F5 Python SDK Documentation

Release 0.1.3

F5 Networks

April 01, 2016

Contents

CHAPTER 1

Introduction

This project implements an object model based SDK for the F5 Networks® BIG-IP® iControl® REST interface. Users of this library can create, edit, update, and delete configuration objects on a BIG-IP®. For more information on the basic principals that the SDK uses, see the User Guide.

CHAPTER 2

Quick Start

2.1 Installation

```
$> pip install f5-sdk
```

Note: If you are using a pre-release version you must use the `--pre` option with the pip command.

2.2 Basic Example

```
from f5.bigip import BigIP

# Connect to the BigIP
bigip = BigIP("bigip.example.com", "admin", "somepassword")

# Get a list of all pools on the BigIP and print their name and their
# members' name
pools = bigip.ltm.pools.get_collection()
for pool in pools:
    print pool.name
    for member in pool.members_s.get_collection():
        print member.name

# Create a new pool on the BigIP
mypool = bigip.ltm.pools.pool.create(name='mypool', partition='Common')

# Load an existing pool and update its description
pool_a = bigip.ltm.pools.pool.load(name='mypool', partition='Common')
pool_a.description = "New description"
pool_a.update()

# Delete a pool if it exists
if bigip.ltm.pools.pool.exists(name='mypool', partition='Common'):
    pool_b = bigip.ltm.pools.pool.load(name='mypool', partition='Common')
    pool_b.delete()
```

CHAPTER 3

Detailed Documentation

3.1 User Guide

To get the most out of using our SDK, it's useful to understand the basic concepts and principals we used when we designed it. It is also important that you are familiar with the F5® BIG-IP® and, at a minimum, how to configure BIG-IP® using the configuration utility (the GUI). More useful still would be if you are already familiar with the iControl® REST API.

3.1.1 Basic Concepts

Familiarizing yourself with the following underlying basic concepts will help you get up and running with the SDK.

Important: When using the SDK, you'll notice that *collection* objects are referenced using the plural version of the `Resource` objects they contain. When the `Resource` object's type is plural (ends in an `s`), you need to add `_s` to the name when referring to the collection.

This `_s` rule applies to all object collections where the object in the collection already ends in `s`.

Examples:

- LTM Pool objects are collected in `f5.bigip.ltm.pool.Pools` and are accessible via the path `f5.bigip.pools.get_collection()`.
- Network Tunnels objects are stored in `f5.bip.net.tunnels.Tunnels_s` and are accessible via `f5.bigip.net.tunnels_s.get_collection()`.

REST URIs

You can directly infer REST URIs from the python expressions, and vice versa.

Examples

```
Expression:     bigip = BigIP('a', 'b', 'c')
URI Returned:   https://a/mgmt/tm/
```

```
Expression:     bigip.ltm
URI Returned:   https://a/mgmt/tm/ltm/
```

```
Expression:     pools1 = bigip.ltm.pools
URI Returned:   https://a/mgmt/tm/ltm/pool
```

```
Expression:     pool_a = pools1.create(partition="Common", name="foo")
URI Returned:   https://a/mgmt/tm/ltm/pool/~Common~foo
```

REST Endpoints

A set of basic REST endpoints can be derived from the object's URI and `kind` (listed below).

- *Organizing Collection*
- *Collection*
- *Resource*
- *Subcollection*
- *Subcollection Resource*

Dynamic Attributes

The python object's attribute can be created dynamically based on the JSON returned when querying the REST API.

iControl REST `kind` Parameters

Almost all iControl REST API entries contain a parameter named `kind`. This parameter provides information about the object that lets you know what you should expect to follow it. The iControl REST API uses three types of `kind`: `collectionstate`, `state`, and `stats`.

`kind`	Associated Objects	Methods
`collectionstate`	*`OrganizingCollection`*, *`Collection`*	*`exists()`*
`state`	*`Resource`*	*`create()`*, *`update()`*, *`refresh()`*, *`delete()`*, *`load()`*, *`exists()`*
`stats`	*`Resource`*	*`refresh()`*, *`load()`*, *`exists()`*

Methods

Method	HTTP Command	Action(s)
`create()`	POST	creates a new resource on the device with its own URI
`update()`	PUT	submits a new configuration to the device resource; sets the Resource attributes to the state reported by the device
`refresh()`	GET	obtains the state of a device resource; sets the representing Python Resource Object; tracks device state via its attributes
`delete()`	DELETE	removes the resource from the device, sets `self.__dict__` to `{'deleted': True}`
`load()`	GET	obtains the state of an existing resource on the device; sets the Resource attributes to match that state
`exists()`	GET	checks for the existence of a named object on the BIG-IP®

Note: Available methods are restricted according to the object's `kind`.

3.1.2 REST API Endpoints

Overview

REST URI Segments

We'll start exploring the iControl REST API's endpoints with an example detailing how the *endpoint types* map to the different parts of the URI. The different types of resources used by the SDK shown in the example are explained in detail later in this guide.

Example: The URI below returns the JSON for an LTM pool member.

```
http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/members/~Common~m1:80
                    |-------|---|----|--------------|-------|-------------|
                       OC     OC Coll     Resource      SC     SubColl Resrc
```

OC	*Organizing Collection*
Coll	*Collection*
Resource	*Resource*
SC	*Subcollection*
SubColl Resrc	*Subcollection Resource*

Endpoints

Organizing Collection

`kind`: `collectionstate`

The iControl REST User Guide defines an *organizing collection* as a URI that designates all of the `tmsh` subordinate modules and components in the specified module. Organizing collections, which appear directly under *`f5.bigip`*, correspond to the various modules available on the BIG-IP® (for example, *`f5.bigip.ltm`*).

The organizing collection names correspond to the items that appear in the drawers on the left-hand side of the BIG-IP® configuration utility (the GUI). The module names are abbreviated in the REST API, but the mapping is otherwise pretty straightforward. For example, the SDK module *`f5.bigip.sys`* maps to the System drawer in the GUI.

`OrganizingCollection` objects do not have configuration parameters. As shown in the example below, the JSON blob received in response to an `HTTP GET` for an organizing collection object contains an `items` parameter with a list of references to *`Collection`* and *`Resource`* objects.

Example

```
{
    "kind":"tm:ltm:ltmcollectionstate",
    "selfLink":"https://localhost/mgmt/tm/ltm?ver=11.5.0",
    "items":[
        {
        "reference":{
        "link":"https://../mgmt/tm/ltm/auth?ver=11.5.0"
        }
        },
        {
        "reference":{
        "link":"https://../mgmt/tm/ltm/classification?ver=11.5.0"
        }
        },
    ]
}
```

Collection

`kind`: `collectionstate`

A collection is similar to an *Organizing Collection* in that no configurations can be applied to it. A collection differs from an organizing collection in that a collection only contains references to objects of the same type in its `items` parameter.

Important: When using the SDK, you'll notice that *collection* objects are referenced using the plural version of the `Resource` objects they contain. When the `Resource` object's type is plural (ends in an `s`), you need to add `_s` to the name when referring to the collection.

This `_s` rule applies to all object collections where the object in the collection already ends in `s`.

Examples:

- LTM Pool objects are collected in `f5.bigip.ltm.pool.Pools` and are accessible via the path `f5.bigip.pools.get_collection()`.
- Network Tunnels objects are stored in `f5.bip.net.tunnels.Tunnels_s` and are accessible via `f5.bigip.net.tunnels_s.get_collection()`.

You can use `get_collection()` to get a list of the objects in the collection.

The example below shows the JSON you would get back from a REST collection endpoint. Note that it contains an `items` attribute that contains `Resource` objects (we know the objects are resources because their `kind` ends in `state`).

Example

```
{
    kind: "tm:ltm:pool:poolcollectionstate",
    selfLink: "https://localhost/mgmt/tm/ltm/pool?ver=11.6.0",
    items: [
        {
            kind: "tm:ltm:pool:poolstate",
            name: "my_newpool",
            partition: "Common",
            fullPath: "/Common/my_newpool",
            generation: 76,
            selfLink: "https://localhost/mgmt/tm/ltm/pool/~Common~my_newpool?ver=11.6.0"
            allowNat: "yes",
            allowSnat: "yes",
            description: "This is my pool",
            ignorePersistedWeight: "disabled",
            ipTosToClient: "pass-through",
            ipTosToServer: "pass-through",
            linkQosToClient: "pass-through",
            linkQosToServer: "pass-through",
            loadBalancingMode: "round-robin",
            minActiveMembers: 0,
            minUpMembers: 0,
            minUpMembersAction: "failover",
            minUpMembersChecking: "disabled",
            queueDepthLimit: 0,
            queueOnConnectionLimit: "disabled",
            queueTimeLimit: 0,
            reselectTries: 0,
            serviceDownAction: "none",
            slowRampTime: 10,
            membersReference: {
            link: "https://localhost/mgmt/tm/ltm/pool/~Common~my_newpool/members?ver=11.6.0",
            isSubcollection: true
            }
        },
        {
            kind: "tm:ltm:pool:poolstate",
            name: "mypool",
            partition: "Common",
            fullPath: "/Common/mypool",
            generation: 121,
            selfLink: "https://localhost/mgmt/tm/ltm/pool/~Common~mypool?ver=11.6.0",
            allowNat: "yes",
            allowSnat: "yes",
            ignorePersistedWeight: "disabled",
            ipTosToClient: "pass-through",
            ipTosToServer: "pass-through",
            linkQosToClient: "pass-through",
            linkQosToServer: "pass-through",
            loadBalancingMode: "round-robin",
            minActiveMembers: 0,
            minUpMembers: 0,
            minUpMembersAction: "failover",
            minUpMembersChecking: "disabled",
            queueDepthLimit: 0,
            queueOnConnectionLimit: "disabled",
            queueTimeLimit: 0,
            reselectTries: 0,
            serviceDownAction: "none",
            slowRampTime: 10,
            membersReference: {
            link: "https://localhost/mgmt/tm/ltm/pool/~Common~mypool/members?ver=11.6.0"
            isSubcollection: true
```

Resource

`kind`: `state`

A resource is a fully configurable object for which the CURDLE *methods* are supported.

- *`create()`*
- *`refresh()`*
- *`update()`*
- *`delete()`*
- *`load()`*
- *`exists()`*

When using the SDK, you will notice that resources are instantiated via their *collection*. Once created or loaded, resources contain attributes that map to the JSON fields returned by the BIG-IP®.

Example

To load a *`f5.bigip.ltm.node.Node`* object, you would use the following code.

```
>>> from f5.bigip import BigIP
>>> bigip = BigIP('192.168.1.1', 'myuser', 'mypass')
>>> n = bigip.ltm.nodes.node.load(partition='Common', name='192.168.15.15')
>>> print n.raw
{
   "kind":"tm:ltm:node:nodestate",
   "name":"192.168.15.15",
   "partition":"Common",
   "fullPath":"/Common/192.168.15.15",
   "generation":16684,
   "selfLink":"https://localhost/mgmt/tm/ltm/node/~Common~192.168.15.15?ver=11.6.0",
   "address":"192.168.15.15",
   "connectionLimit":0,
   "dynamicRatio":1,
   "ephemeral":"false",
   "fqdn":{
     "addressFamily":"ipv4",
     "autopopulate":"disabled",
     "downInterval":5,
     "interval":3600
   },
   "logging":"disabled",
   "monitor":"default",
   "rateLimit":"disabled",
   "ratio":1,
   "session":"user-enabled",
   "state":"unchecked"
}
```

The output of the *`f5.bigip.ltm.node.Node.raw`* shows all of the available attributes.
Once you have loaded the object, you can access the attributes as shown below.

```
>>> n.fqdn['downInterval'] = 10
>>> n.logging = 'enabled'
>>> n.update()
```

Subcollection

`kind`: `collectionstate`

A subcollection is a *`Collection`* that's attached to a higher-level *`Resource`* object. Subcollections are almost exactly the same as collections; the exception is that they can only be accessed via the resource they're attached to (the 'parent' resource). A subcollection can be identified by the value `isSubcollection: true`, followed by an `items` attribute listing the subcollection's resources. Just as with collections, you can use *`get_collection()`* to get a list of the resources in the subcollection.

