**How the program works:**

The program pulls information from tables and translates them into API objects that are then pushed to a MCr endpoint. The most important information in the tables are the column headers. Each column header corresponds to some configuration information in the AOS8 system. If a column header is mistyped or is not currently supported, the program will throw an error and exit. A list of column headers and the corresponding configuration information can be found here.

**Building tables:**

The most basic table that you can configure is a one column table.

| AP Group |
| --- |
| CGR |
| DEP |
| KFD |
| PWK |
| SIG |
| Test |

This will make 6 AP groups with the corresponding names specified in the cells. If you want to add other configuration information to these AP groups, add more columns.

| AP Group | Group VAPs |
| --- | --- |
| CGR | ChoiceAccess,CorpAccess,CorpTest,GuestAccess,MobiAccess |
| DEP | ChoiceAccess,CorpAccess,CorpTest,GuestAccess,MobiAccess |
| KFD | ChoiceAccess,CorpAccess,CorpTest,GuestAccess,MobiAccess,FitnessTest |
| PWK | ChoiceAccess,CorpAccess,CorpTest,GuestAccess,MobiAccess |
| SIG | ChoiceAccess,CorpAccess,CorpTest,GuestAccess,MobiAccess |
| Test | GuestAccess\_LocalCP |

If a column contains multiple values, use a comma separated list.

**Nodes:**

It’s best practice to define a place in the hierarchy where a set of configuration should be applied. Add a Node column in your table to specify this information.

| Node | VLAN ID | VLAN Name |
| --- | --- | --- |
| /md/ATC | 50 | wifi-atc |
|  | 51 | wifi-byod |
|  | 52 | vendor-dmz |
|  | 309 | internal |
|  | 376 | corp-wifi |
|  | 612 | KFD-Fitness |
| /md/ATC/peewaukee | 209 | internal |
|  | 276 | corp-wifi |

It’s tedious to write the same Node information multiple times so just group configuration that will be applied to the same Node and the program will fill in the information in the back end.

If you do not specify a Node column then the set of configurations will be applied to a default node which you can specify when starting the script.

**Devices:**

The Device column is used to apply device specific configuration. Just use the hostname of the device you want to configure.

| Device | Int VLAN ID | Int VLAN IP | Int VLAN IP Mask |
| --- | --- | --- | --- |
| pwkinf302p | 50 | 172.30.64.13 | 255.255.255.248 |
|  | 51 | 172.30.68.13 | 255.255.255.248 |
|  | 52 | 172.30.3.10 | 255.255.255.248 |
|  | 200 | 10.32.0.20 | 255.255.255.128 |

**Nested Profiles:**

If a profile is going to be applied to another profile, name the two profiles the same name.

| Wired Port Profile | Wired Port Bridge Role | Wired Port AAA Prof |
| --- | --- | --- |
| default | logon |  |
| tac-test | logon | tac-test |

There is also an implicit way of associating profiles with each other. Let’s say you have two tables, one is configuration for an SSID profile and the other is configuration for an HE SSID profile:

| Node | WLAN ESSID |
| --- | --- |
| /md/ATC | ChoiceAccess |
|  | CorpAccess |
|  | CorpTest |

| Node | WLAN ESSID |
| --- | --- |
| /md/ATC | ChoiceAccess |
|  | CorpAccess |
|  | CorpTest |

| Node | HE SSID Profile | TX BF | MU-MIMO |
| --- | --- | --- | --- |
| /md/ATC | ChoiceAccess | False | True |
|  | CorpAccess | False | True |
|  | CorpTest | True | True |

If profile names are similar and the nested profile node is a sub-node of the outer profile then the two will be associated with each other. The two tables above could have also been combined into one and the association will still be established. Make sure the associated profiles are on the same row in their respective tables.

There are certain profiles that are assumed to go together like regulatory domain and a/g radio profiles so you don’t explicitly have to associate them.

| RF Profile | 2.4 GHz Minimum | 2.4 GHz Maximum | 5 GHz Minimum | 5 GHz Maximum |
| --- | --- | --- | --- | --- |
| CGR | 3 dBm | 7 dBm | 6 dBm | 10 dBm |
| DEP | 3 dBm | 7 dBm | 6 dBm | 10 dBm |
| KFD | 3 dBm | 7 dBm | 6 dBm | 10 dBm |
| PWK | 3 dBm | 7 dBm | 6 dBm | 10 dBm |
| SIG | 3 dBm | 7 dBm | 6 dBm | 10 dBm |
| Test | 3 dBm | 7 dBm | 6 dBm | 10 dBm |

| RF Profile | 2.4 GHz Channels | 5 GHz Channels | 5 GHz Channel Width |
| --- | --- | --- | --- |
| CGR | 1, 6, 11 | 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161,165 | 20 MHz |
| DEP | 1,6,11 | 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161,165 | 20 MHz |
| KFD | 1, 6, 11 | 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161,165 | 20 MHz |
| PWK | 1, 6, 11 | 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161,165 | 20 MHz |
| SIG | 1, 6, 11 | 36, 40, 44, 48, 52, 56, 60, 64, 149, 153, 157, 161,165 | 20 MHz |
| Test | 1, 6, 11 | 36, 40, 44, 48, 149, 153, 157, 161,165 | 80 MHz |

The two tables above are a mixture of regulatory domain and radio profiles. The corresponding profiles created will be:

CGR\_reg\_domain\_prof, DEP\_reg\_domain\_prof, …

CGR\_ap\_a\_radio\_prof, DEP\_ap\_a\_radio\_prof, …

CGR\_ap\_g\_radio\_prof, DEP\_ap\_a\_radio\_prof, …

Generally, profile names will be suffixed with the corresponding profile type, mimicking the behavior that AOS8 exhibits in the back end. Using the RF Profile column saves some typing but you can also define these profiles individually with the appropriate column header.

