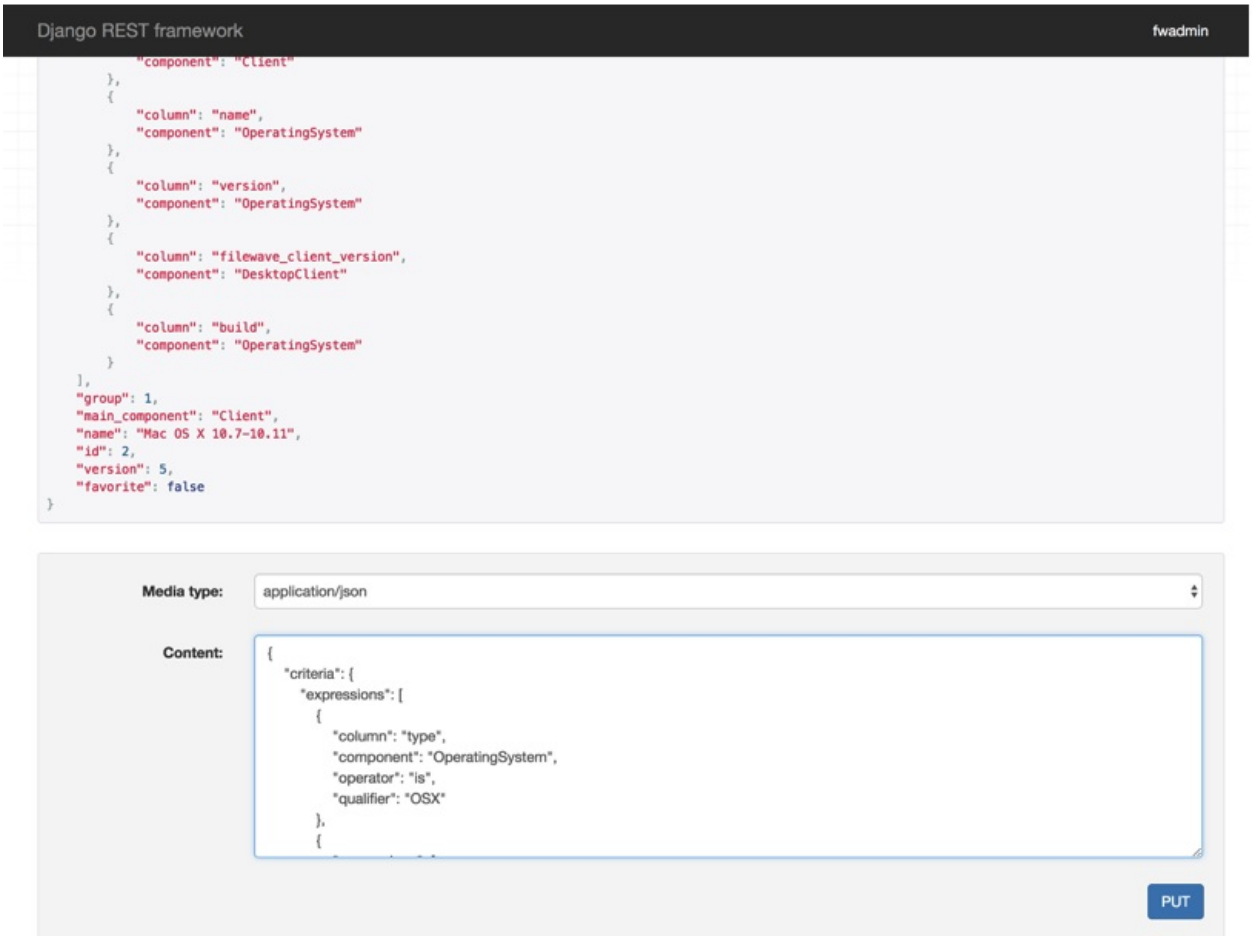
Examples

Using a browser extension

Using Mod-Header (see tools section), you can make Chrome a RESTful API browser by taking advantage of the FileWave Django Framework. Leveraging URLs and authorisation token to return Query Results.



Even if the URL is typically a POST, it provides an output similar to the following



Using the curl command

Viewing all available queries (GET)

```
curl -s -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
  https://myserver.company.org:20445/inv/api/v1/query/ \
  | python3 -mjson.tool
```

Posting a new query (POST)

```
curl -s -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
  --header "Content-Type: application/json" \
  -X POST \
  -d @<path/name of new query.json> \
  https://myserver.company.org:20445/inv/api/v1/query/
```

Removing a query (DELETE)

```
curl -s -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
  -X DELETE https://myserver.company.org:20445/inv/api/v1/query/<id#>
```

For more curl help, see: Using the RESTful API to limit, sort, and offset values returned

Self-Signed Certificates

Hopefully everyone is using official certificates. However, if the FileWave Server does have a self-signed certificate, the above commands should fail. To ignore the warnings the following is required.

macOS

Add the -k option to the command. E.g.

```
curl -s -k -H "Authorization: e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=" \
  https://myserver.company.org:20445/inv/api/v1/query/ \
  | python3 -mjson.tool
```

Windows

PowerShell requires somewhat more code to ignore the warning. Add the below to the beginning of any script calling an API to a server with a self-signed certificate:

```
# Required for self-signed certs only
function Ignore-SSLCertificates
{
    $Provider = New-Object Microsoft.CSharp.CSharpCodeProvider
    $Compiler = $Provider.CreateCompiler()
    $Params = New-Object System.CodeDom.Compiler.CompilerParameters
    $Params.GenerateExecutable = $false
    $Params.GenerateInMemory = $true
    $Params.IncludeDebugInformation = $false
    $Params.ReferencedAssemblies.Add("System.DLL") > $null
    $TASource=@'
        namespace Local.ToolkitExtensions.Net.CertificatePolicy
        {
            public class TrustAll : System.Net.ICertificatePolicy
            {
                public bool CheckValidationResult(System.Net.ServicePoint
sp,System.Security.Cryptography.X509Certificates.X509Certificate cert, System.Net.WebRequest req, int problem)
                {
                    return true;
                }
            }
        }
'@
    $TAResults=$Provider.CompileAssemblyFromSource($Params,$TASource)
    $TAAssembly=$TAResults.CompiledAssembly
    ## We create an instance of TrustAll and attach it to the ServicePointManager
    $TrustAll = $TAAssembly.CreateInstance("Local.ToolkitExtensions.Net.CertificatePolicy.TrustAll")
    $AllProtocols = [System.Net.SecurityProtocolType]'Ssl3,Tls,Tls11,Tls12'
    [System.Net.ServicePointManager]::SecurityProtocol = $AllProtocols
    [System.Net.ServicePointManager]::CertificatePolicy = $TrustAll
}

Ignore-SSLCertificates
```

Using PHP

Saved as a php file (like inv.php), update the URL and auth code, then place the file on a web server where PHP has been enabled. This creates a 'view only' web page version of the inventory. People can enter the URL for the query, hit refresh as many times as they like and will always see the latest information in inventory. All without having to hassle IT for the latest data.

The output of the query results isn't fancy, but this is to illustrate what can be achieved.

▼ PHP Inventory Viewer

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd"><html xmlns="http://www.w3.org/1999/xhtml">
<html>
<?php
$baseurl="myserver.company.org";
$port="20445";
$authcode="e2FjYzRkYmQzLTI3ZjYtNDEyMi1iMGVhLTI1YmY0OGNmYWM0NX0=";


ini_set('display_errors', 'On');
### do not edit below ###
if (!isset($_GET["qid"])){
        $url = "https://".$baseurl.":".$port."/inv/api/v1/query/";
} else {
        $url = "https://".$baseurl.":".$port."/inv/api/v1/query_result/".$_GET["qid"];
}
//  Initiate curl
$ch = curl_init();
// Set the url
```

```
curl_setopt($ch, CURLOPT_URL,$url);
// Disable SSL verification
curl_setopt($ch, CURLOPT_SSL_VERIFYHOST, FALSE);
curl_setopt($ch, CURLOPT_SSL_VERIFYPEER, FALSE);
curl_setopt($ch, CURLOPT_SSLVERSION, 1);
// Will return the response, if false it print the response
curl_setopt($ch, CURLOPT_RETURNTRANSFER, TRUE);
//authenticate
curl_setopt($ch, CURLOPT_HTTPHEADER,array('Authorization:<'.$authcode.'>='));
// Display errors if any
if (curl_errno($ch)) {
   print curl_error($ch);
}
// Execute
$result=curl_exec($ch);
if (curl_errno($ch)) {
   print curl_error($ch);
}
$output=json_decode($result, true);
curl_close($ch);

//create function for looping unknown dimensional array
function printAll($a) {
  if (!is_array($a)) {
    echo $a, ' <br/>';
    return;
  }
echo "<br/>";
  foreach($a as $v) {
    printAll($v);
  }
}

// Start html Page
echo '<head>'
.'<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />'
.'<title>'.$baseurl.'Inventory Page</title>'
.'</head>'
.'<body>'
.'<style type="text/css">'
."body {font-family:'Helvetica Neue Light', 'Helvetica Neue', Helvetica, Arial, 'Lucida Grande', sans-serif;}"
.'h1,h2,h3,h4,{font-weight:100;}'
.'div {padding:10px; color:#fff; background:#333;}'
.'div.output{border:1px solid rgba(0,0,0,0.1); background: rgba(0,0,0,0.03); color:#555;-webkit-border-
radius:3px;border-radius:3px;margin:15px 25px; padding:10px;}'
.'tr:nth-child(even) {background: rgba(255,255,255,0.85);}'
.'tr:nth-child(odd) {background: rgba(0,0,0,0.05);}'
.'a, a:hover {color:#ce1300; text-decoration:none;}'
.'</style>'
.'<div><h1><img src="https://www.filewave.com/images/filewave-1024x788.png" height="60" />'.$baseurl.'
Inventory</h1></div>'
.'<br/>';

// Default homepage
if (!isset($_GET["qid"])){
        echo "<table>"
        ."<thead>"
        ."<tr>"
        ."<th>&hearts;</th>"
        ."<th>Query Name</th>"
        ."<th>Query ID</th>"
        ."</tr>"
        ."</thead>"
        ."<tbody>";
        foreach ($output as &$value) {
                if ($value['favorite'] == true) { $fav="&hearts;";} elseif ($value['favorite'] == false)
{$fav=" ";}
                echo "<tr><td>".$fav."</td><td>".$value['name']."</td><td><a href='".$_SERVER['PHP_SELF']."?
qid=".$value['id']."&n=".$value['name']."'>".$value['id']."</a></td></tr>";
        }
```

