

mdc-api-python

version 1.0.0

ARM mbed

February 17, 2016

Contents

mdc-api-python docs	1
Intro	1
Requirements	1
Install	1
Use	1
API	2
Async Object	2
Connector API	2
Error Object	5
Examples	6
How to use async objects	6
How to use connectorError objects	6
Long polling	6
Callback URL (webhook)	6
List all endpoints	7
List endpoint resources	7
GET resource value	7
PUT value to resource	7
POST value to resource	7
Subscribe to resource	8
DELETE subscriptions	8
Pre-subscription	8
Enable debug	8
Add notification channel handler	8
Index	10
Python Module Index	11

mdc-api-python docs

Contents:

Intro

The mdc-api-python module is an interface between a python application and the connector.mbed.com REST API. This module enables you to write python programs that use mbed Device Connector. All features of the mbed Device Connector REST interface are exposed through this package.

Requirements

1. Python 2.7.9+
2. [connector.mbed.com](<http://www.connector.mbed.com>) account

Install

Install the `mdc_api` module from pip:

```
pip install -U mdc_api
```

Use

There are five steps to using the library. For more detailed examples please see the [Examples](./Examples) section.

You are **required** to set up a notification channel (step 3), either long polling or callback URL, before using any other mbed Device Connector services.

1. Import the package:

```
import mdc_api
```

2. Initialize an instance with an API token from your [Connector Access Keys](#):

```
x = mdc_api.connector("API TOKEN")
```

3. Set a notification channel (use long polling, or callback URL or a webhook):

```
# Long polling
x.startLongPolling()

# Callback URL or webhook must be able to receive PUT messages
x.putCallback('https://www.mywebapp.com:8080/callback')
```

4. (Optional) Register notification channel handlers for various message types:

```
# Handle 'notifications' messages
def updatesReceived(data):
    print("Received Update : %s", data.result)

# Register function with notification channel router
x.setHandler('notifications', updateReceived)
```

5. Use the API functions to access endpoints and resources:

```
x.getEndpoints()      # Get list of all endpoints on domain
x.getResources("Endpoint") # Get list of all resources on endpoint
x.getResourceValue("Endpoint", "Resource") # Get value of resource
x.putResourceValue("Endpoint", "Resource", "Data") # Set value of resource
x.postResource("Endpoint", "Resource", "OptionalData") # Trigger execution function
```

NOTE: All API functions return `AsyncResult` objects and throw `connectorError` objects.

API

Async Object

`class mdc_api.asyncResult (callback='')`

AsyncResult objects returned by all api_L1 library calls. Make sure to check the `.isDone()` function and the `.error` variable before accessing the `.result` variable.

Variables:

- **error** -- False if no error, if error then populated by :class:'connectorError.response_codes' object
- **result** -- initial value: {}
- **status_code** -- status code returned from REST request
- **raw_data** -- raw returned object form the request

`isDone ()`

Returns: True / False based on completion of async operation

Return type: `bool`

Connector API

`class mdc_api.connector (token, webAddress='https://api.connector.mbed.com', port='80')`

Interface class to use the connector.mbed.com REST API. This class will by default handle asynchronous events. All function return :class:'AsyncResult' objects

`debug (onOff)`

Enable / Disable debugging

Parameters: `onOff (bool)` -- turn debugging on / off

Returns: none

`deleteAllSubscriptions ()`

Delete all subscriptions on the domain (all endpoints, all resources)

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

`deleteCallback ()`

Delete the Callback URL currently registered with Connector.

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

`deleteEndpoint (ep, cbfn='')`

Send DELETE message to an endpoint.

Parameters:

- **ep (str)** -- name of endpoint
- **cbfn (fnptr)** -- Optional - callback funtion to call when operation is completed

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

`deleteEnpointSubscriptions (ep)`

Delete all subscriptions on specified endpoint `ep`

Parameters: `ep (str)` -- name of endpoint

Returns: successful `.status_code` / `.is_done`. Check the `.error`
Return type: `AsyncResult`

deleteResourceSubscription (ep, res)

Delete subscription to a resource `res` on an endpoint `ep`

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource

Returns: successful `.status_code` / `.is_done`. Check the `.error`
Return type: `AsyncResult`

getApiVersions ()

Get the REST API versions that connector accepts.