**Using the program:**

Before running the script, you will need to install a few modules:

Pip install -r requirements.txt

Run the main.py script on the CLI. There are no required arguments but optionally, you can supply the program with all the information to start building the configurations and push those configs into the MCr network.

Ex. Python main.py -u admin -p password -a 10.10.10.100 -f path/to/config/table.docx

or simply:

Ex. Python main.py

If arguments aren’t supplied, the program will ask for them before running.

**Extending the program:**

Currently, the program has support for the most commonly configured things in a greenfield deployment. If there is an item not currently supported, it can be added to the program. Note that this will not work on all configuration items since some things require hard coded logic in the backend to work correctly.

For a lot of configuration items, however, you can just update the COL\_TO\_ATTR dictionary found in the data\_structures.py file.

1. Find the attribute you would like to configure in the API JSON files.
   1. All attributes can be found under the "definitions" key.
2. Build your attribute name. For example if you had an attribute that looked like this:

"ids\_general\_prof": {

"type": "object",

"required": [

"profile-name"

],

"properties": {

"profile-name": {

"description": "Profile name",

"type": "string",

"maximum": 256,

"minimum": 1

},

"adhoc\_ap\_max\_unseen\_timeout": {

"type": "object",

"required": [

"adhoc-ap-max-unseen-timeout"

],

"properties": {

"adhoc-ap-max-unseen-timeout": {

"description": "Ageout time in seconds since Adhoc (IBSS) AP was last seen. Minimum is 5.",

"type": "integer",

"maximum": 36000,

"minimum": 5

}

}

},

And you want to be able to specify an adhoc AP max unseen timeout, the attribute name will be:

ids\_general\_prof.adhoc\_ap\_max\_unseen\_timeout.adhoc-ap-max-unseen-timeout

For each level that you go into the entry just add a period separating the levels.

1. Associate your attribute name with a column header and add it to the COL\_TO\_ATTR dictionary.
   1. Let's say that you want to call this attribute Max Unseen Timeout in your table. Then your entry in the COL\_TO\_ATTR table will be:
      1. 'Max Unseen Timeout' : ['ids\_general\_prof.adhoc\_ap\_max\_unseen\_timeout.adhoc-ap-mac-unseen-timeout']
   2. The array is so that you can associate multiple attributes to column headers at once.
2. If your attribute has properties that are booleans then you must add an entry to the BOOLEAN\_DICT. Essentially you come up with a name in your table that you want to use for the property and associate that with the property name in the API.

Ex.

"frame\_types\_for\_rssi": {

"type": "object",

"properties": {

"ba": {

"type": "boolean"

},

"pr": {

"type": "boolean"

},

"dlow": {

"type": "boolean"

},

"dhigh": {

"type": "boolean"

},

"dnull": {

"type": "boolean"

},

"mgmt": {

"type": "boolean"

},

"ctrl": {

"type": "boolean"

},

"all": {

"type": "boolean"

}

}

},

For the property above, you might want to define things in your table in a more human readable format. So instead of ba you want to use Beacon. Probe stands for pr and so on. Add these entries into the BOOLEAN\_DICT dictionary:

'Beacon' : 'ba', 'Probe' : 'pr', 'Low Data': 'dlow', 'High Data': 'dhigh', 'Null Data': 'dnull', 'Management': 'mgmt', 'Control': 'ctrl', 'All': 'all'

1. After adding the entries, add the column in a table somewhere and your attribute is now configurable in the network. The program checks input against the API and does all the validation for you.

**Inconsistent API Naming Conventions:**

For most profiles, when they are nested inside other profiles they retain the same name as the independent object in the API. For example, aaa\_prof is the API name for AAA profiles and when this profile is an attribute of a virtual AP profile, its name remains aaa\_prof.

Sometimes, however, the name is changed. When defined inside a wired port profile, the AAA profile name then becomes wired\_aaa\_prof. Whenever this happens, you will need to add an entry to the NESTED\_DICT dictionary:

‘wired\_aaa\_prof’ : [‘aaa\_prof’]

**Column Headers:**

Here’s a list of currently supported column headers. This information is also in the COL\_TO\_ATTR dictionary in the data\_structures.py file.