```
        echo"</tbody></table>";
}
//If an individual query has been selected
elseif (isset($_GET["qid"],$_GET["n"]))  {
        echo "<h3>Home &gt; Query: "
        .$_GET["n"]
        ."</h3><hr/>"
        ."<strong>Total Results: </strong>"
        .$output['total_results']
        ."<br/>"
        .'<strong>First Column results: </strong>';

        foreach ($output['values'] as &$value) {
                echo $value['0'].' <strong>   |   </strong>';
        }
echo "<br/>"
."<strong> All Results: </strong><br/><div class='output'>";
printAll($output['values']);
echo "</div>";
#var_dump($result);
}
else {
        echo "<h1 style='color:#ff0000;'> An error has occurred</h1> Parhaps you used a bookmark and the URL
has changed";
}
?>
<hr/>
<center><font style=" font-size:9px;"><a href="http://filewave.com" target="_blank">&copy; BenM@ FileWave</a>
</font></center>
</body>
</html>
```

Command Line API (v1)

Command Line RESTful API

It is probably easiest to consider this API as an interaction with Inventory Queries, allowing for example:

- Ad-hoc queries to be run to return results
- New Inventory Queries to be created
- The returning of Current Query results
- Deleting queries

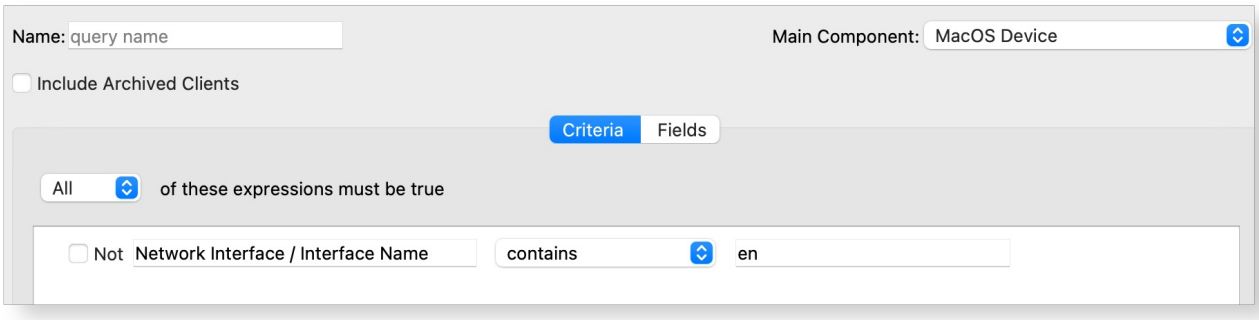
There is even the ability to read or alter Custom Field values for devices.

Examples:

Returning an Inventory Query request

FileWave GUI example

Imagine it is desirable to return a list of macOS devices which have a network interface name that contains 'en', the Field to be returned is the Device Name and the Main Component is 'MacOS Device'. The criteria would appear as below:



JSON Data Example

A RESTful API query, to return the same set of results, would have a JSON structure set out as:

```
{
    “criteria”: {
        “column":"interface_name",
        “component":"NetworkInterface",
        “operator”:”contains",
        “qualifier":"en"
    },
    “fields":[
        {
            “column”:"device_name",
            “component":"Client"
        }
    ],
    “main_component":"MacOSClient"
}
```

It can be seen that the JSON block is just mimicking the FileWave Central Inventory Query.

Creating an Inventory Query

On the KB page explaining what an API is, the following examples were shown:

FileWave Anywhere API:

```
curl -s -H "Authorization: e2M2sssYjIwLTxxx1hMzdiLTFmyyyGIwYTdjOH0="
https://myserver.filewave.net/api/inv/api/v1/query/ \
    --header "Content-Type: application/json" -X POST -d @ios16_incompatible.json
```

Command Line API:

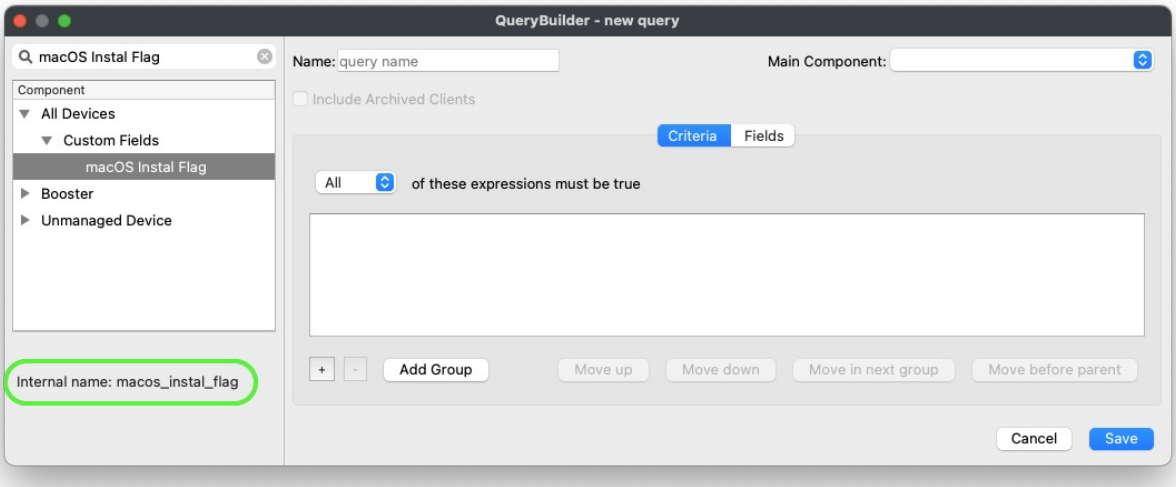
```
curl -s -H "Authorization: e2M2sssYjIwLTxxx1hMzdiLTFmyyyGIwYTdjOH0="
https://myserver.filewave.net:20445/inv/api/v1/query/ \
    --header "Content-Type: application/json" -X POST -d @ios16_incompatible.json
```

These demonstrate a method of creating a new Inventory Query by providing the JSON as a file, rather than text in the actual script line, using the POST option. Also note the subtle difference in URL path.

Interacting with Custom Fields

The first thing to appreciate, is all Inventory items have an 'Internal Name'. This is unique to each item and of course Custom Fields also therefore have their own Internal Name.

Internal Names may be viewed in FileWave Central when in the Inventory Query viewer. Select any item in the left hand sidebar and the bottom of the window will report the Internal Name.



Reading Values

Perhaps it would be desirable to read the Custom Field value of the above 'macOS Instal Flag' Custom Field for a device, for example, based upon its serial number. The JSON data block might look something like:

```
{
  "criteria":
    {
      "column":"serial_number",
      "component":"Client",
      "operator":"is",
      "qualifier":'\"$serial_number\"'
  },
  "fields":
    [
       {
        "column":"macos_instal_flag",
        "component":"CustomFields"
       }
  ],
  "main_component":"Client"
}
```

Using this method, it is possible to provide the Serial Number as a variable value; as such the same script could be ran on all devices, without the need to edit the script. Note the internal name used for the Custom Field as well as the single quotes around the $serial_number variable.

Editing Values

As an alteration of a current value, the command would then use the PATCH option. In this instance, the Custom Field to alter has an 'internal name' of 'admin_test', is a Boolean entry and the JSON data block could look like:

```
{
  "CustomFields":
    {
```

```
      "admin_test":
      {
          "exitCode":null,
          "status":0,
          "updateTime":"'$current_time'",
          "value":true
      }
    }
}
```

When Custom Fields are updated, a time stamp of when the update occurred is shown. When altering the value from the Command Line API, the time should be supplied, along with the value and a couple of other key/value pairs. There should be no reason to provide an other value than those shown for 'exitCode' and 'status', when adjusting using the API.

Date is important and should be of the format "2018-09-10T15:48:08Z":

macOS example:

```
current_time=$(date -u +"%FT%TZ")
```

Windows PowerShell:

```
$current_time = $(Get-Date -UFormat "%Y-%m-%d %R:%S")
```

The code would look something like the following, where $query would be set as the content of the JSON:

macOS shell

```
curl -s -H "Authorization: $auth" \
    https://$server_dns:20445/inv/api/v1/query_result/ \
    -X PATCH \
    -data $query \
    -H "Content-Type: application/json"
```

Windows PowerShell

```
$header = @{Authorization="$auth"}

Invoke-RestMethod -Method PATCH \
    -Headers $header \
    -ContentType application/json \
    -Uri https://$server_dns:20445/inv/api/v1/query_result/ \
    -Body $query
```

As with any Inventory Query, the criteria can be based upon any inventory item. This makes the Command Line API very powerful.

FileWave Anywhere API (v2)

What

Hopefully it is now clear that FileWave Anywhere (formerly called WebAdmin) functions by making API calls to the FileWave server; meaning these same API calls can be used in scripts or by other 3rd party tools.

When/Why

Perhaps to make automated API calls from within Scripts ran on devices or 3rd party tools to GET or POST/PATCH/DELETE entries on the FileWave server.

How

In the main, the queries will look almost the same as the Command Line RESTful commands. Not only are the commands the same, but it is also necessary to use a base64 FileWave Administrator Token. Please view the Working with APIs KB page for more details if required.

Read a current Inventory Query

Imagine it was desirable to read an already existing Inventory Query with an ID of 191. The below shows the Command Line API call, followed by the FileWave Anywhere API call.

Command Line:

```
curl -s  -H "Authorization: $auth" https://$server_dns:20445/inv/api/v1/query_result/191
```

FileWave Anywhere:

```
curl -s  -H "Authorization: $auth" https://$server_dns:/api/inv/api/v1/query_result/191
```

As indicated in an earlier API KB document, FileWave Anywhere uses port 443 (default if not specified) and the URL has an additional '/api' at the beginning of the URL path.

In an example setup, both of these would report exactly the same output, which might look something like:


```
{
  "offset":0,
  "fields":
    [
      "Client_device_name"
  ],
  "values":
    [
      ["C02Z90WDLVDQ"],
      ["FW1063"],
      ["mac0001"],
      ["macadmin1"],
      ["mini"],
      ["ML1015VMNEWNAME"],
      ["ml1063"],
      ["Orac"],
      ["pro"]
  ],
  "filter_results":9,
  "total_results":9,
  "version":3
}
```


Send an MDM Command

Since FileWave Anywhere is a tool with a broader range of control, the API options extend beyond just basic queries. As just one example, the below shows a method to send an MDM command to a device, in this instance a command to restart an Android device, based upon the Device ID:

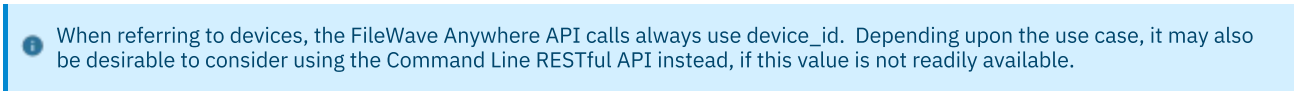
```
curl -X POST "https://${server}/api/android/restart_devices" \
  -H  "Content-Type: application/json" \
  -H "authorization: ${auth}" \
  -d '{"device_ids": ["3aa9cea3eb7ea992"]}'
```

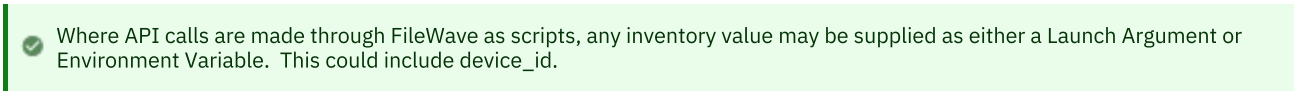
Although FileWave Anywhere has much wider scope than the Command Line, there are some considerations with this API.

- Unlike the Command Line, it is not possible at this time, to run an Inventory Query with the FileWave Anywhere API, without actually creating an Inventory Query on the FileWave server.
- Access to Custom Fields values for devices at this time, is only accessible via an Internal API path and this method reports all associated Custom Field values for a device.
- Setting Custom Field values for a device using the FileWave Anywhere API does not allow the inclusion of a time stamp

As such, the Command Line API should be favoured for these above options.

Please see the FileWave Anywhere Documentation for a more in-depth look at the possible API options available.





FileWave Anywhere API Documentation

What

FileWave Anywhere has a dedicated documentation page, which is built by Swagger and is OpenAPI.

The documentation should only list public (non-internal) URL paths. Not only does it list possible commands, but there is an option to 'Try it Out', allowing execution of the command from the documentation.

> Executing a command from the Swagger API documentation is a live action on the FileWave Server, it will act as requested and can be destructive. Be 100% sure before of any command before executing.

When/Why

Unsure if an API call exists for the intended action? Need to confirm the format or name of a key in the JSON? Are some keys in the JSON required? What is the ID of the item to be targeted?

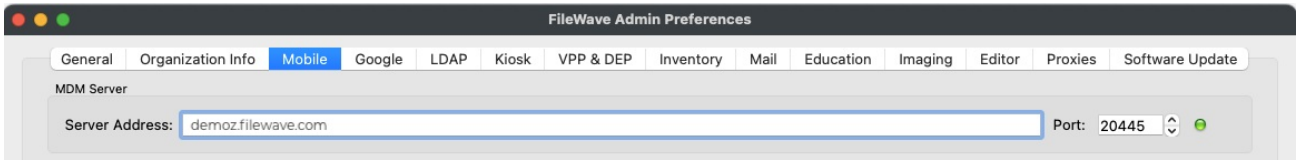
These are some of the questions the Swagger documentation can answer.

> Executing commands from the Swagger documentation requires an active login to the FileWave Anywhere web admin. If this login times out, a fresh login action will be required.

How

> On earlier versions of FileWave, the Documentation page may be accessed without logging in first, but it may then only be viewed as 'read only'.
> This is no longer the case and it should be expected to log into FileWave Anywhere to access the API documentation.

First log into FileWave Anywhere through a web browser. The address should be the same as shown in the FileWave Central Mobile Preferences:

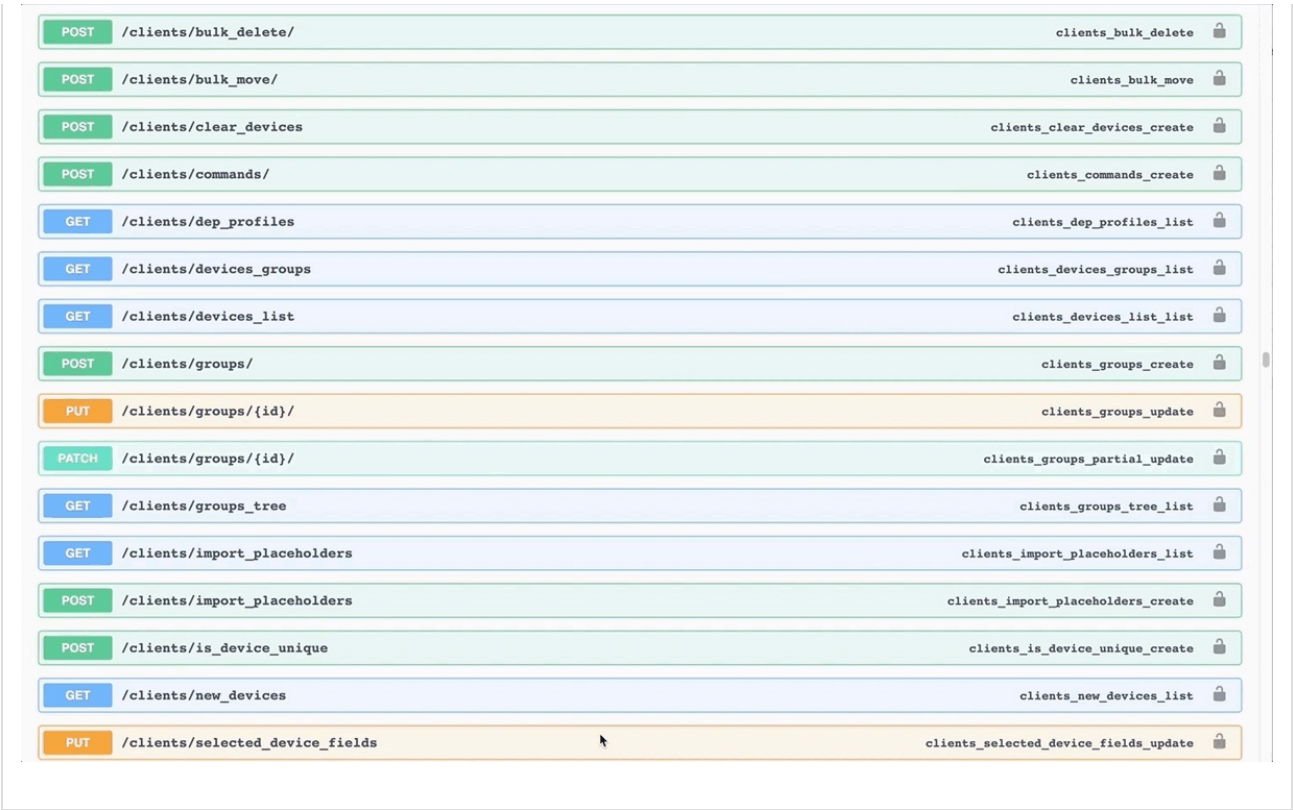


The URL for the Swagger documents page is the server address appended with /api/doc/. From the example above, it would look as follows:

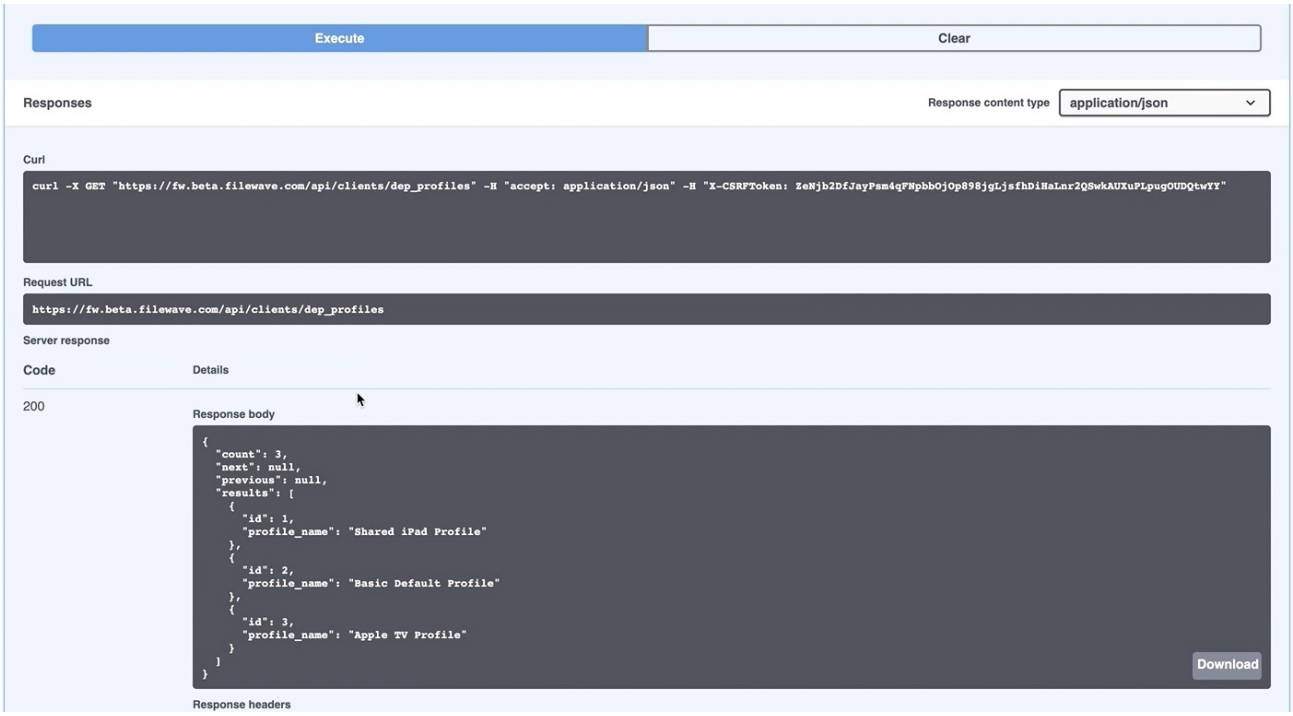
https://demoz.filewave.com/api/doc/

▼ Example GET

The below shows an example of using a 'GET' to return the DEP Profiles:



The response section from an executed action shows the curl command, the requested URL and the returned response (code and body)

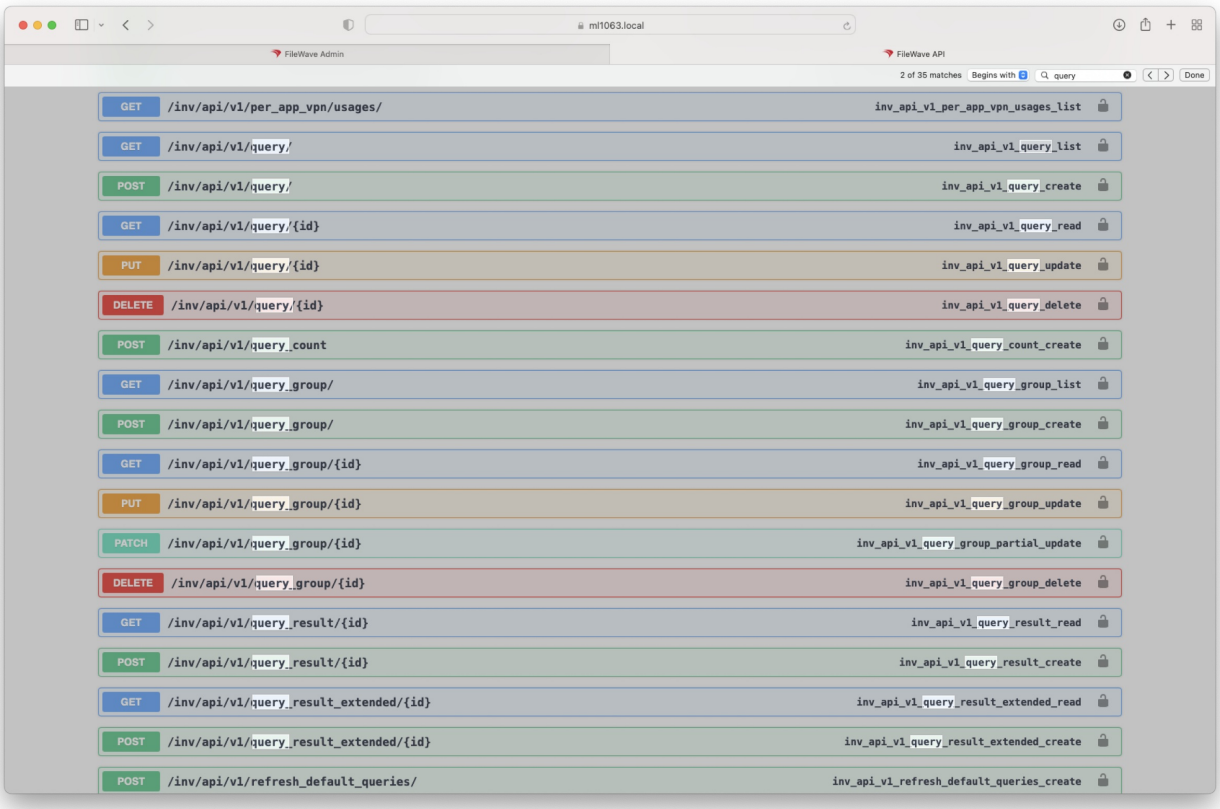


Authentication through the Swagger documentation uses a X-CSRFToken. This token is for use in Swagger alone. The base64 token should be used elsewhere, as shown in all other examples. If copying this command to paste into a script or alternative tool, ensure to remove this part of the line and insert the correct authentication token from FileWave Central.

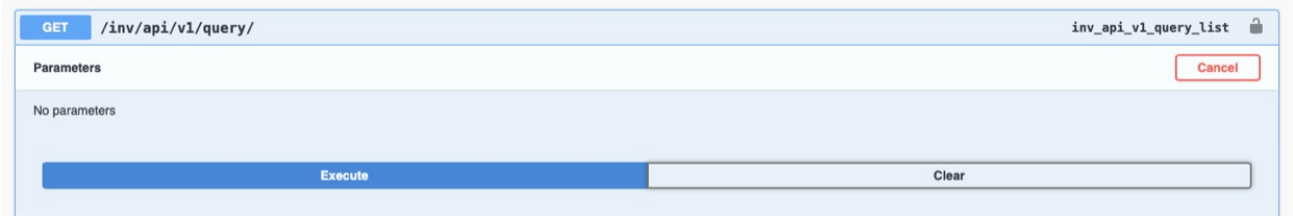
Discovering Content

Swagger can not only be used for testing API calls and discovering the URL paths, but also determining the necessary content of the key/value pairs.

To simply locate all paths relating to queries, for example, the browse 'find' feature could easily assist:

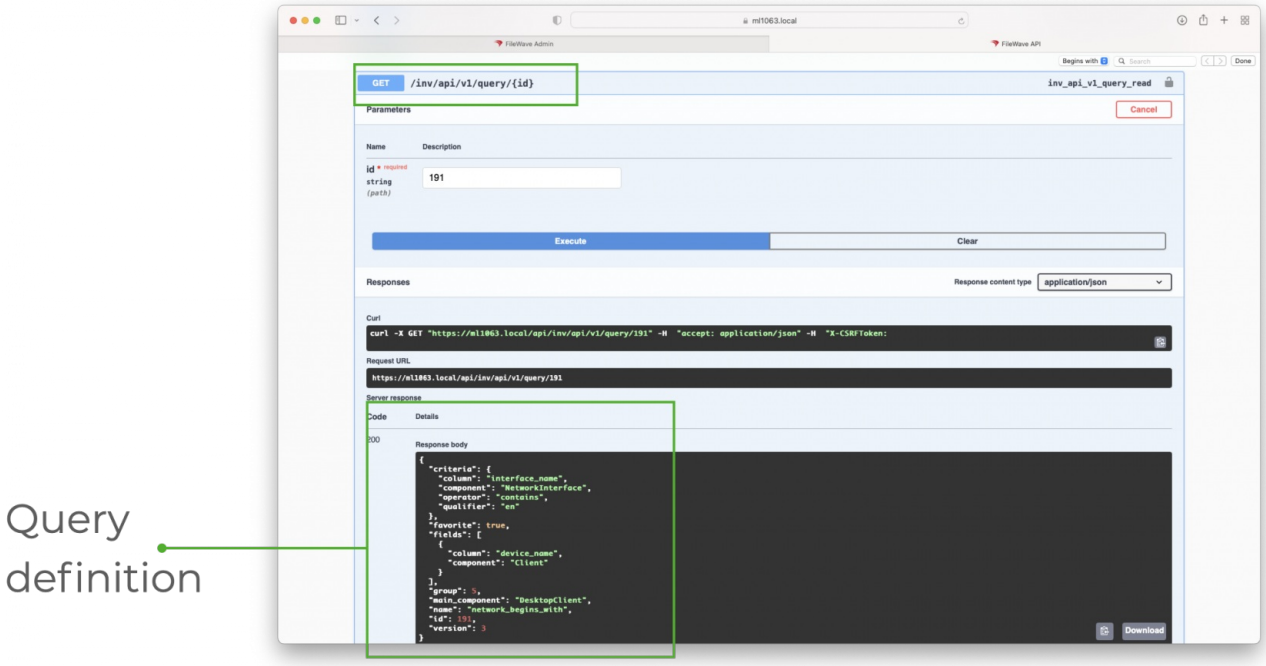


Executing a list of all queries:



The response could then be searched to find the ID of a particular query:

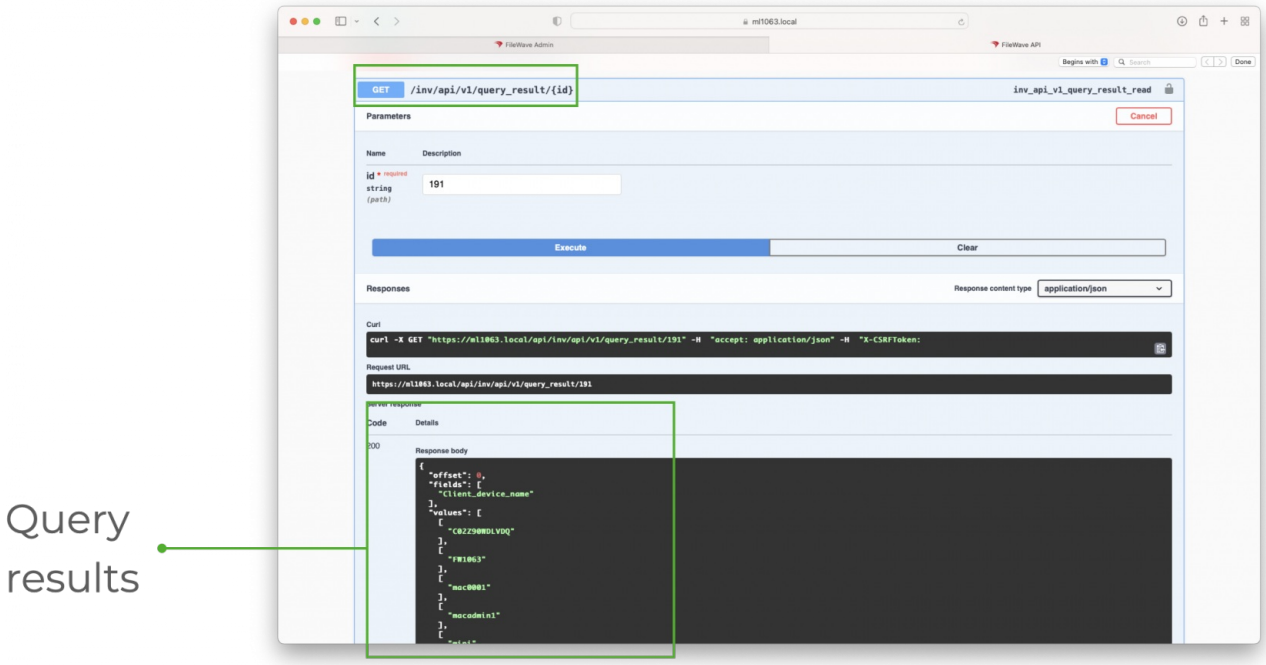
Using Query ID 191 as an example, that ID may then be used in a following GET, which will respond with the definition of that query:



The Query Definition shows the various key/value pairs and can be used to identify values which could then be submitted in other API calls.

> To discover the name used as a value for a particular item, consider building a query in FileWave Central, then using the above steps to locate the name to be used within the API call. For example: interface_name, device_name, DesktopClient.

Taking that another step forward, the same ID could be used to show the response of query results:



The same documentation can therefore be used to assist with the Command Line RESTful API, just remember the path needs to be altered if copying them.

Hopefully this has shown the Swagger documentation to be a great way to discover JSON formatting, key/value names, URL paths and list the commands available through the API.

Related Content

- What is an API?
- Command Line API (v1)

Using the Command Line API to limit, sort, and offset values returned

Problem

When utilizing FileWave's Command Line API to extract inventory information, you may find the need to limit the values returned, sort the data, or return results that are offset by a certain number of values.

Because it can be a topic that can trip someone up it is important to note that API calls are normally done on TCP 443. You may see references to 20445 in some documentation, but that is the Command Line API (v1) and it is on TCP 20445 however you can make a small change in the URL for the API endpoint and use TCP 443 for all calls by prepending /api/ on to the /inv/api/ URLs to make it /api/inv/api/ in the URLs.

Environment

FileWave Command Line API (v1)

Resolution

> The following examples include the pipe to Python to make the output multiple line. Remove this pipe if Python is not installed. This also assumes 'python3' is pathed, such that the full path is not required.

Limit Values

Syntax required to limit the number of values returned in a query to a value of 25:

```
curl -s -H "Authorization: not_the_real_key_here"
https://<your_server_fqdn>:20445/inv/api/v1/query_result/<queryID>?limit=25 | python3 -mjson.tool
```

Of course, you can modify the number of values returned to any number that you like.

Sort Values

You can also sort the results of data returned based on the <column_name> value. For example, to sort the results returned by "device name", ascending:

```
curl -s -H "Authorization: not_the_real_key_here"
https://<your_server_fqdn>:20445/inv/api/v1/query_result/<queryID>?sort=device_name | python3 -mjson.tool
```

To do the same sort, but in descending order (note the "-" in front of the column name):

```
curl -s -H "Authorization: not_the_real_key_here"
https://<your_server_fqdn>:20445/inv/api/v1/query_result/<queryID>?sort=-device_name | python3 -mjson.tool
```

Offset Values

In order to return only a subset of results, offset by some initial number of values returned, you can apply the below syntax:

```
curl -s -H "Authorization: not_the_real_key_here"
https://<your_server_fqdn>:20445/inv/api/v1/query_result/<queryID>?offset=300 | python3 -mjson.tool
```

The above example will return all values, starting from the 300th value.

Related articles

- How to write to a custom field using the FileWave API
- Command Line API (v1)
- Anywhere API (v2)

API Sample Code

Bulk Update the iOS Enrollment User (auth_username) and Client Name using API

What

The api_UpdateiPadNameandAuthUser script is a tool that allows FileWave admins to update the names and assigned users of multiple iOS devices in bulk, using a CSV file containing the serial numbers, desired device names, and LDAP usernames of the devices. This can be particularly useful for education organizations but may also be useful for others who do mass re-distributions of devices like iPads.

When/Why

If you need to quickly and easily reassign a large number of iPads to different users, the api_UpdateiPadNameandAuthUser script can save you time and effort. Instead of manually updating each device individually, you can simply prepare a CSV file with the necessary information and use the script to apply the changes to all of the devices at once. This can be especially useful when you need to set up a large number of new devices for use by different users or when you need to make changes to the assignments of existing devices.

How

To use the api_UpdateiPadNameandAuthUser script, you will need to follow the instructions in the README file that accompanies the script:

1. Download the zip file that contains the script and the README.

   Download: api_UpdateiPadNameANDAuthUser_v2.zip

2. Edit the CSV file to contain a list of serial numbers, device names, and LDAP usernames for the devices you want to update.
3. Open Terminal (on a Mac or Linux) and navigate to the directory that contains the script and CSV file.
4. Make the script executable by running the command chmod +x ./api_UpdateiPadNameandAuthUser.sh.
5. Run the script using the command ./api_UpdateiPadNameandAuthUser.sh SERVER TOKEN CSVFILE, replacing SERVER with the name of your FileWave server, TOKEN with your FileWave API token, and CSVFILE with the path to the CSV file you prepared in step 2.

The script's output will show you the progress of the updates and will indicate when the process is complete.

> If you are running into an issue where the script says that it completed and is Updating Model but no changes are made - please add a blank line at the end of your .csv, save it and try the command again.

API Sample Code

Bulk Update the Enrollment User (auth_username) using API

What

This problem and solution came from a customer who had many devices in FileWave, yet did not have the 'Enrollment User' (internally known as auth_username) populated. In order for automatic associations of iPads with Apple Classroom, devices must have an enrolment user set. While it's possible to set these one by one, that does not scale well. Even hosted customers could benefit from this example.

When/Why

The below solution leverages the FileWave Anywhere API (v2) and is a great example that any customer could build. You can run this script from your mac, Windows, or Linux computer, and it will talk to the API and make the changes in bulk.

> This example happens to be a Python script. As such, if ran on the FileWave Server, Python will already be installed. However, for hosted customers, the script will not be ran on the server and it should be necessary to have Python installed on the device running this script

https://www.python.org/downloads/

https://docs.python.org/3/using/windows.html

How

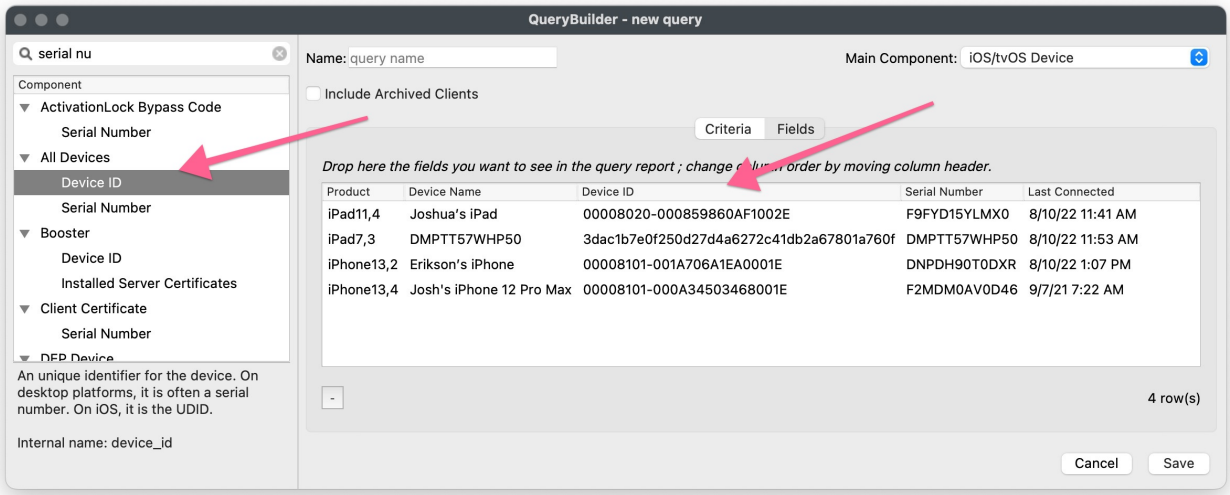
- The first step is to download the script:

- Once downloaded, the zip contains two files: bulk_change_device_authname.py and auth_username.csv
- If the script is not being ran directly on the server, it may be necessary to alter the first line of code, to specify the location of Python on the device running the script. For example, on macOS, that may be: #/usr/local/bin/python3
- The CSV file should be edited to include the desired list of Devices with Users. The supplied template looks like:

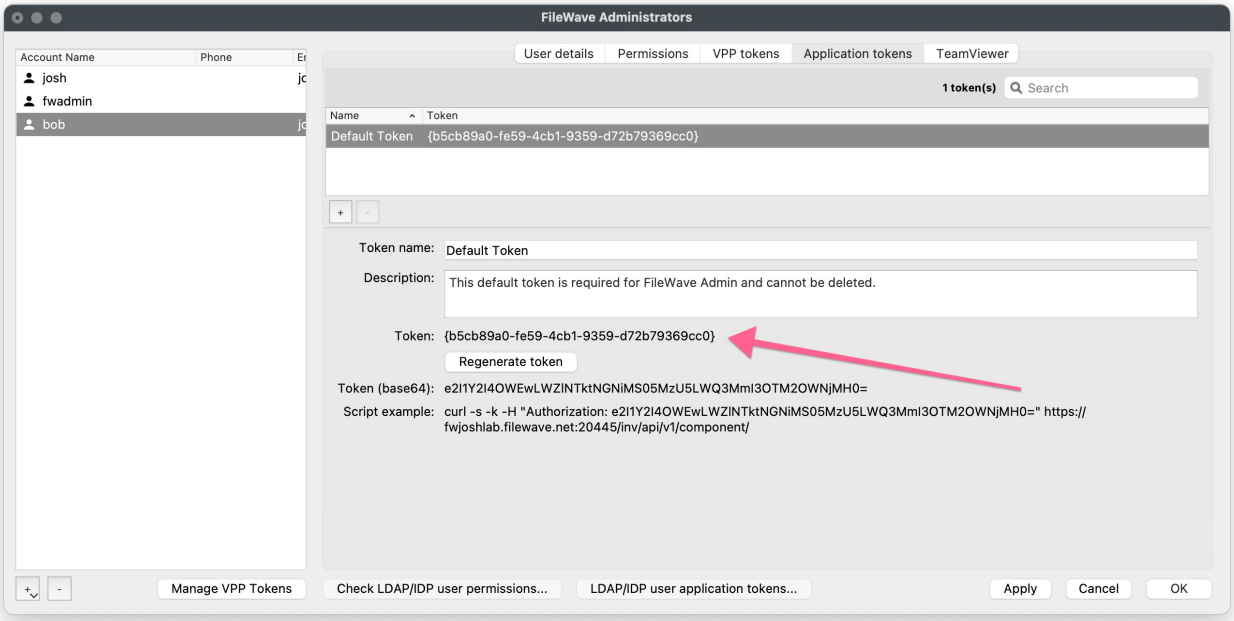
```
Device ID,Enrollment Username
67d6f4bfcf27fa62bb9815365c67ebf7fed8f9c3,test
```

An Inventory query could be used to obtain the list of Device IDs. Note, the script is only expecting two columns in the order of Device ID and Enrollment Username.

> It may assists to initially export additional columns to assist with device identification, but after adding the usernames, be sure to remove any of these additional columns in the CSV file.



- Obtain the desired users base64 Application Token from the FileWave Administrators Assistant view



The following command may now be executed on macOS or Linux (a similar command could be executed on Windows) to action the process.

```
./bulk_change_device_authname.py --token {b5cb89a0-fe59-4cb1-9359-d72b79369c0} --host ExampleCo.filewave.net --mapping ./auth_username.csv
```

The script will give you feedback about the success or failure of any records.

Python Imports

The beginning of the script has a list of imports:

```
import argparse
import base64
import csv
import os
import re
import requests
import sys
```

If any are missing, e.g 'requests', it should be necessary to instal them. The script should report such an error if any are missing when ran. Some are available on a default Python setup.

It is possible to instal any missing. For example, on macOS or Linux, the command may appear as (depending upon the location of

Python)