Returns: `:class:AsyncResult` object, populates error and result fields
Return type: `AsyncResult`

getCallback ()

Get the callback URL currently registered with Connector.

Returns: callback url in `.result`, error if applicable in `.error`
Return type: `AsyncResult`

getConnectorVersion ()

GET the current Connector version.

Returns: `AsyncResult` object, populates error and result fields
Return type: `AsyncResult`

getEndpointSubscriptions (ep)

Get list of all subscriptions on a given endpoint `ep`

Parameters: **ep** (*str*) -- name of endpoint

Returns: successful `.status_code` / `.is_done`. Check the `.error`
Return type: `AsyncResult`

getEndpoints (typeOfEndpoint='')

Get list of all endpoints on the domain.

Parameters: **typeOfEndpoint** (*str*) -- Optional filter endpoints returned by type

Returns: list of all endpoints
Return type: `AsyncResult`

getLimits ()

return limits of account in async result object.

Returns: `AsyncResult` object, populates error and result fields
Return type: `AsyncResult`

getPreSubscription ()

Get the current pre-subscription data from connector

Returns: JSON that represents the pre-subscription data in the `.result` field
Return type: `AsyncResult`

getResourceSubscription (ep, res)

Get list of all subscriptions for a resource `res` on an endpoint `ep`

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

getResourceValue (ep, res, cbfn='', noResp=False, cacheOnly=False)

Get value of a specific resource on a specific endpoint.

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource
- **cbfn** (*fnptr*) -- Optional - callback function to be called on completion
- **noResp** (*bool*) -- Optional - specify no response necessary from endpoint
- **cacheOnly** (*bool*) -- Optional - get results from cache on connector, do not wake up endpoint

Returns: value of the resource, usually a string

Return type: `AsyncResult`

getResources (ep, noResp=False, cacheOnly=False)

Get list of resources on an endpoint.

Parameters:

- **ep** (*str*) -- Endpoint to get the resources of
- **noResp** (*bool*) -- Optional - specify no response necessary from endpoint
- **cacheOnly** (*bool*) -- Optional - get results from cache on connector, do not wake up endpoint

Returns: list of resources

Return type: `AsyncResult`

handler (data)

Function to handle notification data as part of Callback URL handler.

Parameters: **data** (*str*) -- data posted to Callback URL by connector.

Returns: nothing

postResource (ep, res, data='', cbfn='')

POST data to a resource on an endpoint.

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource
- **data** (*str*) -- Optional - data to send via POST
- **cbfn** (*fnptr*) -- Optional - callback function to call when operation is completed

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

putCallback (url, headers='')

Set the callback URL. To be used in place of LongPolling when deploying a webapp.

note: make sure you set up a callback URL in your web app

Parameters:

- **url** (*str*) -- complete url, including port, where the callback url is located
- **headers** (*str*) -- Optional - Headers to have Connector send back with all calls

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

putPreSubscription (JSONdata)

Set pre-subscription rules for all endpoints / resources on the domain. This can be useful for all current and future endpoints/resources.

Parameters: **JSONdata** (*json*) -- data to use as pre-subscription data. Wildcards are permitted

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

putResourceSubscription (ep, res, cbfn='')

Subscribe to changes in a specific resource `res` on an endpoint `ep`

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource
- **cbfn** (*fnptr*) -- Optional - callback funtion to call when operation is completed

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

putResourceValue (ep, res, data, cbfn='')

Put a value to a resource on an endpoint

Parameters:

- **ep** (*str*) -- name of endpoint
- **res** (*str*) -- name of resource
- **data** (*str*) -- data to send via PUT
- **cbfn** (*fnptr*) -- Optional - callback funtion to call when operation is completed

Returns: successful `.status_code` / `.is_done`. Check the `.error`

Return type: `AsyncResult`

setHandler (handler, cbfn)

Register a handler for a particular notification type. These are the types of notifications that are acceptable.