Example

A `pool` resource has a `members_s` subcollection attached to it; you must create or load the 'parent' resource (`pool`) before you can access the subcollection (`members_s`).

```
>>> from f5.bigip import BigIP
>>> bigip = BigIP('192.168.1.1', 'myuser', 'mypass')
>>> pool = bigip.ltm.pools.pool.load(partition='Common', name='p1')
>>> members = pool.members_s.get_collection()
```

Note: In the above example, the subcollection object – `members_s` – ends in `_s` because the subcollection resource object name (`members`) is already plural.

The JSON returned for a pool with one member is shown below. Note the highlighted rows, which indicate the subcollection.

Example

```
{
    "kind": "tm:ltm:pool:poolstate",
    "name": "p1",
    "partition": "Common",
    "fullPath": "/Common/p1",
    "generation": 18703,
    "selfLink": "https://localhost/mgmt/tm/ltm/pool/~Common~p1?expandSubcollections=true&ver=11.6.0"
    "allowNat": "yes",
    "allowSnat": "yes",
    "ignorePersistedWeight": "disabled",
    "ipTosToClient": "pass-through",
    "ipTosToServer": "pass-through",
    "linkQosToClient": "pass-through",
    "linkQosToServer": "pass-through",
    "loadBalancingMode": "round-robin",
    "minActiveMembers": 0,
    "minUpMembers": 0,
    "minUpMembersAction": "failover",
    "minUpMembersChecking": "disabled",
    "queueDepthLimit": 0,
    "queueOnConnectionLimit": "disabled",
    "queueTimeLimit": 0,
    "reselectTries": 0,
    "serviceDownAction": "none",
    "slowRampTime": 10,
    "membersReference": {
        "link": "https://localhost/mgmt/tm/ltm/pool/~Common~p1/members?ver=11.6.0",
        "isSubcollection": true,
        "items": [
              {
                "kind": "tm:ltm:pool:members:membersstate",
                "name": "n1:80",
                "partition": "Common",
                "fullPath": "/Common/n1:80",
                "generation": 18703,
                "selfLink": "https://localhost/mgmt/tm/ltm/pool/~Common~p1/members/~Common~n1:80?ver
                "address": "192.168.51.51",
                "connectionLimit": 0,
                "dynamicRatio": 1,
                "ephemeral": "false",
                "fqdn": {
                  "autopopulate": "disabled",
                }
                "inheritProfile": "enabled",
                "logging": "disabled",
                "monitor": "default",
                "priorityGroup": 0,
                "rateLimit": "disabled",
                "ratio": 1,
                "session": "user-enabled",
                "state": "unchecked",
              }
        ]
    },
}
```

Subcollection Resource

`kind`: `state`

A subcollection resource is essentially the same as a *resource*. As with collections and subcollections, the only difference between the two is that you must access the subcollection resource via the subcollection attached to the main resource.

Example

To build on the *subcollection example*: `pool` is the resource, `members_s` is the subcollection, and `members` (the actual pool member) is the subcollection resource.

```
>>> from f5.bigip import BigIP
>>> bigip = BigIP('192.168.1.1', 'myuser', 'mypass')
>>> pool = bigip.ltm.pools.pool.load(partition='Common', name='p1')
>>> member = pool.members_s.members.load(partition='Common', name='n1:80')
```

The JSON below shows a `f5.bigip.ltm.pool.members_s.members` object.

```
{
    "kind": "tm:ltm:pool:members:membersstate",
    "name": "n1:80",
    "partition": "Common",
    "fullPath": "/Common/n1:80",
    "generation": 18703,
    "selfLink": "https://localhost/mgmt/tm/ltm/pool/~Common~p1/members/~Common~n1:80?ver=11.6.0",
    "address": "192.168.51.51",
    "connectionLimit": 0,
    "dynamicRatio": 1,
    "ephemeral": "false",
    "fqdn": {
      "autopopulate": "disabled",
    }
    "inheritProfile": "enabled",
    "logging": "disabled",
    "monitor": "default",
    "priorityGroup": 0,
    "rateLimit": "disabled",
    "ratio": 1,
    "session": "user-enabled",
    "state": "unchecked",
}
```

Tip: It's easy to tell that this is a Resource object because the `kind` is `state`, not `collectionstate`.

3.1.3 Python Object Paths

The object classes used in the SDK directly correspond to the REST endpoints you'd use to access the objects via the API. Remembering the patterns below will help you easily derive an SDK object class from an object URI.

1. Objects take the form `f5.<product>.<organizing_collection>.<collection>.<resource>.<subcollect`
2. The collection and the resource generally have the same name, so the collection is the *plural* version of the resource. This means that you add `s` to the end of the resource to get the collection, *unless* the resource already

ends in `s`. If the resource is already plural, add `_s` to get the collection.

3. The object itself is accessed by its CamelCase name, but the usage of the object is all lowercase.
4. The characters `.` and `-` are always replaced with `_` in the SDK.

Because the REST API endpoints have a hierarchical structure, you need to load/create the highest-level objects before you can load lower-level ones. The example below shows how the pieces of the URI correspond to the REST endpoints/SDK classes. The first part of the URI is the IP address of your BIG-IP®.

```
http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/members/~Common~m1:80
                         |-------|---|----|--------------|-------|-------------|
                            OC     OC Coll     Resource      SC     SubColl Resrc
```

OC	*Organizing Collection*
Coll	*Collection*
Resource	*Resource*
SC	*Subcollection*
SubColl Resrc	*Subcollection Resource*

In the sections below, we'll walk through the Python object paths using LTM® pools and pool members as examples. You can also skip straight to the *Coding Example*.

Organizing Collection

The `mgmt/tm` and `ltm` organizing collections define what area of the BIG-IP® you're going to work with. The `mgmt/tm` organizing collection corresponds to the management plane of your BIG-IP® device (TMOS). Loading `ltm` indicates that we're going to work with the BIG-IP®'s *Local Traffic Manager*® module.

Endpoint	http://192.168.1.1/mgmt/tm/
Kind	`tm:restgroupresolverviewstate`
Type	organizing collection
Class	*`f5.bigip.BigIP`*
Instantiation	`bigip = BigIP('192.168.1.1', 'myuser', 'mypass')`

Endpoint	http://192.168.1.1/mgmt/tm/ltm
Kind	`tm:ltm:collectionstate`
Type	organizing collection
Class	*`f5.bigip.ltm`*
Instantiation	`ltm = bigip.ltm`

Example: Connect to the BIG-IP® and load the LTM® module

```
from f5.bigip import BigIP
bigip = BigIP('192.168.1.1', 'myuser', 'mypass')
ltm = bigip.ltm

>>> print bigip
<f5.bigip.BigIP object at 0x8a29d0>

>>> print ltm
<f5.bigip.ltm.LTM object at 0x8c0b30>
```

Collection

Now that the higher-level organizing collections are loaded (in other words, we signed in to the BIG-IP® and accessed the LTM® module), we can load the `pool` collection.

Endpoint	http://192.168.1.1/mgmt/tm/ltm/pool
Kind	`tm:ltm:pool:poolcollectionstate`
Type	collection
Class	*`f5.bigip.ltm.pool.Pools`*
Instantiation	`pools = bigip.ltm.pools`

Example: Load the pools collection

```
from f5.bigip import BigIP

bigip = BigIP('192.168.1.1', 'myuser', 'mypass')
pool_collection = bigip.ltm.pools
pools = bigip.ltm.pools.get_collection()

for pool in pools:
    print pool.name

my_newpool
mypool
pool2
pool_1
```

In the above example, we instantiated the class *`f5.bigip.ltm.pool.Pools`*, then used the *`f5.bigip.ltm.pool.Pools.get_collection()`* method to fetch the collection (in other words, a list of the pool *resources* configured on the BIG-IP®).

Resource

In the SDK, we refer to a single instance of a configuration object as a resource. As shown in the previous sections, we are able to access the `pool` resources on the BIG-IP® after loading the `mgmt\tm\ltm` organizing collections and the `pools` collection.

Endpoint	http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/
Kind	`tm:ltm:pool:poolstate`
Type	resource
Class	*`f5.bigip.ltm.pool.Pool`*
Instantiation	`pool = pools.pool.load(partition='Common', name='mypool')`

Example: Load a pool resource

```
from f5.bigip import BigIP
pool = pools.pool.load(partition='Common', name='mypool')
```

In the example above, we instantiated the class *`f5.bigip.ltm.pool.Pool`* and loaded the `f5.bigip.ltm.pools.pool` object. The object is a python representation of the BIG-IP® pool we loaded (in this case, `Common/mypool`).

Tip: You can always see the representation of an object using the *`raw()`* method.

```
>>> pool.raw
{
 u'generation': 123,
 u'minActiveMembers': 0,
 u'ipTosToServer': u'pass-through',
 u'loadBalancingMode': u'round-robin',
 u'allowNat': u'yes',
 u'queueDepthLimit': 0,
 u'membersReference': {
    u'isSubcollection': True,
    u'link': u'https://localhost/mgmt/tm/ltm/pool/~Common~mypool/members?ver=11.6.0'},
    u'minUpMembers': 0, u'slowRampTime': 10,
    u'minUpMembersAction': u'failover',
    '_meta_data': {
        'attribute_registry': {
            'tm:ltm:pool:memberscollectionstate': <class 'f5.bigip.ltm
        .pool.Members_s'>
        },
        'container': <f5.bigip.ltm.pool.Pools object at 0x835ef0>,
        'uri': u'https://10.190.6.253/mgmt/tm/ltm/pool/~Common~mypool/',
        'exclusive_attributes': [],
        'read_only_attributes': [],
        'allowed_lazy_attributes': [<class 'f5.bigip.ltm.pool.Members_s'>],
        'required_refresh_parameters': set(['name']),
        'required_json_kind': 'tm:ltm:pool:poolstate',
        'bigip': <f5.bigip.BigIP object at 0x5826f0>,
        'required_creation_parameters': set(['name']),
        'creation_uri_frag': '',
        'creation_uri_qargs': {u'ver': [u'11.6.0']}
    },
    u'minUpMembersChecking': u'disabled',
    u'queueTimeLimit': 0,
    u'linkQosToServer': u'pass-through',
    u'queueOnConnectionLimit': u'disabled',
    u'fullPath': u'/Common/mypool',
    u'kind': u'tm:ltm:pool:poolstate',
    u'name': u'mypool',
    u'partition': u'Common',
    u'allowSnat': u'yes',
    u'ipTosToClient': u'pass-through',
    u'reselectTries': 0,
    u'selfLink': u'https://localhost/mgmt/tm/ltm/pool/~Common~mypool?ver=11.6.0',
    u'serviceDownAction': u'none',
    u'ignorePersistedWeight': u'disabled',
    u'linkQosToClient': u'pass-through'
   }
```

Subcollection

A subcollection is a collection of resources that can only be accessed via its parent resource.