```
/usr/local/bin/python3 -m pip install requests
```

Hopefully the script at this point has been successful and can be see as a great example of bulk actions.

Related articles

- How to write to a custom field using the FileWave API
- Command Line API (v1)
- Anywhere API (v2)

How to write to a custom field using the FileWave API

It is often desirable to alter FileWave Custom Field values, especially when it comes to driving automated workflows.

> API calls have the distinct advantage of changing Custom Field values directly on the server, essentially making the change immediate.

> Where Custom Field values are driving Smart Group device inclusion, despite the API call making an immediate change, there will still be a period of time before the next Smart Group evaluation.

How

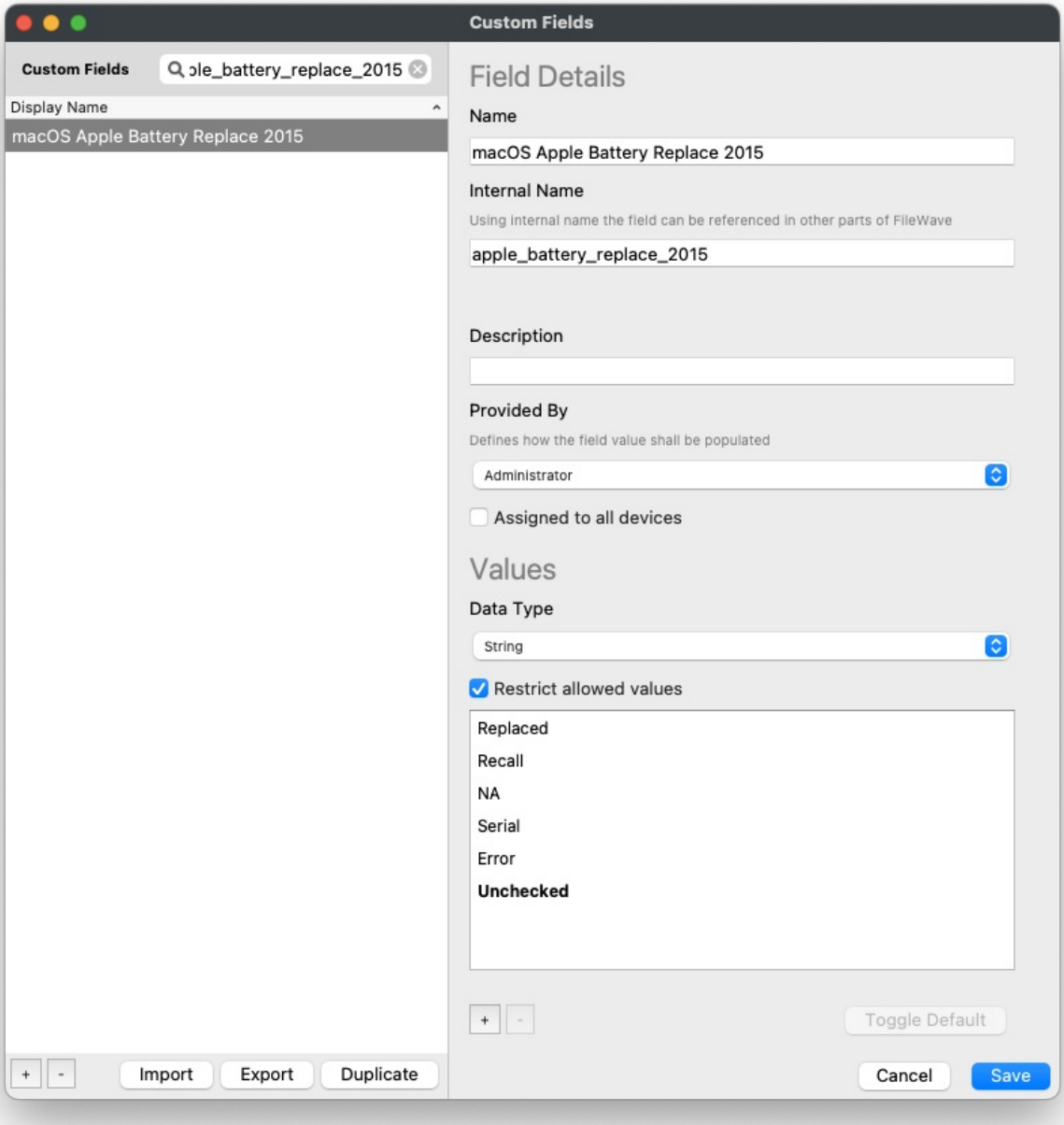
Writing back to a Custom Field relies upon the following details:

- Server URL
- A way to recognise which device's Custom Field should be updated
- A way to recognise the Custom Field
- Custom Field type
- An authentication token

> Some Custom Fields may have restricted values. Only one of these values should be posted in the API call to ensure expected behaviour.

Restricted Values

Value restriction may be observed from within the Custom Field definition.



It is also possible to use Swagger (or therefore another API call) to return the 'choices' list of any Custom Field, by observing the Custom Field definition:

URL path for query:

```
/api/inv/api/v1/custom_field/
```

Query response:

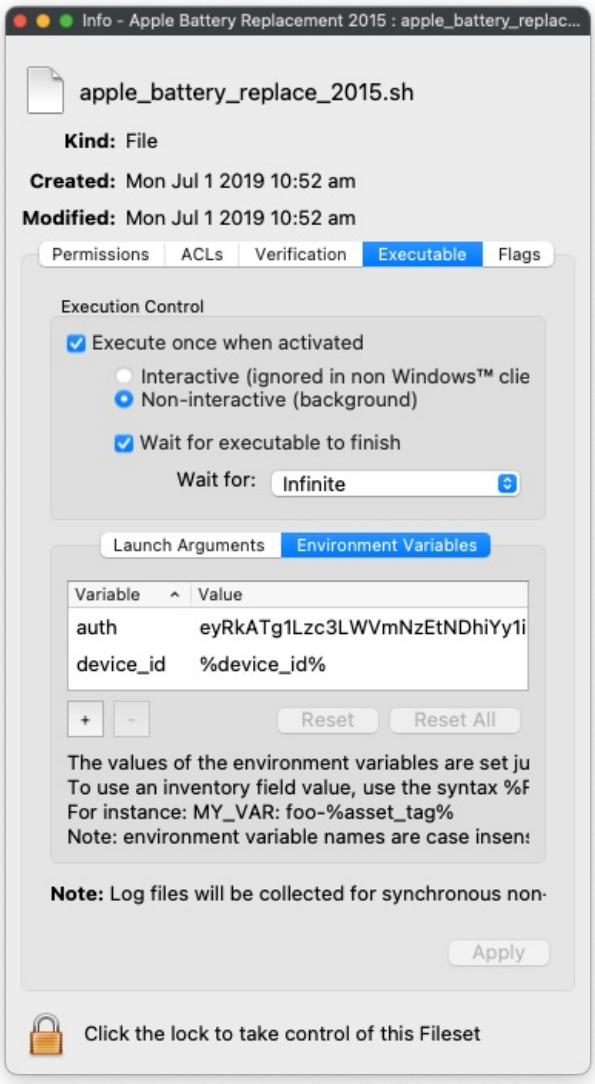

```
{
  "to_be_deleted": false,
  "field_name": "apple_battery_replace_2015",
  "display_name": "macOS Apple Battery Replace 2015",
  "data_type": "string",
  "provider": 0,
  "metadata": {},
  "description": "",
  "default_value": "Unchecked",
  "choices": [
    "Replaced",
    "Recall",
    "NA",
    "Serial",
    "Error",
```

```
      "Unchecked"
    ],
    "is_global": false,
    "used_in_inventory_queries": false,
    "used_in_smart_groups": false,
    "used_in_filesets": false,
    "used_in_license_definitions": false,
    "used_in_dep_profiles": false,
    "used_in_dep_rules": false,
    "used_in_workflows": false
  },
```

Which API

Custom Fields, as mentioned in the other API KB documents, are best targeted with the Command Line RESTful API.

This example is demonstrating the possibility of targeting devices using both Serial Number and Device ID. The Server's FQDN, API authentication token and Device ID can be supplied as Launch Arguments, however, for security reasons it is better to supply the token as an Environment variable. When targeting Internal Names of Custom Fields (as either Launch Arguments or Environment Variables), surround the Internal Name with % symbols.



The start of a macOS script may look like:

```
#!/bin/zsh

# Environment Variables
# $auth - base64 authentication token
# $device_id  - Device ID

server_fqdn=$(defaults read /usr/local/etc/fwcld.plist server) # FW Server FQDN
```

```
serial_number=$(ioreg -l -d 2 | awk -F "\"" '/IOPlatformSerialNumber/ {print $(NF-1)}') # device serial number
```

It is of course possible that all of these values could be supplied to the script as Executable variables.

The start of a PowerShell script may look like:

```
# Environment Variables
# $Env:auth - base64 authentication token
# $Env:device_id - Device ID
# $Env:serial_number - device serial number
# $Env:server_fqdn - FW Server FQDN
```

Reading a Custom Field

It may be necessary to read the Custom Field during the script execution. A JSON will be required for the data portion of the command:

```
{"criteria":
  {
    "column":"serial_number",
    "component":"Client",
    "operator":"is",
    "qualifier":'\"$serial_number\"'
  },
  "fields":
  [
    {
      "column":"apple_battery_replace_2015",
      "component":"CustomFields"
    }
  ],
  "main_component":"Client"
}
```

To continue the scripts, this could be assigned in the script as a variable

macOS script:

```
query='{"criteria":
{"column":"serial_number","component":"Client","operator":"is","qualifier":'\"$serial_number\"'},"fields":
[{"column":"apple_battery_replace_2015","component":"CustomFields"}],"main_component":"Client"}'
```

Windows PowerShell:

```
$query = '{"criteria":{"column":"serial_number","component":"Client","operator":"is","qualifier":"' +
$serial_number + '"},"fields":
[{"column":"apple_battery_replace_2015","component":"CustomFields"}],"main_component":"Client"}'
```

With the server details and JSON configured, it is now possible to read the value with a command:

macOS script:

```
curl -s -H "Authorization: $auth" https://$server_fqdn:20445/inv/api/v1/query_result/ --data $query -H "Content-
Type: application/json"
```

The response may look something like:

```
{"offset":0,"fields":["CustomFields_apple_battery_replace_2015"],"values":
[["Replaced"]],"filter_results":1,"total_results":1,"version":0}
```

Windows PowerShell script:

```
$header = @{Authorization=“$auth"}
$api = "https://" + $server_dns + ":20445/inv/api/v1/query_result/"

Invoke-RestMethod -Method GET -Headers $header -Uri $api
```

Due to this being a Custom Field designed for an Apple replacement programme, hopefully the response will look something like:

```
{"offset":0,"fields":["CustomFields_apple_battery_replace_2015"],"values":
[["NA"]],"filter_results":1,"total_results":1,"version":0}
```

Handling JSON response

As noted earlier, since the response is also a JSON block, the desired information is somewhat buried within the response. Windows PowerShell has tools to directly work with JSON and as such the desired item is more easily attainable.

ConvertFrom-Json

ConvertTo-Json

macOS on the other hand, it would be beneficial to either instal Python and use Pythons tools to extract the response or get crazy with a tool like 'AWK'.