'async-responses'
'registrations-expired'
'de-registrations'
'reg-updates'
'registrations'
'notifications'

Parameters:

- **handler** (*str*) -- name of the notification type
- **cbfn** (*fnptr*) -- function to pass the notification channel messages to.

Returns: Nothing.

startLongPolling (noWait=False)

Start LongPolling Connector for notifications.

Parameters: **noWait** (*bool*) -- Optional - use the cached values in connector, do not wait for the device to respond

Returns: Thread of constantly running LongPoll. To be used to kill the thred if necessary.

Return type: `pythonThread`

stopLongPolling ()

Stop LongPolling thread

Returns: none

Error Object

`class connectorError.response_codes (errParent, status_code)`
 Error class for connector L1 library. Contains the error type, and error string.

Variables:

- **status_code** -- status code returned by connector request
- **errType** -- combination of parent calling function and status code
- **error** -- error given by the <https://docs.mbed.com/docs/mbed-device-connector-web-interface> docs.

Examples

How to use async objects

AsyncResult objects have several useful fields. After each API call you should do the following:

1. Check the `.isDone()` function. This returns `True` when the operation has completed.
2. Check the `.error` variable. If it is set to `False` then there is no error. If it is not `False` then the `.error` variable contains a `connectorError` object.
3. Read the `.result` variable to get the result of the API action.

```
# x is an initialized Connector object
e = x.getEndpoints()
while not e.isDone():
    None
if e.error:
    print("Error : %s", e.error.error)
else:
    print("Result is %s", e.result)
```

How to use connectorError objects

You will probably only encounter this object when something has gone wrong. To find what the error was you can check the `.error` variable, which contains a string representing the error. The `.status_code` variable contains the returned status code related to the error.

```
# x is an initialized mdc_api object
e = x.getEndpoints()
if e.error:
    print e.error.error          # Error message
    print e.error.errType       # Error type
    print e.error.status_code    # Error status code
```

Long polling

When running code on your local machine you will want to use long polling instead of a callback URL (webhook). This is because your local machine does not have a publically addressable IP, so it cannot register a webhook. You should start longpolling before doing any other operations as the long poll will serve as a notification channel between Connector and your app.

```
import mdc_api.connector          # Import library
x = mdc_api.connector("API-Token") # Initialize object
x.startLongPolling()              # Start long polling
# ... Do stuff
```

Callback URL (webhook)

Instead of long polling you can use a callback URL, also known as a webhook. To do this you will need a public web address for your web app and a function that can receive PUT requests. You can use the `.putCallback('url')`

Examples

function to register the callback URL with Connector. It is important that the callback URL be publically accessible, otherwise the registration will fail. The code below assumes you are using the `web.py` library.

<TODO>

List all endpoints

Get all endpoints by using the `getEndpoints()` function.

```
# x is an initialized mdc_api object
r = x.getEndpoints()
while not r.isDone():    # Wait for asynch operation to complete
    None
if r.error:              # Check whether there was an error
    print("Error : %s",r.error.error)
else:
    print r.result       # No error; grab the list of endpoints
```

Example Output:

```
>>> []
```

List endpoint resources

Get all resources on an endpoint by using the `getResources()` function.

```
# x is an initialized mdc_api object
r = x.getResources("endpointName")
while not r.isDone():
    None
if r.error:
    print("Error : %s",r.error.error)
else:
    print r.result
```

Example Output

```
>>> []
```

GET resource value

Get the value of a resource on an endpoint.

```
# Callback function to handle result and pass asyncResult object
def test(data):
    if data.error:
        print("Error: %s", data.error.error)
    else:
        print("Resource Value = %s",data.result)

# x is an initialized mdc_api object
r = x.getResourceValue(ep="EndpointName",res="ResourceName",cbfn=test)
```

PUT value to resource

Change the value of a resource on an endpoint by using PUT.

```
# x is an initialized mdc_api object
r = x.putResourceValue('EndpointName', 'ResourceName', 'DataToSend')
# Check error. Optional: CBFN will be called when operation is completed.
```

POST value to resource

POSTing a value to a resource triggers the associated callback function and passes optional data to it. This method is usually used to trigger events.

```
# x is an initialized mdc_api object
r = x.postResource('EndpointName', 'ResourceName', 'Optional Data')
# Check error. Optional: CBFN will be called when operation is completed.
```

Subscribe to resource

Subscribe to a resource to automatically be notified of changes to resource values. Note that all changes to the resource value are received in the notification channel (long polling or callback URL (webhook)).

```
# x is an initialized mdc_api object
r = x.pubResourceSubscription('endpointName', 'resourceName')
# Check error, or use optional CBFN to handle failure and success.
```

DELETE subscriptions

You can delete subscriptions at three levels.