To continue our example: The *f5.bigip.ltm.pool.Pool* resource object contains `f5.bigip.ltm.pool.Member` *subcollection resource* objects. These subcollection resources – the real-servers that are attached to the pool, or 'pool members' – are part of the `members_s` subcollection. (Remember, we have to add `_s` to the end of collection object names if the name of the resource object it contains already ends in `s`).

Endpoint	http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/members
Kind	`tm:ltm:pool:members:memberscollectionstate`
Type	subcollection
Class	*f5.bigip.ltm.pool.Members_s*
Instantiation	`members = pool.members_s`

Example: Load the members_s collection

```
from f5.bigip import BigIP
members = pool.members_s.get_collection()
print members
[<f5.bigip.ltm.pool.Members object at 0x9d7ff0>, <f5.bigip.ltm.pool.Members object at 0x9d7830>]
```

Subcollection Resource

As explained in the previous section, a subcollection contains subcollection resources. These subcollection resources can only be loaded after all of the parent objects (organizing collections, resource, and subcollection) have been loaded.

Endpoint	http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/members/~Common~member1
Kind	`tm:ltm:pool:members:membersstate`
Type	subcollection resource
Class	*f5.bigip.ltm.pool.Members*
Instantiation	`members = pool.members_s.members.load(partition='Common', name='member1:<port>')`

Example: Load member objects

```
from f5.bigip import BigIP
member = members_s.members.load(partition='Common', name='m1')
print member
<f5.bigip.ltm.pool.Members object at 0x9fd530>
```

Coding Example

3.1.4 Coding Example

Managing LTM Pools and Members via the F5 SDK

```
from f5.bigip import BigIP

# Connect to the BigIP and configure the basic objects
bigip = BigIP('10.190.6.253', 'admin', 'default')
ltm = bigip.ltm
pools = bigip.ltm.pools.get_collection()
pool = bigip.ltm.pools.pool

# Define a pool object and load an existing pool
pool_obj = bigip.ltm.pools.pool
pool_1 = pool_obj.load(partition='Common', name='mypool')

# We can also skip creating the object and load the pool directly
pool_2 = bigip.ltm.pools.pool.load(partition='Common', name='mypool')

# Print the object
print pool_1.raw

# Make sure 1 and 2 have the same names and generation
assert pool_1.name == pool_2.name
assert pool_1.generation == pool_2.generation

# Update the description
pool_1.description = "This is my pool"
pool_1.update()

# Check the updated description
print pool_1.description

# Since we haven't refreshed pool_2 it shouldn't match pool_1 any more
assert pool_1.generation > pool_2.generation

# Refresh pool_2 and check that is now equal
pool_2.refresh()
assert pool_1.generation == pool_2.generation

print pool_1.generation
print pool_2.generation

# Create members on pool_1

members = pool_1.members_s.get_collection()
member = pool_1.members_s.members

m1 = pool_1.members_s.members.create(partition='Common', name='m1:80')
m2 = pool_1.members_s.members.create(partition='Common', name='m2:80')

# load the pool members
m1 = pool_1.members_s.members.load(partition='Common', name='m1:80')
m2 = pool_1.members_s.members.load(partition='Common', name='m2:80')

# Get all of the pool members for pool_1 and print their names

for member in members:
    print member.name

# Delete our pool member m1
m1.delete()

# Make sure it is gone
if pool_1.members_s.exists(partition='Common', name='m1:80'):
    raise Exception("Object should have been deleted")
```

3.1.5 Further Reading

- F5 SDK API Docs
- F5 iControl REST DevCentral Site
- F5 iControl REST API Reference (PDF)
- ‘F5 iControl REST API Guide (PDF) <https://devcentral.f5

.com/d/the-user-guide-for-the-icontrol-rest-interface-in-big-ip-version-1160?download=true>‘_

3.2 Developer Guide

COMING SOON

3.3 f5

3.3.1 f5 package

f5.bigip

f5.bigip module

Classes and functions for configuring BIG-IP

cm	BIG-IP® cluster module
ltm	BIG-IP® Local Traffic Manager™ (LTM®) module.
net	BIG-IP® net module
shared	BIG-IP® Shared (shared) module
sys	BIG-IP® System (sys) module

Organizing Collection Modules

resource.ResourceBase(container)	Base class for all BIG-IP® iControl REST API endpoints.
resource.OrganizingCollection(bigip)	Base class for objects that collect resources under them.
resource.Collection(container)	Base class for objects that collect a list of `Resources`
resource.Resource(container)	Base class to represent a Configurable Resource on the device.
resource.PathElement(container)	Base class to represent a URI path element that does not contain data.

Resource Base Classes

resource.KindTypeMismatch	Raise this when server JSON keys are incorrect for the Resource type
resource.DeviceProvidesIncompatibleKey	Raise this when server JSON keys are incompatible with Python.
resource.InvalidResource	Raise this when a caller tries to invoke an unsupported CRUDL op.
resource.MissingRequiredCreationParameter	Various values MUST be provided to create different Resources.

Continued on next page

Table 3.3 – continued from previous page

resource.MissingRequiredReadParameter	Various values MUST be provided to refresh some Resources.
resource.UnregisteredKind	The returned server JSON *kind* key wasn't expected by this Resource
resource.GenerationMismatch	The server reported BIG-IP® is not the expacted value.
resource.InvalidForceType	Must be of type bool.
resource.URICreationCollision	self._meta_data['uri'] can only be assigned once. In create or load.
resource.UnsupportedOperation	Object does not support the method that was called.

Resource Exceptions

mixins.ToDictMixin	Convert an object's attributes to a dictionary
mixins.LazyAttributesMixin	
mixins.ExclusiveAttributesMixin	Overrides `__setattr__` to remove exclusive attrs from the object.
mixins.UnnamedResourceMixin	This makes a resource object work if there is no name.
mixins.LazyAttributesRequired	Raised when a object accesses a lazy attribute that is not listed

Mixins

class f5.bigip.**BigIP**(*hostname*, *username*, *password*, ***kwargs*)

Bases: *f5.bigip.resource.OrganizingCollection*

An interface to a single BIG-IP

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Call to obtain a list of the reference dicts in the instance *items*

Returns List of self.items

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

f5.bigip.cm

Module Contents BIG-IP® cluster module

REST URI `http://localhost/mgmt/tm/cm/`

GUI Path `Device Management`

REST Kind `tm:cm:*`

device	BIG-IP® cluster device submodule
device_group	BIG-IP® cluster device-group submodule
traffic_group	BIG-IP® cluster traffic-group submodule

Submodule List

class `f5.bigip.cm.`**`Cm`**(*bigip*)

Bases: *f5.bigip.resource.OrganizingCollection*

BIG-IP® Cluster Organizing Collection.

`sync`(*device_group_name*)

Sync the configuration of the device-group.

Execute the run command via the iControl REST session with the config-sync to group device-group options. Any exceptions triggered by the POST to the iControl REST server are raised back to the caller.

Parameters **`device_group_name`** (*str*) – Name of the device group to sync.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Call to obtain a list of the reference dicts in the instance *items*

Returns List of self.items

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

Submodules

device BIG-IP® cluster device submodule

REST URI `http://localhost/mgmt/tm/cm/device/`

GUI Path `Device Management --> Devices`

REST Kind `tm:cm:device:*`

class `f5.bigip.cm.device.`**`Devices`**`(`*cm*`)`
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® cluster devices collection.

`create``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection``(`***kwargs*`)`
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh``(`***kwargs*`)`
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.cm.device.`**`Device`**`(`*device_s*`)`
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® cluster device object.

`create``(`***kwargs*`)`
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

device_group BIG-IP® cluster device-group submodule

REST URI `http://localhost/mgmt/tm/cm/device-group`

GUI Path `Device Management --> Device Groups`

REST Kind `tm:cm:device-group:*`

class f5.bigip.cm.device_group.**Device_Groups**(*cm*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® cluster device-groups collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.cm.device_group.`**`Device_Group`**(*device_groups*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® cluster device-group resource

sync()

Sync the configuration of the device-group

Executes the containing object's cm *`sync()`* method to sync the configuration of the device-group.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters kwargs – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters kwargs – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.cm.device_group.`**Devices_s**(*device_group*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® cluster devices-group devices subcollection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.cm.device_group.`**`Devices`**(*devices_s*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® cluster devices-group devices subcollection resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

traffic_group BIG-IP® cluster traffic-group submodule

REST URI `http://localhost/mgmt/tm/cm/traffic-group`

GUI Path `Device Management --> Traffic Groups`

REST Kind `tm:cm:traffic-group:*`

class `f5.bigip.cm.traffic_group.`**`Traffic_Groups`**(*cm*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® cluster traffic-group collection

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.cm.traffic_group.`**`Traffic_Group`**(*traffic_groups*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® cluster traffic-group resource

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

f5.bigip.ltm

Module Contents BIG-IP® Local Traffic Manager™ (LTM®) module.

REST URI `http://localhost/mgmt/tm/ltm/`

GUI Path `Local Traffic`

REST Kind

`tm:ltm:*`

monitor	BIG-IP® LTM monitor submodule.
nat	BIG-IP® Local Traffic Manager (LTM) Nat module.
node	BIG-IP® Local Traffic Manager (LTM) node module.
policy	BIG-IP® Local Traffic Manager (LTM) policy module.
pool	BIG-IP® Local Traffic Manager™ (LTM®) pool module.
rule	BIG-IP® Local Traffic Manager (LTM) rule module.
snat	BIG-IP® Local Traffic Manager (LTM) Snat module.
snatpool	BIG-IP Local Traffic Manager (LTM) SNAT pool module.
snat_translation	BIG-IP Local Traffic Manager (LTM) SNAT Translation module.
ssl	This module provides some more Pythonic support for SSL.
virtual	BIG-IP® Local Traffic Manager (LTM) virtual module.
virtual_address	Directory: ltm module: virtual-address.

class f5.bigip.ltm.**Ltm**(*bigip*)

Bases: *f5.bigip.resource.OrganizingCollection*

BIG-IP® Local Traffic Manager (LTM) organizing collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Call to obtain a list of the reference dicts in the instance *items*

Returns List of self.items

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

Submodules

monitor BIG-IP® LTM monitor submodule.