```
curl -s -H "Authorization: $auth" \
  https://$server_fqdn:20445/inv/api/v1/query_result/ \
  --data $query -H "Content-Type: application/json" \
  | awk -F  '[\\\[|\\\]]' '{gsub(/\"/,"",$0);print  substr( $(NF-2), 1, length($(NF-2)))}'
```

Response with AWK:

```
Replaced
```

Writing a Custom Field

Once the script has continued and actioned anything else desired, it then may be desirable to set the Custom Field to a new value, which may vary depending upon the outcome of the scripting.

In this example, we will consider the script will be writing back NA to the Custom Field:

```
current_time=$(date -u +"%FT%TZ")
data='{"CustomFields":{"apple_battery_replace_2015":
{"exitCode":null,"status":0,"updateTime":"'$current_time'","value":"NA"}}}'

curl -X PATCH https://$server_fqdn:20445/inv/api/v1/client/$device_id -d "$data" -H 'content-type:
application/json'  -s -H "authorization: $auth_key"
```

Note:

- This is now using the PATCH option, since an already existing value is being altered by the script
- The date is being supplied as a variable to ensure the current time is pushed back with the API JSON data
- This command is referencing 'device_id' in the URL path

Related articles

- How to write to a custom field using the FileWave API
- Command Line API (v1)
- Anywhere API (v2)

API Sample Code

Managing Client States via the FileWave API

What

This article explains how to manage client device states in FileWave, specifically focusing on how to archive and reinstate clients using the FileWave API. The client state can be defined as Tracked, Archived, Missing, or Untracked, each represented by a numerical value.

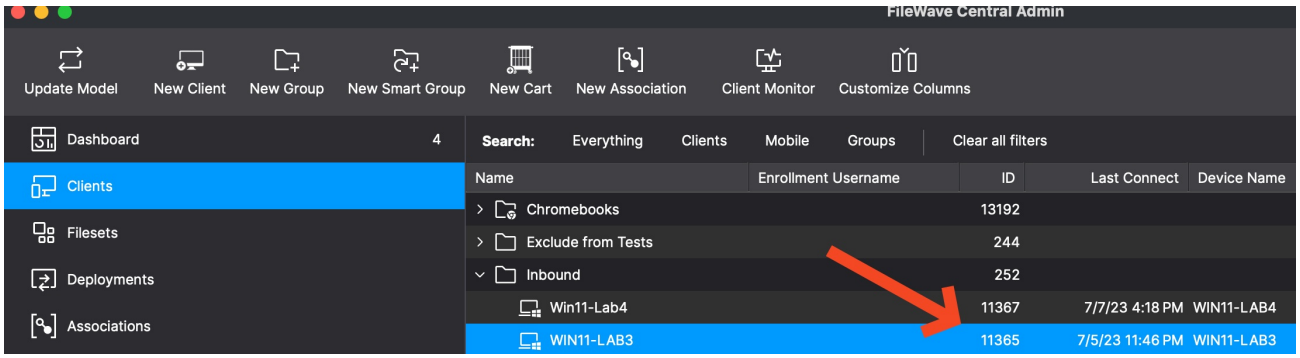
When/Why

Changing the state of a client may be necessary during device management tasks such as inventory control, security audits, or when a device is no longer in active use but needs to be retained in the system for record-keeping. Archiving clients helps in decluttering the active management list without permanently deleting the device record, allowing for easy reinstatement if needed.

> ⚠ Note that Apple MDM enrolled devices will break their MDM enrollment upon being Archived so they can't as easily be reinstated simply by changing their state.

How

To change the state of a device, you can use the FileWave API to send a PATCH request that updates the device state. Below is a script using zsh followed by a PowerShell script to change the state of a device and ensure the model is updated to reflect this change. The DeviceID can be seen in FileWave Central as shown in the below image or in FileWave Anywhere you will see it in the URL when looking at a device.



Shell script:

```
#!/bin/zsh

# Variables
ServerURL="https://fwjoshlab.filewave.net"   # Replace with your server address.
Token="your_token_here"  # Replace 'your_token_here' with your actual token.
DeviceID="11365"          # Specify the device ID.
NewState="0"              # Set the desired state (0: Tracked, 1: Archived, 2: Missing, 3: Untracked).

# Update device state
curl -X PATCH "$ServerURL/filewave/api/devices/v1/devices/$DeviceID" \
     -H "Authorization: Bearer $Token" \
     -H 'Content-Type: application/json' \
     -d '{"state":'$NewState'}'

# Update the model to reflect changes
curl -X POST "$ServerURL/filewave/api/fwserver/update_model" \
     -H "Authorization: Bearer $Token" \
     -H 'Content-Type: application/json'
```

PowerShell:

```
# PowerShell Script to Manage FileWave Client States

# Variables
```

```
$ServerURL = "https://fwjoshlab.filewave.net" # Replace with your server address.
$Token = "your_token_here"  # Replace 'your_token_here' with your actual token.
$DeviceID = "11365"         # Specify the device ID.
$NewState = "0"             # Set the desired state (0: Tracked, 1: Archived, 2: Missing, 3: Untracked).

# Headers for authorization and content type
$headers = @{
    "Authorization" = "Bearer $Token"
    "Content-Type" = "application/json"
}

# Body data for changing the state
$body = @{
    "state" = $NewState
} | ConvertTo-Json

# Update device state
Invoke-RestMethod -Uri "$ServerURL/filewave/api/devices/v1/devices/$DeviceID" -Method Patch -Headers $headers -
Body $body

# Update the model to reflect changes
Invoke-RestMethod -Uri "$ServerURL/filewave/api/fwserver/update_model" -Method Post -Headers $headers

# Output for user confirmation
Write-Host "Device state updated and model refreshed successfully."
```

Tips:

- Ensure that the `Token` variable contains a valid authorization token.
- Replace `DeviceID` and `NewState` with the appropriate values according to your needs.

Related Links

- FileWave API Documentation - Official API documentation.
- CURL Command Line Tool - Learn more about how to use curl.

Digging Deeper

Understanding the model update process is crucial for ensuring that changes made via the API are reflected in the FileWave management interface. The `update_model` API call triggers the FileWave server to reprocess its internal data models, ensuring that any state changes are accurately shown in the admin console. This is especially important after bulk changes to device states to maintain consistency across the system.

Returning Device Information as a JSON

What

The Client Info > Device Details of a particular client, contains a wealth of information that may be useful to repurpose in other systems (Help Desks, centralised inventory systems, etc). Using the FileWave API, this information could be pulled by alternate systems or to a file locally on the client

> Since the command refers to Device IDs, it may be necessary to make 2 calls from external systems. The first to obtain Device IDs and the second to target particular devices based upon their Device ID.

> When ran through Filesets, Device ID may be sent with the Fileset as either a Launch Argument or Environment Variable

HOW

This information could be returned using either the FileWave Anywhere API or the Command Line RESTful API

> Remove the pipe to Python if not installed. This just displays the output as multiple lines instead of one long line.

FileWave Anywhere API from macOS or Linux:

```
curl -s -H "Authorization: $auth" \
  https://$server_dns/api/inv/api/v1/client/details/${device_id}/DesktopClient \
  -H "Content-Type: application/json" \
  | python3 -mjson.tool
```

Command Line RESTful API from macOS or Linux:

```
curl -s -H "Authorization: $auth" \
    https://$server_dns:20445/inv/api/v1/client/details/${device_id}/DesktopClient \
    -H "Content-Type: application/json" \
    | python3 -mjson.tool
```

> Note, the commands look almost identical, but just the additional /api at the beginning of the path for the FileWave Anywhere API call.

The output should look similar to the below, where an appropriate device_id is supplied:


```
{
    "CustomFields__ldap_username": {
        "status": 0,
        "type": "string",
        "updateTime": "2018-06-21T19:37:23.585851Z",
        "value": "mdm mdm"
    },
    "CustomFields__local_ip_address": {
        "status": 0,
        "type": "string",
        "updateTime": "2018-06-21T19:49:51Z",
        "value": "10.20.30.29"
    },
    "CustomFields__malwarebytes_installed": {
        "status": 0,
        "type": "bool",
        "updateTime": "2018-06-21T19:49:51Z",
        "value": false
    },
    "CustomFields__po_number": {
        "status": 0,
        "type": "string",
```

```
    "updateTime": "2018-06-21T19:49:51Z",
    "value": "54654561"
},
"CustomFields__property_tag": {
            "status": 0,
    "updateTime": "2018-06-21T19:49:51Z",
    "type": "string",
    "value": "Device Owned by FileWave"
},
"CustomFields__purchase_date": {
    "updateTime": null,
    "value": null
},
"CustomFields__school_name": {
    "status": 0,
    "type": "string",
    "updateTime": "2018-06-21T19:49:51Z",
    "value": "Landing Trail Elementary"
},
"CustomFields__site_description": {
    "updateTime": null,
    "value": null
},
"CustomFields__textedit_version": {
    "status": 0,
    "type": "string",
    "updateTime": "2018-06-21T19:49:51Z",
    "value": "1.13"
},
"CustomFields__user_role": {
    "updateTime": null,
    "value": null
},
"archived": null,
"auth_username": "mdm",
"building": null,
"cpu_count": 2,
"cpu_speed": 2759000000,
"cpu_type": "Intel(R) Core(TM) i5-3470S CPU @ 2.90GHz",
"current_ip_address": "10.20.30.29",
"deleted_from_admin": false,
"department": null,
"device_id": "f96b8c66c50b358889ba2fbf2dc53bc21036406a",
"device_manufacturer": "VMware, Inc.",
"device_name": "FUSION-VM1-10.12",
"device_product_name": "VMware7,1",
"enroll_date": "2018-06-17T17:11:08.709785Z",
"enrollment_state": 2,
"filewave_client_locked": false,
"filewave_client_name": "FUSION-VM1-10.13",
"filewave_client_version": "12.8.1",
"filewave_id": 219,
"filewave_model_number": 617,
"free_disk_space": 56772587520,
"is_system_integrity_protection_enabled": true,
"is_tracking_enabled": false,
"last_check_in": "2018-06-21T19:54:31.615710Z",
"last_enterprise_app_validation_date": null,
"last_ldap_username": null,
"last_logged_in_username": "dhadmin",
"last_state_change_date": "2018-06-21T19:50:09.339609Z",
"location": null,
"management_mode": 0,
"monitor_id": null,
"operating_system__build": "17B48",
"operating_system__edition": "Desktop",
"operating_system__name": "macOS 10.13 High Sierra",
"operating_system__type": "OSX",
"operating_system__version": "10.13.1",
"operating_system__version_major": 10,
```

```
    "operating_system__version_minor": 13,
    "operating_system__version_patch": 1,
    "ram_size": 2147483648,
    "rom_bios_version": "VMW71.00V.0.B64.1706210604",
    "security__enrolled_via_dep": null,
    "security__fde_enabled": false,
    "security__firmware_password_change_pending": false,
    "security__firmware_password_exists": false,
    "security__firmware_password_rom_enabled": true,
    "security__hardware_encryption_caps": null,
    "security__passcode_is_compliant": null,
    "security__passcode_is_compliant_with_profiles": null,
    "security__passcode_lock_grace_period": null,
    "security__passcode_lock_grace_period_enforced": null,
    "security__passcode_present": null,
    "security__system_integrity_protection_enabled": true,
    "security__user_approved_enrollment": null,
    "serial_number": "VMx4NvUkh/Co",
    "state": 0,
    "total_disk_space": 85689589760,
    "unenrolled": false
}
```

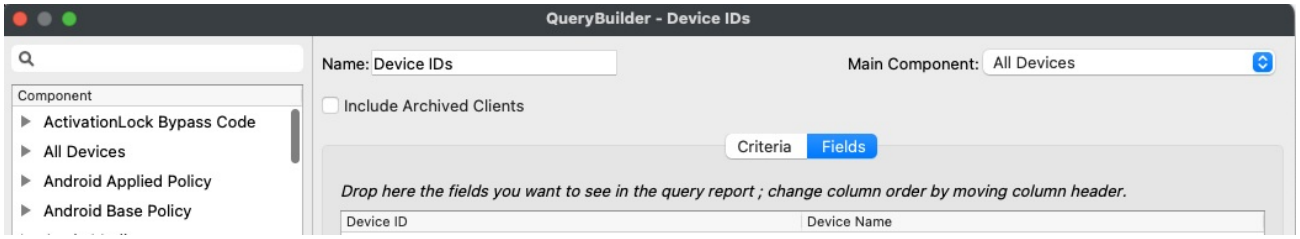

If desired, the information could be stored into a JSON file:

```
curl -s -H "Authorization: $auth" \
  https://$server_dns/api/inv/api/v1/client/details/${device_id}/DesktopClient \
  -H "Content-Type: application/json" \
  | python3 -mjson.tool > /my/path/device_info_${device_id}.json
```

The same $device_id variable has been used to define the name of the JSON file also. Alter /my/path for a path of choice.

Obtaining Device IDs

One way to retrieve a bulk list of Device IDs is via an Inventory Query. First make a query to include desired columns, one of which will need to be Device ID. In the below example Device ID and Device Name have been included as columns for the Fields:



Once saved, use the details outlined in the FileWave Anywhere Documentation, to locate the ID of this Inventory Query. The query result may then be used to pull a list of Device IDs. From the below example, set $query_id to the value of the chosen Inventory Query. Make sure to set $auth to the token and server to your server:

```
#!/bin/zsh
# Shell script for macOS/Linux
$server = "widget.filewave.net"
$token = "ezMyxxxM2UyLTNjN2ItNxxxS04ZjQ5LTkxMxxxxzEzODZmNn0="
$query_id = "65"

curl -s  -H "Authorization: $auth" \
  https://$server/api/inv/api/v1/query_result/$query_id
```

```
#PowerShell for Windows
$server = "widget.filewave.net"
$token = "ezMyxxxM2UyLTNjN2ItNxxxS04ZjQ5LTkxMxxxxzEzODZmNn0="
$header = @{Authorization="$token"}
$query_id = "65"

Invoke-RestMethod -Method GET \
    -Headers $header \
    -ContentType application/json \
```

```
    -Uri https://$server:/api/inv/api/v1/query_result/$query_id
```

Related articles

- How to write to a custom field using the FileWave API
- Command Line API (v1)
- Anywhere API (v2)

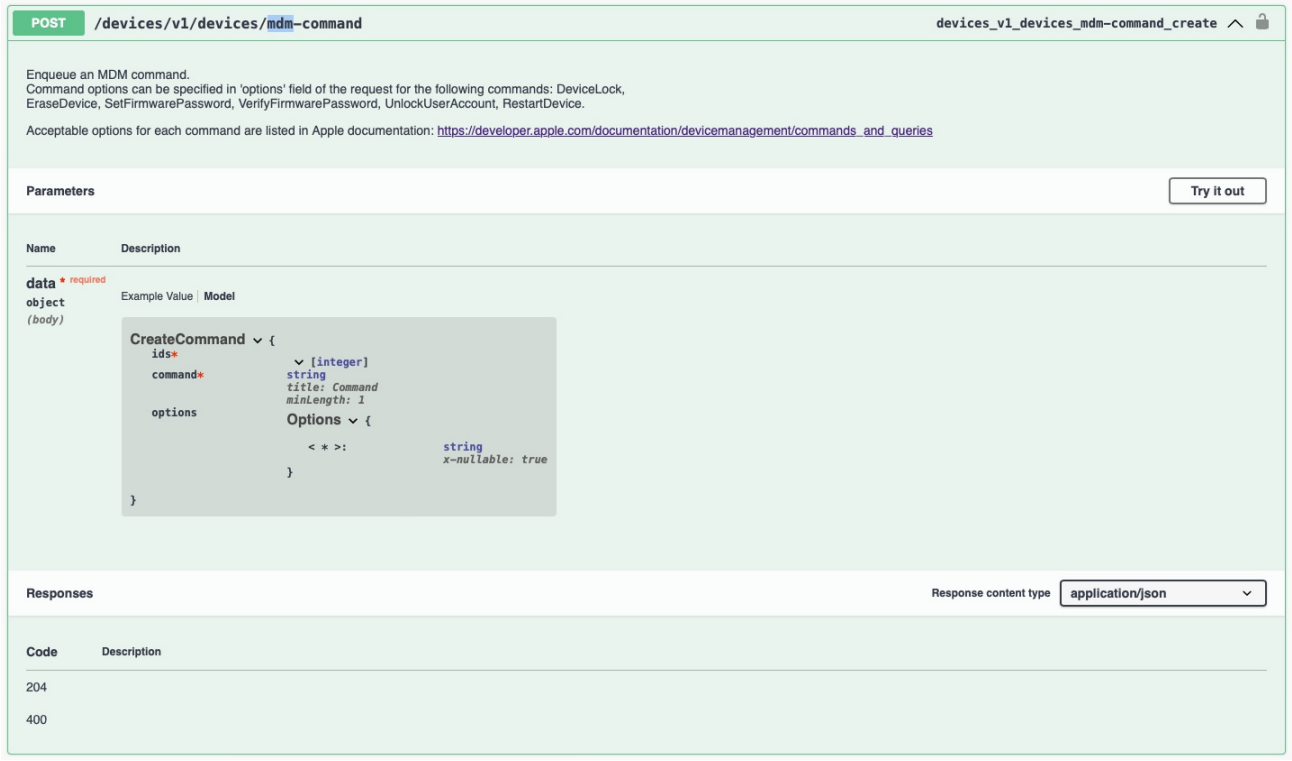
API Sample Code

Sending MDM Commands

What

One of the powerful additional features of FileWave Anywhere, is the ability to send MDM commands to devices. As such, the FileWave Anywhere API also has this incredible ability.

From, the Swagger Documentation, the following can be seen:



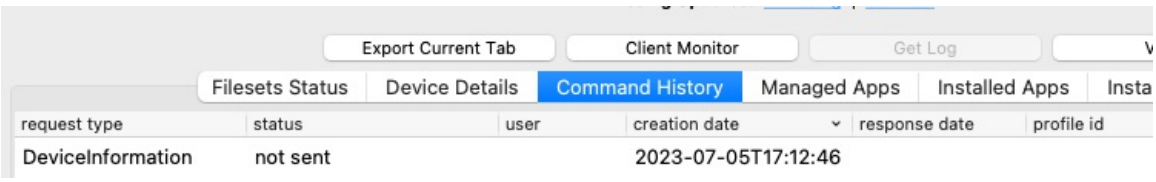
> Note the reference to the device(s) is now by 'ids'. This refers to the Client ID, as oppose to the Device ID, and is always an Integer.

Example data could look like:

```
{
  "ids": [
    737581
  ],
  "command": "DeviceInformation"
}
```

> 'ids' is a list of devices, so multiple devices could be targeted in one RESTful API command.

Commands sent by the API will show in the device's Client Info:



HOW

This is an example of a RESTful API request that is only available through the FileWave Anywhere API.

Running the command from the Swagger Documentation will show the URL path required to send a command. For example, to restart devices:

macOS and Linux:

```
mdm_command='{"ids": [737581, 562620],"command": "RestartDevice"}'
```

```
curl -H "Authorization: $auth" \
  -X POST https://${server_dns}/api/devices/v1/devices/mdm-command \
  -d "$mdm_command" \
  -H "Content-Type: application/json"
```

Windows Powershell:

```
$mdm_command = '{"ids": [737581, 562620],"command": "RestartDevice"}'
```

```
$header = @{Authorization="“$auth"}

Invoke-RestMethod -Method POST \
    -Headers $header \
    -ContentType application/json \
    -Uri https://${server_dns}/api/devices/v1/devices/mdm-command \
    -Body $mdm_command
```

What commands are available

From the Swagger, there is the following text:

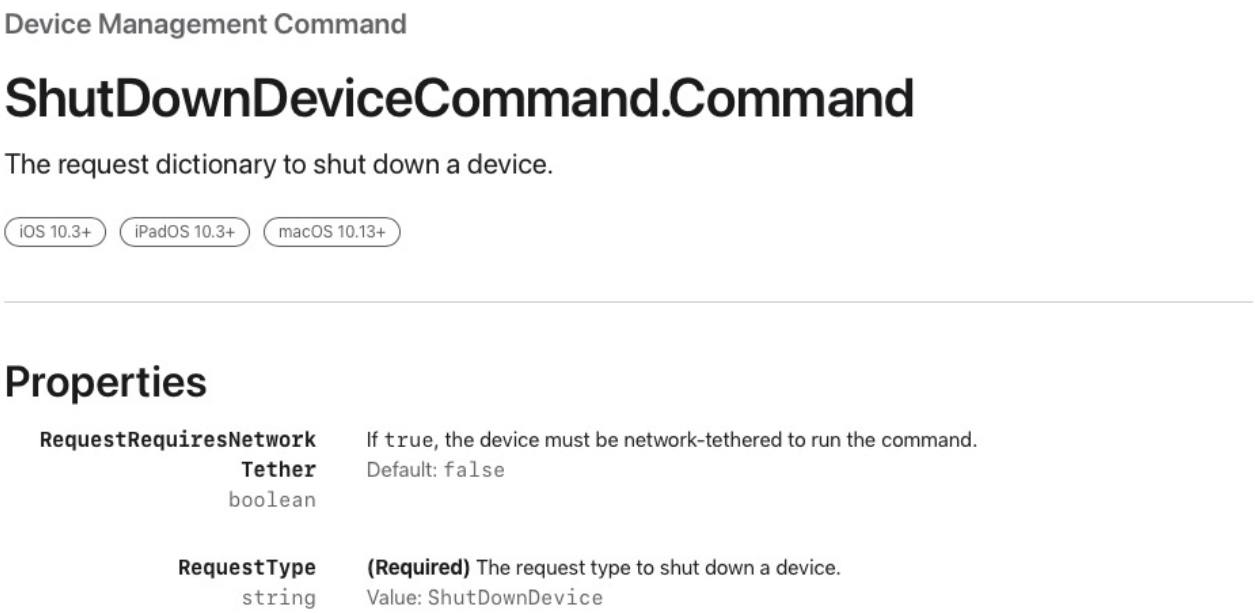
Command options can be specified in 'options' field of the request for the following commands: DeviceLock, EraseDevice, SetFirmwarePassword, VerifyFirmwarePassword, UnlockUserAccount, RestartDevice.

Acceptable options for each command are listed in Apple documentation:
https://developer.apple.com/documentation/devicemanagement/commands_and_queries

Some commands have been listed, but the link to Apple's documentation shows all possible commands:

> If a new command is released by Apple before it appears in FileWave, the API should be able to trigger that command

To use Apple's documentation, navigate through the pages for the chosen command to locate the 'RequestType'. For example, the following shows the command to shut a device down is: ShutDownDevice



Related articles

- How to write to a custom field using the FileWave API
- Command Line API (v1)
- Anywhere API (v2)