**Authentication:**

AAA Profile: String

AAA Default Role: String

AAA Download Role: True/False

AAA RFC3576 IP: String

AAA Server Group: String

Rad Server Name: String

Rad Server Hostname: String

Rad Server Key: String

Rad Server AuthPort: Integer

Rad Server AcctPort: Integer

Dot1X Profile: String

Machine Auth: True/False

Server Group: String

SG Server Name: String

Server Derived Role: True/False

RFC 3576 Servers: String/List of Strings (used inside AAA profile)

MAC Auth: True/False

CP Profile: String

CP Login Page: String

CP Server Group: String

CP Default Role: String

CP User Logon: True/False

CP Whitelist: String/List of Strings

AUP: True/False

**Interfaces:**

MGMT Int VLAN: Integer

MGMT Int IP: String

MGMT Int DHCP: String

MGMT Int Netmask: String

PortChannel: Integer

PC Interfaces: String/List of Strings (slot/module/port)

PC Trunk: Trunk/Access

PC Access VLAN: Integer

PC Allowed VLANs: String/List of Strings

PC Native VLAN: Integer

PC Jumbo Frames: True/False

PC Description: String

Loopback Interface: String/IP address

Int VLAN ID: Integer

Int VLAN IP: String/IP address

Int VLAN IP Mask: String/IP address

Int VLAN IP DHCP: String/IP address

Int VLAN IP Helper Address: String/IP address

Gig Interface: String (slot/module/port)

Gig Int Speed: 10,100,1000,Auto

Gig Int Duplex: Full,Half,Auto

Gig Int Access VLAN: Integer

Gig Int Allowed VLANs: String/List of String

Gig Int Allowed All VLANs: True/False

Gig Int Native VLAN: Integer

Gig Int Jumbo Frames: True/False

VLAN ID: Integer

VLAN Option 82: True/False

VLAN Description: String

VLAN Name: String

VLAN Associated IDs: String/List of Strings

**L2L3:**

NTP Server IP: String/IP address

NTP Server IPv6: String/IP address

NTP Server FQDN: String

SNMPv2c Server Host IP: String/IP address

SNMPv2c Server Host Community: String/IP address

SNMP Server Community: String

DNS Server IPs: String/List of Strings (Up to 3 addresses allowed per MC/Node)

IP Route: String/IP Address

**DHCP Pools:**

DHCP Pool Name: String

DHCP Pool DNS IP: String/List of Strings

DHCP Pool Default Router: String/List of String

DHCP Pool Subnet: String/IP address

DHCP Pool Subnet Mask: String/IP address

DHCP Start Address: String/IP address

DHCP End Address: String/IP address

DHCP Pool Lease (days): Integer

DHCP Pool Lease (hours): Integer

DHCP Pool Lease (minutes): Integer

**Roles and Policies:**

Session ACL: String

Session ACEs: String/List of Strings (use standard Aruba ACL syntax for each ACE)

Role: String

Role ACLs: String/List of Strings

Role VLAN: String

Role CP Profile: String

Netdest: String

Netdest Hosts: String

Netdest Network: String/IP address

Netdest Network Netmask: String/IP address

Netdest Network Names: String

**WLAN:**

2.4 GHz Minimum: Integer (dBm)

2.4 GHz Maximum: Integer (dBm)

2.4 GHz Channels: Integer/List of Integers

2.4 GHz AM Scan Prof: String

5 GHz Minimum: Integer (dBm)

5 GHz Maximum: Integer (dBm)

5 GHz Channel Width: 20, 40, 80, 160

5 GHz AM Scan Prof: String

WLAN ESSID: String – String \*\*makes SSID profile and Virutal AP profile.

G Rates Required: String/List of Strings

G Rates Allowed: String/List of Strings

A Rates Required: String/List of Strings

A Rates Allowed: String/List of Strings

AP Group: String

AP Group 5 GHz Profile: String

Group VAPs: String/List of Strings

VAP VLAN Mapping: VLAN Name or list of VLAN IDs

Forwarding Mode: Tunnel, Split-Tunnel, Decrypt-Tunnel, Bridge

Frequency Bands: G Only, A Only, All

WMM EAP AC: Default, Best-Effort, Background, Video, Voice

QoS Profile: String

QoS BW Allocation VAP: Integer

QoS BW Allocation Share: Integer (percentage)

MFP/PMF: True/False (MFP capable)

WLAN OPMODE: WPA2-AES, WPA2-PSK-AES, WPA3-SAE-AES, Open, Enhanced Open

Transition Mode: True/False

RF Profile: String \*\*\* Configures G radio profile, A radio profile and regulatory domain profile.

**APs:**

Wired Port Profile: String

Wired Port AAA Prof: String

Wired Port Bridge Role: String

Wired Port Loop Protect: True/False

AP System Profile: String

AP Sys LMS IP: String/IP Address

AP Sys Bkup LMS IP: String/IP Address

AP Sys AP Console PW: String

**Devices:**

Device MAC: String/MAC address

Device Node: String (/md/…)

Device Model: A7005, A7008, A7010, A7024, A7030, A7205, A7210, A7220, A7240, A7240XM, A7280, A9004, A9004-LTE, A9012, MC-VA

MC VLAN ID: Integer

MC Loopback: True/False

Cluster Profile: String

Cluster MC IP: String/IP address

Cluster MC VRRP IP: String/IP address

Cluster VRRP VLAN: String