REST URI `http://localhost/mgmt/tm/ltm/monitors/`

GUI Path `Local Traffic --> Monitors`

REST Kind `tm:ltm:monitors*`

Https(monitor)	BIG-IP® Http monitor collection.
Http(https)	BIG-IP® Http monitor resource.
Https_s(monitor)	BIG-IP® Https monitor collection.
HttpS(https_s)	BIG-IP® Https monitor resource.
Diameters(monitor)	BIG-IP® diameter monitor collection.
Diameter(diameters)	BIG-IP® diameter monitor resource.
Dns_s(monitor)	BIG-IP® Dns monitor collection.
Dns(dns_s)	BIG-IP® Dns monitor resource.
Externals(monitor)	BIG-IP® external monitor collection.
External(externals)	BIG-IP® external monitor resrouce.
Firepass_s(monitor)	BIG-IP® Fire Pass monitor collection.
Firepass(firepass_s)	BIG-IP® external monitor resource.
Ftps(monitor)	BIG-IP® Ftp monitor collection.
Ftp(ftps)	BIG-IP® Ftp monitor resource.
Gateway_Icmps(monitor)	BIG-IP® Gateway Icmp monitor collection.
Gateway_Icmp(gateway_icmps)	BIG-IP® Gateway Icmp monitor resource.
Icmps(monitor)	BIG-IP® Icmp monitor collection.
Icmp(icmps)	BIG-IP® Icmp monitor resource.
Imaps(monitor)	BIG-IP® Imap monitor collection.
Imap(imaps)	BIG-IP® Imap monitor resource.
Inbands(monitor)	BIG-IP® in band monitor collection.
Inband(inbands)	BIG-IP® in band monitor resource.
Ldaps(monitor)	BIG-IP® Ldap monitor collection.
Ldap(ldaps)	BIG-IP® Ldap monitor resource.
Module_Scores(monitor)	BIG-IP® module scores monitor collection.

Continued on next page

Table 3.7 – continued from previous page

Module_Score(gateway_icmps)	BIG-IP® module scores monitor resource.
Mssqls(monitor)	BIG-IP® Mssql monitor collection.
Mssql(mssqls)	BIG-IP® Mssql monitor resource.
Mysqls(monitor)	BIG-IP® MySQL monitor collection.
Mysql(mysqls)	BIG-IP® MySQL monitor resource.
Nntps(monitor)	BIG-IP® Nntps monitor collection.
Nntp(nntps)	BIG-IP® Nntps monitor resource.
Nones(monitor)	BIG-IP® None monitor collection.
NONE(nones)	BIG-IP® None monitor resource.
Oracles(monitor)	BIG-IP® Oracle monitor collection.
Oracle(oracles)	BIG-IP® Oracle monitor resource.
Pop3s(monitor)	BIG-IP® Pop3 monitor collection.
Pop3(pop3s)	BIG-IP® Pop3 monitor resource.
Postgresqls(monitor)	BIG-IP® PostGRES SQL monitor collection.
Postgresql(postgresqls)	BIG-IP® PostGRES SQL monitor resource.
Radius_s(monitor)	BIG-IP® radius monitor collection.
Radius(radius_s)	BIG-IP® radius monitor resource.
Radius_Accountings(monitor)	BIG-IP® radius accounting monitor collection.
Radius_Accounting(radius_accountings)	BIG-IP® radius accounting monitor resource.
Real_Servers(monitor)	BIG-IP® real-server monitor collection.
Real_Server(real_servers)	BIG-IP® real-server monitor resource.
Rpcs(monitor)	BIG-IP® Rpc monitor collection.
Rpc(rpcs)	BIG-IP® Rpc monitor resource.
Sasps(monitor)	BIG-IP® Sasp monitor collection.
Sasp(sasps)	BIG-IP® Sasp monitor resource.
Scripteds(monitor)	BIG-IP® scripted monitor collection.
Scripted(scripteds)	BIG-IP® scripted monitor resource.
Sips(monitor)	BIG-IP® Sip monitor collection.
Sip(sips)	BIG-IP® Sip monitor resource.
Smbs(monitor)	BIG-IP® Smb monitor collection.
Smb(smbs)	BIG-IP® Smb monitor resource.
Smtps(monitor)	BIG-IP® Smtp monitor collection.
Smtp(smtps)	BIG-IP® Smtp monitor resource.
Snmp_Dcas(monitor)	BIG-IP® SNMP DCA monitor collection.
Snmp_Dca(snmp_dcas)	BIG-IP® SNMP DCA monitor resource.
Snmp_Dca_Bases(monitor)	BIG-IP® SNMP DCA bases monitor collection.
Snmp_Dca_Base(snmp_dca_bases)	BIG-IP® SNMP DCA monitor resource.
Soaps(monitor)	BIG-IP® Soap monitor collection.
Soap(soaps)	BIG-IP® Soap monitor resource.
Tcps(monitor)	BIG-IP® Tcp monitor collection.
Tcp(tcps)	BIG-IP® Tcp monitor resource.
Tcp_Echos(monitor)	BIG-IP® Tcp echo monitor collection.
Tcp_Echo(tcp_echos)	BIG-IP® Tcp echo monitor resource.
Tcp_Half_Opens(monitor)	BIG-IP® Tcp half open monitor collection.
Tcp_Half_Open(tcp_half_opens)	BIG-IP® Tcp half open monitor resource.
Udps(monitor)	BIG-IP® Udp monitor collection.
Udp(udps)	BIG-IP® Udp monitor resource.
Virtual_Locations(monitor)	BIG-IP® virtual-locations monitor collection.
Virtual_Location(virtual_locations)	BIG-IP® virtual-locations monitor resource.
Waps(monitor)	BIG-IP® Wap monitor collection.

Continued on next page

Table 3.7 – continued from previous page

Wap(waps)	BIG-IP® Wap monitor resource.
Wmis(monitor)	BIG-IP® Wmi monitor collection.
Wmi(wmis)	BIG-IP® Wmi monitor resource.

Monitor Collections and Resources

class `f5.bigip.ltm.monitor.`**`Https`**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Http monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Http`**(*https*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *f5.bigip.resource.Resource*

BIG-IP® Http monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Https_s`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Https monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**HttpS**(*https_s*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Https monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh``(`***kwargs*`)`
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update``(`***kwargs*`)`
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Diameters`**`(`*monitor*`)`
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® diameter monitor collection.

`create``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection``(`***kwargs*`)`
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Diameter`**(*diameters*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® diameter monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Dns_s`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Dns monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Dns`**(*dns_s*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Dns monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**Externals**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® external monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**External**(*externals*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® external monitor resrouce.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Firepass_s`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Fire Pass monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Firepass`**(*firepass_s*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® external monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Ftps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Ftp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Ftp`**(*ftps*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Ftp monitor resource.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Gateway_Icmps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Gateway Icmp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Gateway_Icmp`**(*gateway_icmps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Gateway Icmp monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Icmps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Icmp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Icmp`**(*icmps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Icmp monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**Imaps**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Imap monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource'̀s that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Imap`**(*imaps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Imap monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Inbands`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® in band monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Inband`**(*inbands*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® in band monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Ldaps**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Ldap monitor collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Ldap`**(*ldaps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Ldap monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters kwargs – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Module_Scores`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® module scores monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Module_Score`**(*gateway_icmps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® module scores monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Mysqls**(*monitor*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® MySQL monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Mysql`**(*mysqls*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® MySQL monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Mssqls**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Mssql monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Mssql`**(*mssqls*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Mssql monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Nntps`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Nntps monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Nntp`**(*nntps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Nntps monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Nones`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® None monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**NONE**(*nones*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® None monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters kwargs – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Oracles**(*monitor*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® Oracle monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Oracle**(*oracles*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Oracle monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**Pop3s**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Pop3 monitor collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Pop3`**(*pop3s*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Pop3 monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**Postgresqls**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® PostGRES SQL monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Postgresql**(*postgresqls*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® PostGRES SQL monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Radius_s`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® radius monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Radius`**(*radius_s*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® radius monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Radius_Accountings**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® radius accounting monitor collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Radius_Accounting`**(*radius_accountings*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® radius accounting monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Real_Servers**(*monitor*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® real-server monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Real_Server**(*real_servers*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® real-server monitor resource.

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `tmCommand` attribute removed prior to PUT
- `agent` attribute removed prior to PUT
- `post` attribute removed prior to PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

class f5.bigip.ltm.monitor.**Rpcs**(*monitor*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® Rpc monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Rpc`**(*rpcs*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Rpc monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**Sasps**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Sasp monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Sasp`**(*sasps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Sasp monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Scripteds`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® scripted monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Scripted`**(*scripteds*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® scripted monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Sips**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Sip monitor collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Sip**(*sips*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, `f5.bigip.resource.Resource`

BIG-IP® Sip monitor resource.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

•read-only attributes that are unchangeable are removed

•`defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Smbs`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Smb monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Smb`**(*smbs*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Smb monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Smtps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Smtp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Smtp`**(*smtps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Smtp monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError‘ exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains “name” and “partition”

NOTE: If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data[’uri’])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Snmp_Dcas**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® SNMP DCA monitor collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Snmp_Dca**(*snmp_dcas*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® SNMP DCA monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object’s

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Snmp_Dca_Bases`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® SNMP DCA bases monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Snmp_Dca_Base`**(*snmp_dca_bases*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® SNMP DCA monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **kwargs** – keys and associated values to alter on the device

class f5.bigip.ltm.monitor.**Soaps**(*monitor*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® Soap monitor collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Soap**(*soaps*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Soap monitor resource.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Tcps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Tcp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Tcp`**(*tcps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Tcp monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Tcp_Echos`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Tcp echo monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Tcp_Echo`**(*tcp_echos*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Tcp echo monitor resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters kwargs – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters kwargs – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Tcp_Half_Opens`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Tcp half open monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Tcp_Half_Open`**(*tcp_half_opens*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Tcp half open monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters `kwargs` – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Udps`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Udp monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Udp`**(*udps*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Udp monitor resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Virtual_Locations`**(*monitor*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® virtual-locations monitor collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Virtual_Location`**(*virtual_locations*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® virtual-locations monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Waps`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Wap monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**Wap**(*waps*)
Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Wap monitor resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `defaultsFrom` attribute is removed from JSON before the PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

class `f5.bigip.ltm.monitor.`**`Wmis`**(*monitor*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® Wmi monitor collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.monitor.`**`Wmi`**(*wmis*)

Bases: `f5.bigip.ltm.monitor.UpdateMonitorMixin`, *`f5.bigip.resource.Resource`*

BIG-IP® Wmi monitor resource.

`update`(***kwargs*)

Change the configuration of the resource on the device.

This method uses Http PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- `agent` attribute removed prior to PUT
- `post` attribute removed prior to PUT
- `method` attribute removed prior to PUT

Parameters **`kwargs`** – keys and associated values to alter on the device

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

nat BIG-IP® Local Traffic Manager (LTM) Nat module.

REST URI `http://localhost/mgmt/tm/ltm/nat`

GUI Path `Local Traffic --> Nat`

REST Kind `tm:ltm:nat:*`

`Nats`(ltm)	BIG-IP® LTM Nat collection object
`Nat`(nat_s)	BIG-IP® LTM Nat collection resource

node Collections and Resources

class `f5.bigip.ltm.nat.`**`Nats`**(*ltm*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM Nat collection object

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.nat.`**`Nat`**(*nat_s*)

Bases: *`f5.bigip.resource.Resource`*, *`f5.bigip.mixins.ExclusiveAttributesMixin`*

BIG-IP® LTM Nat collection resource

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Note: If you are creating with ‘‘inheritedTrafficGroup‘ set to `False` you just also have a *trafficGroup*.

Parameters **`kwargs`** – All the key-values needed to create the resource

Returns `self` - A python object that represents the object’s configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError‘ exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains “name” and “partition”

NOTE: If kwargs has a ‘requests_params’ key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data[’uri’])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

node BIG-IP® Local Traffic Manager (LTM) node module.