1. Delete single resource subscription: `deleteResourceSubscription('endpoint', 'resource')`.
2. Delete all subscriptions on an endpoint: `deleteEnpointSubscriptions('endpoint')`.
3. Delete all resource subscriptions on all endpoints on domain: `deleteAllSubscriptions()`.

Pre-subscription

You can use pre-subscriptions to subscribe to all domain resources or endpoints that match a certain pattern. This applies to both existing and future resources.

```
#TODO < CODE HERE>
```

Enable debug

If you want debug messages to be printed to the terminal, you need to enable debug for your `mdc_api` object. By default, debugging displays all notification channel messages.

```
# x is an initialized mdc_api object
x.debug(True) # Turn on debug
```

Add notification channel handler

Add a function to handle different message types on the notification channel. The following notifications types are permitted:

1. `'async-responses'`: handled by `api_L1`, can be overridden.
2. `'registrations-expired'`: endpoint has disappeared.
3. `'de-registrations'`: endpoint has willingly left.
4. `'reg-updates'`: endpoint has pinged Connector.
5. `'registrations'`: new endpoints added to domain.
6. `'notifications'`: subscribed resource value changed.

For more information see the [Connector docs](<https://docs.mbed.com/docs/mbed-device-connector-web-interfaces>).

```
def test(message):
    print("Received Notification message : %s", message)

# x is an initialized mdc_api object
x.sethandler('notifications', test)
```


Index

A

[asyncResult](#) (class in [mdc_api](#))

C

[connector](#) (class in [mdc_api](#))

[connectorError](#) (module)

D

[debug\(\)](#) ([mdc_api.connector](#) method)

[deleteAllSubscriptions\(\)](#) ([mdc_api.connector](#) method)

[deleteCallback\(\)](#) ([mdc_api.connector](#) method)

[deleteEndpoint\(\)](#) ([mdc_api.connector](#) method)

[deleteEnpointSubscriptions\(\)](#) ([mdc_api.connector](#) method)

[deleteResourceSubscription\(\)](#) ([mdc_api.connector](#) method)

G

[getApiVersions\(\)](#) ([mdc_api.connector](#) method)

[getCallback\(\)](#) ([mdc_api.connector](#) method)

[getConnectorVersion\(\)](#) ([mdc_api.connector](#) method)

[getEndpoints\(\)](#) ([mdc_api.connector](#) method)

[getEndpointSubscriptions\(\)](#) ([mdc_api.connector](#) method)

[getLimits\(\)](#) ([mdc_api.connector](#) method)

[getPreSubscription\(\)](#) ([mdc_api.connector](#) method)

[getResources\(\)](#) ([mdc_api.connector](#) method)

[getResourceSubscription\(\)](#) ([mdc_api.connector](#) method)

[getResourceValue\(\)](#) ([mdc_api.connector](#) method)

H

[handler\(\)](#) ([mdc_api.connector](#) method)

I

[isDone\(\)](#) ([mdc_api.asyncResult](#) method)

P

[postResource\(\)](#) ([mdc_api.connector](#) method)

[putCallback\(\)](#) ([mdc_api.connector](#) method)

[putPreSubscription\(\)](#) ([mdc_api.connector](#) method)

[putResourceSubscription\(\)](#) ([mdc_api.connector](#) method)

[putResourceValue\(\)](#) ([mdc_api.connector](#) method)

R

[response_codes](#) (class in [connectorError](#))

S

[setHandler\(\)](#) ([mdc_api.connector](#) method)

[startLongPolling\(\)](#) ([mdc_api.connector](#) method)

[stopLongPolling\(\)](#) ([mdc_api.connector](#) method)

Python Module Index

c

[connectorError](#)