REST URI `http://localhost/mgmt/tm/ltm/node`

GUI Path `Local Traffic --> Nodes`

REST Kind `tm:ltm:node:*`

Nodes(ltm)	BIG-IP® LTM node collection
Node(nodes)	BIG-IP® LTM node resource

node Collections and Resources
class `f5.bigip.ltm.node.`**Nodes**(*ltm*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® LTM node collection

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.node.`**`Node`**(*nodes*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM node resource

`update`(***kwargs*)

Call this to change the configuration of the service on the device.

This method uses HTTP PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- If `fqdn` is in the kwargs or set as an attribute, removes the `autopopulate` and `addressFamily` keys from it if there.

Parameters `kwargs` – keys and associated values to alter on the device

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters `kwargs` – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters `kwargs` – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

policy BIG-IP® Local Traffic Manager (LTM) policy module.

REST URI `http://localhost/mgmt/tm/ltm/policy`

GUI Path `Local Traffic --> policy`

REST Kind `tm:ltm:policy:*`

Policys(ltm)	BIG-IP® LTM policy collection.
Policy(policy_s)	BIG-IP® LTM policy resource.
Rules_s(policy)	BIG-IP® LTM policy rules sub-collection.
Rules(rules_s)	BIG-IP® LTM policy rules sub-collection resource.
Actions_s(rules)	BIG-IP® LTM policy actions sub-collection.
Actions(actions_s)	BIG-IP® LTM policy actions sub-collection resource.
Conditions_s(rules)	BIG-IP® LTM policy conditions sub-collection.
Conditions(conditions_s)	BIG-IP® LTM policy conditions sub-collection resource.

Policy Collections and Resources

class `f5.bigip.ltm.policy.`**`Policys`**`(`*ltm*`)`

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM policy collection.

`create``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection``(`***kwargs*`)`

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh``(`***kwargs*`)`

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.policy.`**`Policy`**`(`*policy_s*`)`

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM policy resource.

`create``(`***kwargs*`)`

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class f5.bigip.ltm.policy.**Rules_s**(*policy*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® LTM policy rules sub-collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.policy.`**`Rules`**(*rules_s*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM policy rules sub-collection resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.ltm.policy.`**`Actions_s`**(*rules*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM policy actions sub-collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.policy.`**`Actions`**(*actions_s*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM policy actions sub-collection resource.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.ltm.policy.`**`Conditions_s`**(*rules*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM policy conditions sub-collection.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.policy.`**`Conditions`**(*conditions_s*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM policy conditions sub-collection resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

pool BIG-IP® Local Traffic Manager™ (LTM®) pool module.

REST URI `http://localhost/mgmt/tm/ltm/pool`

GUI Path `Local Traffic --> Pools`

REST Kind `tm:ltm:pools:*`

`Pools`(ltm)	BIG-IP® LTM pool collection
`Pool`(pool_s)	BIG-IP® LTM pool resource
`Members_s`(pool)	BIG-IP® LTM pool members sub-collection
`Member`	

Pool Collections and Resources

class `f5.bigip.ltm.pool.`**`Pools`**`(`*ltm*`)`

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM pool collection

`create``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection``(`***kwargs*`)`

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh``(`***kwargs*`)`

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update``(`***kwargs*`)`

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.pool.`**`Pool`**`(`*pool_s*`)`

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM pool resource

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class f5.bigip.ltm.pool.**Members_s**(*pool*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® LTM pool members sub-collection

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all

CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.pool.`**Members**(*members_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® LTM pool members sub-collection resource

update(***kwargs*)
Call this to change the configuration of the service on the device.

This method uses HTTP PUT alter the service state on the device.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device. Various edge cases are handled:

- read-only attributes that are unchangeable are removed
- If `fqdn` is in the kwargs or set as an attribute, removes the `autopopulate` and `addressFamily` keys from it if there.

Parameters

- **state=** – state value or `None` required.
- **kwargs** – keys and associated values to alter on the device

exists(***kwargs*)
Check for the existence of the named object on the BigIP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it must then check the contents of the json contained in the response, this is because the "pool/... /members" resource provided by the server returns a status code of 200 for queries that do not correspond to an existing configuration. Therefore this method checks for the presence of the "address" key in the response JSON... of course, this means that exists depends on an unexpected idiosyncrancy of the server, and might break with version updates, edge cases, or other unpredictable changes.

Parameters kwargs – Keyword arguments required to get objects, "partition"

and "name" are required

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BigIP or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

rule BIG-IP® Local Traffic Manager (LTM) rule module.

REST URI `http://localhost/mgmt/tm/ltm/rule`

GUI Path `Local Traffic --> Rules`

REST Kind `tm:ltm:rule:*`

Rules(ltm)	BIG-IP® LTM rule collection
Rule(rule_s)	BIG-IP® LTM rule resource

Rule Collections and Resources

class `f5.bigip.ltm.rule.`**Rules**(*ltm*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® LTM rule collection

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.ltm.rule.**Rule**(*rule_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® LTM rule resource

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

snat BIG-IP® Local Traffic Manager (LTM) Snat module.

REST URI `http://localhost/mgmt/tm/ltm/snat`

GUI Path `Local Traffic --> Snat`

REST Kind `tm:ltm:snat:*`

`Snats`(ltm)	BIG-IP® LTM Snat collection
`Snat`(snat_s)	BIG-IP® LTM Snat resource

Snat Collections and Resources

class `f5.bigip.ltm.snat.`**`Snats`**(*ltm*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM Snat collection

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.snat.`**`Snat`**(*snat_s*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® LTM Snat resource

create(***kwargs*)

Call this to create a new snat on the BIG-IP®.

Uses HTTP POST to 'containing' URI to create a service associated with a new URI on the device.

Note this is the one of two fundamental Resource operations that returns a different uri (in the returned object) than the uri the operation was called on. The returned uri can be accessed as Object.selfLink, the actual uri used by REST operations on the object is Object._meta_data['uri']. The _meta_data['uri'] is the same as Object.selfLink with the substring 'localhost' replaced with the value of Object._meta_data['BIG-IP']._meta_data['hostname'], and without query args, or hash fragments.

The following is done prior to the POST * Ensures that one of `automap`, `snatpool`, `translastion`

parameter is passed in.

Parameters kwargs – All the key-values needed to create the resource

Returns An instance of the Python object that represents the device's

uri-published resource. The uri of the resource is part of the object's _meta_data.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

snatpool BIG-IP Local Traffic Manager (LTM) SNAT pool module.

REST URI `https://localhost/mgmt/tm/ltm/snatpool?ver=11.6.0`

GUI Path `Local Traffic --> Address Translation --> SNAT Pool List`

REST Kind `tm:ltm:snatpool:*`

Snatpools(ltm)	BIG-IP SNAT pool collection.
Snatpool(Snatpools)	BIG-IP SNAT Pool resource.

Snat Collections and Resources

class f5.bigip.ltm.snatpool.**Snatpools**(*ltm*)

Bases: *f5.bigip.resource.Collection*

BIG-IP SNAT pool collection.

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.snatpool.`**`Snatpool`**(*Snatpools*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP SNAT Pool resource.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

snat_translation BIG-IP Local Traffic Manager (LTM) SNAT Translation module.

REST URI `https://localhost/mgmt/tm/ltm/snat-translation?ver=11.6.0`

GUI Path `Local Traffic --> Address Translation --> Address Translation List`

REST Kind `tm:ltm:snat-translation:*`

Snat_Translations(ltm)	BIG-IP SNAT Translation collection.
Snat_Translation(Snat_Translations)	BIG-IP SNAT Translation resource.

Snat Collections and Resources

class `f5.bigip.ltm.snat_translation.`**`Snat_Translations`**(*ltm*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP SNAT Translation collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all

CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.ltm.snat_translation.**Snat_Translation**(*Snat_Translations*)
Bases: *f5.bigip.mixins.ExclusiveAttributesMixin*, *f5.bigip.resource.Resource*

BIG-IP SNAT Translation resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters kwargs – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters kwargs – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters kwargs – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

ssl

virtual BIG-IP® Local Traffic Manager (LTM) virtual module.

REST URI `http://localhost/mgmt/tm/ltm/virtual`

GUI Path `Local Traffic --> Virtual Servers`

REST Kind `tm:ltm:virtual:*`

`Virtuals`(ltm)	BIG-IP® LTM virtual collection
`Virtual`(virtual_s)	BIG-IP® LTM virtual resource

Virtual Collections and Resources

class `f5.bigip.ltm.virtual.`**`Virtuals`**(*ltm*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® LTM virtual collection

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.ltm.virtual.**Virtual**(*virtual_s*)

Bases: *f5.bigip.resource.Resource*

BIG-IP® LTM virtual resource

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.ltm.virtual.`**`Profiles`**(*Profiles_s*)

Bases: *`f5.bigip.resource.Resource`*

A Resource concrete subclass.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class f5.bigip.ltm.virtual.**Profiles_s**(*virtual*)
Bases: *f5.bigip.resource.Collection*

A Collection concrete subclass docstring.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh` (***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update` (***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

virtual_address Directory: ltm module: virtual-address.

REST URI `https://localhost/mgmt/tm/ltm/virtual-address?ver=11.6.0`

GUI Path `Local Traffic Manager --> Virtual Servers --> Virtual Address List`

REST Kind `tm:ltm:virtual-address:*`

Virtual_Address_s(ltm)	BIG-IP LTM virtual address collection.
Virtual_Address(Virtual_Address_s)	BIG-IP LTM virtual address resource.

Virtual Address Collections and Resources

class `f5.bigip.ltm.virtual_address.`**`Virtual_Address_s`**(*ltm*)

Bases: *f5.bigip.resource.Collection*

BIG-IP LTM virtual address collection.

`create` (***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete` (***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection` (***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.ltm.virtual_address.`**Virtual_Address**(*Virtual_Address_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP LTM virtual address resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

f5.bigip.net

Module Conents BIG-IP® net module

REST URI `http://localhost/mgmt/tm/net/`

GUI Path `Network`

REST Kind `tm:net:*`

arp	BIG-IP® Network ARP module.
fdb	Directory: net module: fdb.
interface	BIG-IP® Network interface module.
route	BIG-IP® Network route module.
route_domain	Directory: net module: route-domain.
selfip	BIG-IP® Network self-ip module.

Continued on next page

Table 3.18 – continued from previous page

`tunnels`	BIG-IP® Network tunnels module.
`vlan`	BIG-IP® Network vlan module.

Submodule List

Submodules

arp BIG-IP® Network ARP module.

REST URI `http://localhost/mgmt/tm/net/arp`

GUI Path `Network --> ARP`

REST Kind `tm:net:arp:*`

Arps(net)	BIG-IP® network ARP collection
Arp(arp_s)	BIG-IP® network ARP resource

ARP Collections and Resources

class `f5.bigip.net.arp.`**`Arps`**(*net*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® network ARP collection

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.arp.`**`Arp`**(*arp_s*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® network ARP resource

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters `kwargs` – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters `kwargs` – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters `kwargs` – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

interface BIG-IP® Network interface module.

REST URI `http://localhost/mgmt/tm/net/interface`

GUI Path `Network --> Interfaces`

REST Kind `tm:net:interface:*`

`Interfaces`(net)	BIG-IP® network interface collection
`Interface`(interface_s)	BIG-IP® network interface collection

Interface Collections and Resources

class `f5.bigip.net.interface.`**`Interfaces`**(*net*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® network interface collection

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.interface.`**Interface**(*interface_s*)

Bases: *f5.bigip.resource.Resource*, *f5.bigip.mixins.ExclusiveAttributesMixin*

BIG-IP® network interface collection

create(***kwargs*)

Create is not supported for interfaces.

Raises `UnsupportedOperation`

delete()

Delete is not supported for interfaces.

Raises `UnsupportedOperation`

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

route BIG-IP® Network route module.

REST URI `http://localhost/mgmt/tm/net/route`

GUI Path `Network --> Routes`

REST Kind `tm:net:route:*`

`Routes`(net)	BIG-IP® network route collection
`Route`(route_s)	BIG-IP® network route resource

Route Collections and Resources

class `f5.bigip.net.route.`**`Routes`**(*net*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® network route collection

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.net.route.**Route**(*route_s*)
Bases: *f5.bigip.resource.Resource*, *f5.bigip.mixins.ExclusiveAttributesMixin*

BIG-IP® network route resource

create(***kwargs*)
Create a Route on the BIG-IP® and the associated python object.

One of the following gateways is required when creating the route objects: `blackhole`, `gw`, `tmInterface`, `pool`.

Params kwargs keyword arguments passed in from create call

Raises KindTypeMismatch

Raises MissingRequiredCreationParameter

Raises HTTPError

Returns Python Route object

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters `kwargs` – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

route_domain Directory: net module: route-domain.

REST URI `https://localhost/mgmt/tm/net/route-domain?ver=11.6.0`

GUI Path `XXX`

REST Kind `tm:net:route-domain:*`

Route_Domains(net)	A Collection concrete subclass docstring.
Route_Domain(Route_Domains)	A Resource concrete subclass.

Route Collections and Resources

class `f5.bigip.net.route_domain.`**`Route_Domains`**(*net*)

Bases: *f5.bigip.resource.Collection*

A Collection concrete subclass docstring.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.route_domain.`**`Route_Domain`**(*Route_Domains*)

Bases: *`f5.bigip.resource.Resource`*

A Resource concrete subclass.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

selfip BIG-IP® Network self-ip module.

Note: Self IPs path does not match their kind or URI because the string `self` causes problems in Python because it is a reserved word.

REST URI `http://localhost/mgmt/tm/net/self`

GUI Path `Network --> Self IPs`

REST Kind `tm:net:self:*`

Selfips(net)	BIG-IP® network Self-IP collection
Selfip(selfip_s)	BIG-IP® Self-IP resource

Selfip Collections and Resources

class f5.bigip.net.selfip.**Selfips**(*net*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® network Self-IP collection

Note: The objects in the collection are actually called 'self' in iControlREST, but obviously this will cause problems in Python so we changed its name to Selfip.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.selfip.`**Selfip**(*selfip_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® Self-IP resource

Use this object to create, refresh, update, delete, and load self ip configuration on the BIG-IP®. This requires that a `VLAN` object be present on the system and that object's **:attrib:'fullPath'** be used as the VLAN name.

The address that is used for create is a <*ipaddress*>/<*netmask*>. For example `192.168.1.1/32`.

Note: The object is actually called `self` in iControlREST, but obviously this will cause problems in Python so we changed its name to `Selfip`.

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters `kwargs` – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters `kwargs` – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters `kwargs` – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters `kwargs` – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

tunnels BIG-IP® Network tunnels module.

REST URI `http://localhost/mgmt/tm/net/tunnels`

GUI Path `Network --> tunnels`

REST Kind `tm:net:tunnels:*`

`Tunnels_s`(net)	BIG-IP® network tunnels collection
`Tunnels`(tunnels_s)	BIG-IP® network tunnels resource (collection for GRE, Tunnel, VXLANs
`Tunnel`(tunnels)	BIG-IP® tunnels tunnel resource
`Gres`(tunnels_s)	BIG-IP® tunnels GRE sub-collection
`Gre`(gres)	BIG-IP® tunnels GRE sub-collection resource
`Vxlans`(tunnels_s)	BIG-IP® tunnels VXLAN sub-collection
`Vxlan`(vxlans)	BIG-IP® tunnels VXLAN sub-collection resource

Tunnels Collections and Resources

class `f5.bigip.net.tunnels.`**`Tunnels_s`**(*net*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® network tunnels collection

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.tunnels.`**Tunnels**(*tunnels_s*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® network tunnels resource (collection for GRE, Tunnel, VXLANs

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.tunnels.`**`Tunnel`**(*tunnels*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® tunnels tunnel resource

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.net.tunnels.`**`Gres`**(*tunnels_s*)
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® tunnels GRE sub-collection

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.tunnels.`**`Gre`**(*gres*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® tunnels GRE sub-collection resource

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.net.tunnels.`**Vxlans**(*tunnels_s*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® tunnels VXLAN sub-collection

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.tunnels.`**`Vxlan`**(*vxlans*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® tunnels VXLAN sub-collection resource

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

vlan BIG-IP® Network vlan module.

REST URI `http://localhost/mgmt/tm/net/vlan`

GUI Path `Network --> Vlans`

REST Kind `tm:net:vlan:*`

`Vlans`(net)	BIG-IP® network Vlan collection.
`Vlan`(vlan_s)	BIG-IP® network Vlan resource.
`Interfaces_s`(vlan)	BIG-IP® network Vlan interface collection.
`Interfaces`(interfaces_s)	BIG-IP® network Vlan interface resource.

Vlan Collections and Resources

class `f5.bigip.net.vlan.`**`Vlans`**(*net*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® network Vlan collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.vlan.`**`Vlan`**(*vlan_s*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® network Vlan resource.

`create`(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh``(`***kwargs*`)`
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update``(`***kwargs*`)`
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.net.vlan.`**`Interfaces_s`**`(`*vlan*`)`
Bases: *`f5.bigip.resource.Collection`*

BIG-IP® network Vlan interface collection.

Note: Not to be confused with `tm/mgmt/net/interface`. This is object is actually called `interfaces` with an `s` by the BIG-IP's REST API.

`create``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete``(`***kwargs*`)`
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection``(`***kwargs*`)`
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.vlan.`**`Interfaces`**(*interfaces_s*)
Bases: *`f5.bigip.resource.Resource`*, *`f5.bigip.mixins.ExclusiveAttributesMixin`*

BIG-IP® network Vlan interface resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

fdb Directory: net module: fdb.

REST URI `https://localhost/mgmt/tm/net/fdb`

GUI Path `XXX`

REST Kind `tm:net:fdb:*`

Fdbs(net)	A Collection concrete subclass docstring.
Tunnel(Tunnels)	A Resource concrete subclass.
Tunnels(fdb)	A Collection concrete subclass docstring.
Vlans(fdb)	A Collection concrete subclass docstring.

FDB Collections and Resources

class `f5.bigip.net.fdb.`**`Fdbs`**(*net*)

Bases: *`f5.bigip.resource.Collection`*

A Collection concrete subclass docstring.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource‘s that map to the most recently ‘refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a “requests_params” dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.fdb.`**`Tunnel`**(*Tunnels*)

Bases: *`f5.bigip.resource.Resource`*

A Resource concrete subclass.

create(***kwargs*)

Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.net.fdb.`**`Tunnels`**(*fdb*)
Bases: *`f5.bigip.resource.Collection`*

A Collection concrete subclass docstring.

`create`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all

CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.net.fdb.`**`Vlans`**(*fdb*)
Bases: *`f5.bigip.resource.Collection`*

A Collection concrete subclass docstring.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

f5.bigip.shared

Module Contents BIG-IP® Shared (shared) module

REST URI `http://localhost/mgmt/tm/shared/`

GUI Path `System`

REST Kind N/A – HTTP GET returns an error

bigip_failover_state	BIG-IP® shared failover state module
licensing	BIG-IP® system failover module

Submodule List

Submodules

bigip_failover_state BIG-IP® shared failover state module

REST URI `http://localhost/mgmt/tm/shared/bigip-failover-state`

GUI Path N/A

REST Kind `tm:shared:licensing:*`

class `f5.bigip.shared.bigip_failover_state.`**`Bigip_Failover_State`**(*shared*)

Bases: *f5.bigip.mixins.UnnamedResourceMixin*, *f5.bigip.resource.Resource*

BIG-IP® failover state information

Failover state objects only support the *load()* method because they cannot be modified via the API.

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

update(***kwargs*)

Update is not supported for BIG-IP® failover state.

Raises UnsupportedOperation

create(***kwargs*)

Create is not supported for unnamed resources

Raises UnsupportedOperation

delete(***kwargs*)

Delete is not supported for unnamed resources

Raises UnsupportedOperation

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError‘ exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

licensing BIG-IP® system failover module

REST URI `http://localhost/mgmt/tm/shared/license`

GUI Path `System --> License`

REST Kind `tm:shared:licensing:*`

class `f5.bigip.shared.licensing.`**`Licensing`**(*shared*)
Bases: *`f5.bigip.resource.PathElement`*

BIG-IP® licensing stats and states.

Licensing objects themselves do not support any methods and are just containers for lower level objects.

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

class `f5.bigip.shared.licensing.`**`Activation`**(*licensing*)
Bases: *`f5.bigip.mixins.UnnamedResourceMixin`*, *`f5.bigip.resource.Resource`*

BIG-IP® license activation status

Activation state objects only support the *`load()`* method because they cannot be modified via the API.

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

`update`(***kwargs*)
Update is not supported for License Activation

Raises UnsupportedOperation

`create`(***kwargs*)
Create is not supported for unnamed resources

Raises UnsupportedOperation

`delete`(***kwargs*)
Delete is not supported for unnamed resources

Raises UnsupportedOperation

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

class `f5.bigip.shared.licensing.`**`Registration`**(*licensing*)
Bases: *`f5.bigip.mixins.UnnamedResourceMixin`*, *`f5.bigip.resource.Resource`*

BIG-IP® license registration status

Registration state objects only support the *`load()`* method because they cannot be modified via the API.

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

`update`(***kwargs*)
Update is not supported for License Registration

Raises UnsupportedOperation

`create`(***kwargs*)
Create is not supported for unnamed resources

Raises UnsupportedOperation

`delete`(***kwargs*)
Delete is not supported for unnamed resources

Raises UnsupportedOperation

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

f5.bigip.sys

Module Contents BIG-IP® System (sys) module

REST URI `http://localhost/mgmt/tm/sys/`

GUI Path `System`

REST Kind `tm:sys:*`

application	BIG-IP® iApp (application) module
db	BIG-IP® db module
failover	BIG-IP® system failover module
folder	BIG-IP® system folder (partition) module
global_settings	BIG-IP® system global-settings module
ntp	BIG-IP® system ntp module
performance	BIG-IP® system peformance stats module.

Submodule List

Submodules

application BIG-IP® iApp (application) module

REST URI `http://localhost/mgmt/sys/application/`

GUI Path `iApps`

REST Kind `tm:sys:application:*`

`Applications`(sys)	BIG-IP® iApp collection.
`Aplscripts`(application)	BIG-IP® iApp script collection.
`Aplscript`(apl_script_s)	BIG-IP® iApp script resource.
`Customstats`(application)	BIG-IP® iApp custom stats sub-collection.
`Customstat`(custom_stat_s)	BIG-IP® iApp custom stats sub-collection resource.
`Services`(application)	BIG-IP® iApp service sub-collection.
`Service`(service_s)	BIG-IP® iApp service sub-collection resource
`Templates`(application)	BIG-IP® iApp template sub-collection
`Template`(template_s)	BIG-IP® iApp template sub-collection resource

Application Collections and Resources

class `f5.bigip.sys.application.`**`Applications`**(*sys*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® iApp collection.

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all

CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.sys.application.**Aplscripts**(*application*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® iApp script collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.sys.application.**Aplscript**(*apl_script_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® iApp script resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the

device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class `f5.bigip.sys.application.`**Customstats**(*application*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® iApp custom stats sub-collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.application.`**Customstat**(*custom_stat_s*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® iApp custom stats sub-collection resource.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **kwargs** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class f5.bigip.sys.application.**Services**(*application*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® iApp service sub-collection.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python Resource objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python Resource objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.application.`**Service**(*service_s*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® iApp service sub-collection resource

update(***kwargs*)
Push local updates to the object on the device.

Params kwargs keyword arguments for accessing/modifying the object

Returns updated Python object

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Override of resource.Resource exists() to build proper URI unique to service resources.

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **kwargs** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

class `f5.bigip.sys.application.`**Templates**(*application*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® iApp template sub-collection

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.application.`**`Template`**(*template_s*)
Bases: *`f5.bigip.resource.Resource`*

BIG-IP® iApp template sub-collection resource

`create`(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`delete`(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)
Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

db BIG-IP® db module

REST URI `http://localhost/mgmt/sys/db/`

GUI Path N/A

REST Kind `tm:sys:db:*`

Dbs(sys)	BIG-IP® db collection
Db(dbs)	BIG-IP® db resource

DB Collections and Resources

class `f5.bigip.sys.db.`**`Dbs`**(*sys*)

Bases: *`f5.bigip.resource.Collection`*

BIG-IP® db collection

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`get_collection`(***kwargs*)

Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.db.`**`Db`**(*dbs*)

Bases: *`f5.bigip.resource.Resource`*

BIG-IP® db resource

Note: db objects are read-only.

`create`(***kwargs*)

Create is not supported for db resources.

Raises UnsupportedOperation

delete(***kwargs*)

Delete is not supported for db resources.

Raises UnsupportedOperation

exists(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

load(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **kwargs** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

failover BIG-IP® system failover module

REST URI `http://localhost/mgmt/tm/sys/failover`

GUI Path `System --> Failover`

REST Kind `tm:sys:failover:*`

`Failover`(sys)	BIG-IP® Failover stats and state change.

Failover Resources

class `f5.bigip.sys.failover.`**`Failover`**(*sys*)

Bases: *`f5.bigip.mixins.UnnamedResourceMixin`*, *`f5.bigip.resource.Resource`*

BIG-IP® Failover stats and state change.

The failover object only supports load, update, and refresh because it is an unnamed resource.

To force the unit to standby call the `update()` method as follows:

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

`update`(***kwargs*)

Update is not supported for Failover

Raises UnsupportedOperation

`toggle_standby`(***kwargs*)

Toggle the standby status of a traffic group.

WARNING: This method which used POST obtains json keys from the device that are not available in the response to a GET against the same URI.

Unique to refresh/GET: u"apiRawValues" u"selfLink" Unique to toggle_standby/POST: u"command" u"standby" u"traffic-group"

`create`(***kwargs*)

Create is not supported for unnamed resources

Raises UnsupportedOperation

`delete`(***kwargs*)

Delete is not supported for unnamed resources

Raises UnsupportedOperation

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters `kwargs` – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS

HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

folder BIG-IP® system folder (partition) module

REST URI `http://localhost/mgmt/tm/sys/folder`

GUI Path `System --> Users --> Partition List`

REST Kind `tm:sys:folder:*`

Folders(sys)	BIG-IP® system folder collection.
Folder(folder_s)	

Folder Collections and Resources

class f5.bigip.sys.folder.**Folders**(*sys*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® system folder collection.

These are what we refer to as `partition` in the SDK.

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

global_settings BIG-IP® system global-settings module

REST URI `http://localhost/mgmt/tm/sys/global-settings`

GUI Path `System --> Configuration --> Device`

REST Kind `tm:sys:global-settings:*`

Global_Settings(sys)	BIG-IP® system global-settings resource

Global_Settings Resources

class `f5.bigip.sys.global_settings.`**`Global_Settings`**(*sys*)
Bases: *f5.bigip.mixins.UnnamedResourceMixin*, *f5.bigip.resource.Resource*

BIG-IP® system global-settings resource

The global_settings object only supports load and update because it is an unnamed resource.

Note: This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

`create`(***kwargs*)
Create is not supported for unnamed resources

Raises UnsupportedOperation

`delete`(***kwargs*)
Delete is not supported for unnamed resources

Raises UnsupportedOperation

`exists`(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`raw`
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

ntp BIG-IP® system ntp module

REST URI `http://localhost/mgmt/tm/sys/ntp`

GUI Path `System --> Configuration --> Device --> NTP`

REST Kind `tm:sys:ntp:*`

`Ntp`(sys)	BIG-IP® system NTP unnamed resource
`Restricts`(ntp)	BIG-IP® system NTP restrict sub-collection
`Restrict`(restricts)	BIG-IP® system NTP restrict sub-collection resource

NTP Resources and Subcollections

class `f5.bigip.sys.ntp.`**`Ntp`**(*sys*)
Bases: *`f5.bigip.mixins.UnnamedResourceMixin`*, *`f5.bigip.resource.Resource`*

BIG-IP® system NTP unnamed resource

This is an unnamed resource so it has not ~Partition~Name pattern at the end of its URI.

create(***kwargs*)
Create is not supported for unnamed resources

Raises UnsupportedOperation

delete(***kwargs*)
Delete is not supported for unnamed resources

Raises UnsupportedOperation

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **kwargs** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

class f5.bigip.sys.ntp.**Restricts**(*ntp*)
Bases: *f5.bigip.resource.Collection*

BIG-IP® system NTP restrict sub-collection

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.ntp.`**Restrict**(*restricts*)
Bases: *f5.bigip.resource.Resource*

BIG-IP® system NTP restrict sub-collection resource

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters kwargs – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

delete(***kwargs*)
Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

performance BIG-IP® system peformance stats module.

REST URI `http://localhost/mgmt/tm/sys/performance`

GUI Path `System --> Users --> Partition List`

REST Kind `tm:sys:performance:*`

Performance(sys)	BIG-IP® system performace stats collection
All_Stats(performance)	BIG-IP® system performace stats unnamed resource

Performance Resources and Subcollections

class `f5.bigip.sys.performance.`**`Performance`**(*sys*)

Bases: *f5.bigip.resource.Collection*

BIG-IP® system performace stats collection

`get_collection`()

Performance collections are not proper BIG-IP® collection objects.

Raises `UnsupportedOperation`

`create`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`delete`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

`refresh`(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

`update`(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.sys.performance.`**`All_Stats`**(*performance*)

Bases: *f5.bigip.mixins.UnnamedResourceMixin*, *f5.bigip.resource.Resource*

BIG-IP® system performace stats unnamed resource

update(***kwargs*)
Update is not supported for statistics.

Raises `UnsupportedOperation`

create(***kwargs*)
Create is not supported for unnamed resources

Raises UnsupportedOperation

delete(***kwargs*)
Delete is not supported for unnamed resources

Raises UnsupportedOperation

exists(***kwargs*)
Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **kwargs** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

resource module

This module provides classes that specify how RESTful resources are handled.

THE MOST IMPORTANT THING TO KNOW ABOUT THIS API IS THAT YOU CAN DIRECTLY INFER REST URIs FROM PYTHON EXPRESSIONS, AND VICE VERSA.

Examples:

- Expression: bigip = BigIP('a', 'b', 'c')
- URI Returned: https://a/mgmt/tm/
- Expression: bigip.ltm
- URI Returned: https://a/mgmt/tm/ltm/

- Expression: pools1 = bigip.ltm.pools
- URI Returned: https://a/mgmt/tm/ltm/pool
- Expression: pool_a = pools1.create(partition="Common", name="foo")
- URI Returned: https://a/mgmt/tm/ltm/pool/~Common~foo

There are different types of resources published by the BIG-IP® REST Server, they are represented by the classes in this module.

We refer to a server-provided resource as a "service". Thus far all URI referenced resources are "services" in this sense.

We use methods named Create, Refresh, Update, Load, and Delete to manipulate BIG-IP® device services.

Methods:

- create – uses HTTP POST, creates a new resource and with its own URI on the device
- refresh – uses HTTP GET, obtains the state of a device resource, and sets the representing Python Resource Object tracks device state via its attrs
- **update – uses HTTP PUT, submits a new configuration to the device resource** and sets the Resource attrs to the state the device reports
- load – uses HTTP GET, obtains the state of an existing resource on the device and sets the Resource attrs to that state
- delete – uses HTTP DELETE, removes the resource from the device, and sets self.__dict__ to {'deleted': True}

Available Classes:

- ResourceBase – only *refresh* is generally supported in all resource types, this class provides *refresh*. ResourceBase objects are usually instantiated via setting lazy attributes. ResourceBase provides a constructor to match its call in LazyAttributeMixin.__getattr__. The expected behavior is that all resource subclasses depend on this constructor to correctly set their self._meta_data['uri']. All ResourceBase objects (except BIG-IPs) have a container (BIG-IPs contain themselves). The container is the object the ResourceBase is an attribute of.
- OrganizingCollection – These resources support lists of "reference" "links". These are json blobs without a Python class representation.

  Example URI_path: /mgmt/tm/ltm/
- **Collection – These resources support lists of ResourceBase Objects.** Example URI_path: /mgmt/tm/ltm/nat
- Resource – These resources are the only resources that support *create*, *update*, and *delete* operations. Because they support HTTP post (via _create) they uniquely depend on 2 uri's, a uri that supports the creating post, and the returned uri of the newly created resource.

  Example URI_path: /mgmt/tm/ltm/nat/~Common~testnat1

exception `f5.bigip.resource.`**`KindTypeMismatch`**

Bases: *`f5.sdk_exception.F5SDKError`*

Raise this when server JSON keys are incorrect for the Resource type.

exception `f5.bigip.resource.`**`DeviceProvidesIncompatibleKey`**

Bases: *`f5.sdk_exception.F5SDKError`*

Raise this when server JSON keys are incompatible with Python.

exception f5.bigip.resource.**InvalidResource**
Bases: *f5.sdk_exception.F5SDKError*

Raise this when a caller tries to invoke an unsupported CRUDL op.

All resources support *refresh* and *raw*. Only *Resource*'s support *load*, *create*, *update*, and *delete*.

exception f5.bigip.resource.**MissingRequiredCreationParameter**
Bases: *f5.sdk_exception.F5SDKError*

Various values MUST be provided to create different Resources.

exception f5.bigip.resource.**MissingRequiredReadParameter**
Bases: *f5.sdk_exception.F5SDKError*

Various values MUST be provided to refresh some Resources.

exception f5.bigip.resource.**UnregisteredKind**
Bases: *f5.sdk_exception.F5SDKError*

The returned server JSON *kind* key wasn't expected by this Resource.

exception f5.bigip.resource.**GenerationMismatch**
Bases: *f5.sdk_exception.F5SDKError*

The server reported BIG-IP® is not the expacted value.

exception f5.bigip.resource.**InvalidForceType**
Bases: exceptions.ValueError

Must be of type bool.

exception f5.bigip.resource.**URICreationCollision**
Bases: *f5.sdk_exception.F5SDKError*

self._meta_data['uri'] can only be assigned once. In create or load.

exception f5.bigip.resource.**UnsupportedOperation**
Bases: *f5.sdk_exception.F5SDKError*

Object does not support the method that was called.

class f5.bigip.resource.**PathElement**(*container*)
Bases: *f5.bigip.mixins.LazyAttributeMixin*

Base class to represent a URI path element that does not contain data.

The BIG-IP® iControl REST API has URIs that are made up of path components that do not return data when they are queried. This class represents those elements and does not support any of the CURDLE methods that the other objects do.

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

class f5.bigip.resource.**ResourceBase**(*container*)
Bases: *f5.bigip.resource.PathElement*, *f5.bigip.mixins.ToDictMixin*

Base class for all BIG-IP® iControl REST API endpoints.

The BIG-IP® is represented by an object that converts device-published uri's into Python objects. Each uri maps to a Python object. The mechanism for instantiating these objects is the __getattr__ Special Function in the LazyAttributeMixin. When a registered attribute is *dot* referenced, on the device object (e.g. `bigip.ltm` or simply `bigip`), an appropriate object is instantiated and attributed to the referencing object:

```
    bigip.ltm = LTM(bigip)
    bigip.ltm.nats
    nat1 = bigip.ltm.nats.nat.create('Foo', 'Bar', '0.1.2.3', '1.2.3.4')
```

This can be shortened to just the last line:

```
    nat1 = bigip.ltm.nats.nat.create('Foo', 'Bar', '0.1.2.3', '1.2.3.4')
```

Critically this enforces a convention relating device published uris to API objects, in a hierarchy similar to the uri paths. I.E. the uri corresponding to a `Nats` object is `mgmt/tm/ltm/nat/`. If you query the BIG-IP's uri (e.g. print(bigip._meta_data['uri'])), you'll see that it ends in: `/mgmt/tm/`, if you query the `ltm` object's uri (e.g. print(bigip.ltm._meta_data['uri'])) you'll see it ends in `/mgmt/tm/ltm/`.

In general the objects build a required *self._meta_data['uri']* attribute by: 1. Inheriting this class. 2. calling super(Subclass, self).__init__(container) 3. self.uri = self.container_uri['uri'] + '/' + self.__class__.__name__

The net result is a succinct mapping between uri's and objects, that represents objects in a hierarchical relationship similar to the device's uri path hierarchy.

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

create(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

update(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)

Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

raw

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

class f5.bigip.resource.**OrganizingCollection**(*bigip*)

Bases: *f5.bigip.resource.ResourceBase*

Base class for objects that collect resources under them.

`OrganizingCollection` objects fulfill the following functions:

- represent a uri path fragment immediately 'below' /mgmt/tm
- provide a list of dictionaries that contain uri's to other resources on the device.

get_collection(***kwargs*)

Call to obtain a list of the reference dicts in the instance *items*

Returns List of self.items

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class f5.bigip.resource.**Collection**(*container*)
Bases: *f5.bigip.resource.ResourceBase*

Base class for objects that collect a list of `Resources`

The Collection Resource is responsible for providing a list of Python objects, where each object represents a unique URI, the URI contains the URI of the Collection at the front of its path, and the 'kind' of the URI-associated-JSON has been registered with the attribute registry of the Collection subclass.

Note: Any subclass of this base class must have `s` at the end of its name unless it ends in `s` then it must have `_s`.

get_collection(***kwargs*)
Get an iterator of Python `Resource` objects that represent URIs.

The returned objects are Pythonic *Resource's that map to the most recently 'refreshed* state of uris-resources published by the device. In order to instantiate the correct types, the concrete subclass must populate its registry with acceptable types, based on the *kind* field returned by the REST server.

Note: This method implies a single REST transaction with the Collection subclass URI.

Raises UnregisteredKind

Returns list of reference dicts and Python `Resource` objects

create(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

delete(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

raw
Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)
Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

update(***kwargs*)
Implement this by overriding it in a subclass of *Resource*

Raises InvalidResource

class `f5.bigip.resource.`**Resource**(*container*)
Bases: *`f5.bigip.resource.ResourceBase`*

Base class to represent a Configurable Resource on the device.

Warning: Objects instantiated from subclasses of Resource do NOT contain a URI (self._meta_data['uri']) at instantiation!

Resource objects provide the interface for the Creation of new services on the device. Once a new service has been created, (via `self.create` or `self.load`), the instance constructs its URI and stores it as `self._meta_data['uri']`.

It is an error to attempt to call *`create()`* or *`load()`* on an instance more than once. `self._meta_data['uri']` MUST not be changed after creation or load.

Note: creation query args, and creation hash fragments are stored as separate _meta_data values.

By "Configurable" we mean that submitting JSON via the PUT method to the URI managed by subclasses of Resource, changes the state of the corresponding service on the device.

It also means that the URI supports *DELETE*.

create(***kwargs*)
Create the resource on the BIG-IP®.

Uses HTTP POST to the *collection* URI to create a resource associated with a new unique URI on the device.

Parameters **`kwargs`** – All the key-values needed to create the resource

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.post method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: `self` - A python object that represents the object's

configuration and state on the BIG-IP®.

`load`(***kwargs*)

Load an already configured service into this instance.

This method uses HTTP GET to obtain a resource from the BIG-IP®.

Parameters **`kwargs`** – typically contains "name" and "partition"

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: a Resource Instance (with a populated _meta_data['uri'])

`update`(***kwargs*)

Update the configuration of the resource on the BIG-IP®.

This method uses HTTP PUT alter the resource state on the BIG-IP®.

The attributes of the instance will be packaged as a dictionary. That dictionary will be updated with kwargs. It is then submitted as JSON to the device.

Various edge cases are handled: * read-only attributes that are unchangeable are removed

Parameters **`kwargs`** – keys and associated values to alter on the device

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.put method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`delete`(***kwargs*)

Delete the resource on the BIG-IP®.

Uses HTTP DELETE to delete the resource on the BIG-IP®.

After this method is called, and status_code 200 response is received `instance.__dict__` is replace with `{'deleted': True}`

Parameters **`kwargs`** – The only current use is to pass kwargs to the requests

API. If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.delete method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS!

`exists`(***kwargs*)

Check for the existence of the named object on the BIG-IP

Sends an HTTP GET to the URI of the named object and if it fails with a :exc:~requests.HTTPError' exception it checks the exception for status code of 404 and returns `False` in that case.

If the GET is successful it returns `True`.

For any other errors are raised as-is.

Parameters **`kwargs`** – Keyword arguments required to get objects

NOTE: If kwargs has a 'requests_params' key the corresponding dict will be passed to the underlying requests.session.get method where it will be handled according to that API. THIS IS HOW TO PASS QUERY-ARGS! :returns: bool – The objects exists on BIG-IP® or not. :raises: `requests.HTTPError`, Any HTTP error that was not status

code 404.

`raw`

Display the attributes that the current object has and their values.

Returns A dictionary of attributes and their values

refresh(***kwargs*)

Use this to make the device resource be represented by self.

This method makes an HTTP GET query against the device service. This method is run for its side-effects on self. If successful the instance attribute __dict__ is replaced with the dict representing the device state. To figure out what that state is, run a subsequest query of the object like this: As with all CURDLE methods use a "requests_params" dict to pass parameters to requests.session.HTTPMETHOD. See test_requests_params.py for an example. >>> resource_obj.refresh() >>> print(resource_obj.raw)

mixins module

class `f5.bigip.mixins.`**`ToDictMixin`**

Bases: `object`

Convert an object's attributes to a dictionary

exception `f5.bigip.mixins.`**`LazyAttributesRequired`**

Bases: *`f5.sdk_exception.F5SDKError`*

Raised when a object accesses a lazy attribute that is not listed

class `f5.bigip.mixins.`**`LazyAttributeMixin`**

Bases: `object`

Allow attributes to be created lazily based on the allowed values

class `f5.bigip.mixins.`**`ExclusiveAttributesMixin`**

Bases: `object`

Overrides __`setattr`__ to remove exclusive attrs from the object.

class `f5.bigip.mixins.`**`UnnamedResourceMixin`**

Bases: `object`

This makes a resource object work if there is no name.

These objects do not support create or delete and are often found as Resources that are under an organizing collection. For example the *mgmt/tm/sys/global-settings* is one of these and has a kind of *tm:sys:global-settings:global-settingsstate* and the URI does not match the kind.

create(***kwargs*)

Create is not supported for unnamed resources

Raises UnsupportedOperation

delete(***kwargs*)

Delete is not supported for unnamed resources

Raises UnsupportedOperation

f5.common

Subpackages

Submodules

f5.common.constants module

f5.common.iapp_parser module

class `f5.common.iapp_parser.`**`IappParser`**`(`*template_str*`)`
Bases: `object`

`template_sections` = **[u'presentation', u'implementation', u'html-help', u'role-acl']**

`tcl_list_for_attr_re` = **'{(\\s*\\w+\\s*)+}'**

`tcl_list_for_section_re` = **'(\\s*\\w+\\s*)+'**

`section_map` = **{u'html-help': u'htmlHelp', u'role-acl': u'roleAcl'}**

`attr_map` = **{u'requires-modules': u'requiresModules'}**

`sections_not_required` = **[u'html-help', u'role-acl']**

`tcl_list_patterns` = **{u'requires-modules': '{(\\s*\\w+\\s*)+}', u'role-acl': '(\\s*\\w+\\s*)+'}**

`template_attrs` = **[u'description', u'partition', u'requires-modules']**

`parse_template``()`
Parse the template string into a dict.

Find the (large) inner sections first, save them, and remove them from a modified string. Then find the template attributes in the modified string.

Returns dictionary of parsed template

exception `f5.common.iapp_parser.`**`EmptyTemplateException`**
Bases: *`f5.sdk_exception.F5SDKError`*

`args`

`message`

exception `f5.common.iapp_parser.`**`CurlyBraceMismatchException`**
Bases: *`f5.sdk_exception.F5SDKError`*

`args`

`message`

exception `f5.common.iapp_parser.`**`NonextantSectionException`**
Bases: *`f5.sdk_exception.F5SDKError`*

`args`

`message`

exception `f5.common.iapp_parser.`**`NonextantTemplateNameException`**
Bases: *`f5.sdk_exception.F5SDKError`*

`args`

`message`

exception `f5.common.iapp_parser.`**`MalformedTCLListException`**

Bases: *`f5.sdk_exception.F5SDKError`*

`args`

`message`

f5.common.logger module

Module contents

f5.sdk_exception

A base exception for all exceptions in this library.

Base Exception

`F5SDKError`	Import and subclass this exception in all exceptions in this library.

exception `f5.sdk_exception.`**`F5SDKError`**

Bases: `exceptions.Exception`

Import and subclass this exception in all exceptions in this library.

CHAPTER 4

Copyright

CHAPTER 5

License

5.1 Apache V2.0

5.2 Contributor License Agreement

Individuals or business entities who contribute to this project must have completed and submitted the F5 Contributor License Agreement to Openstack_CLA@f5.com prior to their code submission being included in this project.

Python Module Index