

---

# **F5 Python SDK Documentation**

***Release 0.1.3***

**F5 Networks**

April 01, 2016



<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Quick Start</b>	<b>3</b>
2.1	Installation . . . . .	3
2.2	Basic Example . . . . .	3
<b>3</b>	<b>Detailed Documentation</b>	<b>5</b>
3.1	User Guide . . . . .	5
3.1.1	Basic Concepts . . . . .	5
	REST URIs . . . . .	5
	REST Endpoints . . . . .	6
	Dynamic Attributes . . . . .	6
	iControl REST kind Parameters . . . . .	6
	Methods . . . . .	7
3.1.2	REST API Endpoints . . . . .	7
	Overview . . . . .	7
	Endpoints . . . . .	8
3.1.3	Python Object Paths . . . . .	14
	Organizing Collection . . . . .	15
	Collection . . . . .	16
	Resource . . . . .	16
	Subcollection . . . . .	17
	Subcollection Resource . . . . .	18
3.1.4	Coding Example . . . . .	18
3.1.5	Further Reading . . . . .	20
3.2	Developer Guide . . . . .	20
3.3	f5 . . . . .	20
3.3.1	f5 package . . . . .	20
	f5.bigip . . . . .	20
	f5.common . . . . .	218
	f5.sdk_exception . . . . .	219
<b>4</b>	<b>Copyright</b>	<b>221</b>
<b>5</b>	<b>License</b>	<b>223</b>
5.1	Apache V2.0 . . . . .	223
5.2	Contributor License Agreement . . . . .	223
	<b>Python Module Index</b>	<b>225</b>



---

# 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](#).



---

## 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()
```





---

## 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	<i>OrganizingCollection</i> , <i>Collection</i>	<i>exists()</i>
state	<i>Resource</i>	<i>create()</i> , <i>update()</i> , <i>refresh()</i> , <i>delete()</i> , <i>load()</i> , <i>exists()</i>
stats	<i>Resource</i>	<i>refresh()</i> , <i>load()</i> , <i>exists()</i>

## Methods

Method	HTTP Command	Action(s)
<code>create()</code>	POST	creates a new resource on the device with its own URI
<code>update()</code>	PUT	submits a new configuration to the device resource; sets the Resource attributes to the state reported by the device
<code>refresh()</code>	GET	obtains the state of a device resource; sets the representing Python Resource Object; tracks device state via its attributes
<code>delete()</code>	DELETE	removes the resource from the device, sets <code>self.__dict__</code> to <code>{ 'deleted' : True }</code>
<code>load()</code>	GET	obtains the state of an existing resource on the device; sets the Resource attributes to match that state
<code>exists()</code>	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	<i>Organizing Collection</i>
Coll	<i>Collection</i>
Resource	<i>Resource</i>
SC	<i>Subcollection</i>
SubColl Resrc	<i>Subcollection Resource</i>

## 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=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",
      }
    ]
  },
}

```

## 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>.<subcollection>`
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	<i>Organizing Collection</i>
Coll	<i>Collection</i>
Resource	<i>Resource</i>
SC	<i>Subcollection</i>
SubColl Resrc	<i>Subcollection Resource</i>

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	<code>http://192.168.1.1/mgmt/tm/</code>
Kind	<code>tm:restgroupresolvviewstate</code>
Type	organizing collection
Class	<code>f5.bigip.BigIP</code>
Instantiation	<code>bigip = BigIP('192.168.1.1', 'myuser', 'mypass')</code>

Endpoint	<code>http://192.168.1.1/mgmt/tm/ltm</code>
Kind	<code>tm:ltm:collectionstate</code>
Type	organizing collection
Class	<code>f5.bigip.ltm</code>
Instantiation	<code>ltm = bigip.ltm</code>

### 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	<a href="http://192.168.1.1/mgmt/tm/ltm/pool">http://192.168.1.1/mgmt/tm/ltm/pool</a>
Kind	tm:ltm:pool:poolcollectionstate
Type	collection
Class	<a href="#">f5.bigip.ltm.pool.Pools</a>
Instantiation	<code>pools = bigip.ltm.pools</code>

### 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	<a href="http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/">http://192.168.1.1/mgmt/tm/ltm/pool/~Common~mypool/</a>
Kind	tm:ltm:pool:poolstate
Type	resource
Class	<a href="#">f5.bigip.ltm.pool.Pool</a>
Instantiation	<code>pool = pools.pool.load(partition='Common', name='mypool')</code>

### 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	<a href="http://192.168.1.1/mgmt/tm/ltn/pool/~Common~mypool/members">http://192.168.1.1/mgmt/tm/ltn/pool/~Common~mypool/members</a>
Kind	tm:ltn:pool:members:memberscollectionstate
Type	subcollection
Class	<code>f5.bigip.ltn.pool.Members_s</code>
Instantiation	<code>members = pool.members_s</code>

**Example: Load the members\_s collection**

```
from f5.bigip import BigIP
members = pool.members_s.get_collection()
print members
[<f5.bigip.ltn.pool.Members object at 0x9d7ff0>, <f5.bigip.ltn.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	<a href="http://192.168.1.1/mgmt/tm/ltn/pool/~Common~mypool/members/~Common~member1">http://192.168.1.1/mgmt/tm/ltn/pool/~Common~mypool/members/~Common~member1</a>
Kind	tm:ltn:pool:members:membersstate
Type	subcollection resource
Class	<code>f5.bigip.ltn.pool.Members</code>
Instantiation	<code>members = pool.members_s.members.load(partition='Common', name='member1:&lt;port&gt;')</code>

**Example: Load member objects**

```
from f5.bigip import BigIP
member = members_s.members.load(partition='Common', name='m1')
print member
<f5.bigip.ltn.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()

```

### 3.1. User Guide

```

# Make sure it is gone
if pool_1.members_s.members.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

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

### Organizing Collection Modules

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

### Resource Base Classes

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

Continued on next page



Table 3.3 – continued from previous page

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

## Resource Exceptions

<code>mixins.ToDictMixin</code>	Convert an object's attributes to a dictionary
<code>mixins.LazyAttributesMixin</code>	
<code>mixins.ExclusiveAttributesMixin</code>	Overrides <code>__setattr__</code> to remove exclusive attrs from the object.
<code>mixins.UnnamedResourceMixin</code>	This makes a resource object work if there is no name.
<code>mixins.LazyAttributesRequired</code>	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 subsequent 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:*`

<a href="#"><i>device</i></a>	BIG-IP® cluster device submodule
<a href="#"><i>device_group</i></a>	BIG-IP® cluster device-group submodule
<a href="#"><i>traffic_group</i></a>	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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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:\*

<a href="#">monitor</a>	BIG-IP® LTM monitor submodule.
<a href="#">nat</a>	BIG-IP® Local Traffic Manager (LTM) Nat module.
<a href="#">node</a>	BIG-IP® Local Traffic Manager (LTM) node module.
<a href="#">policy</a>	BIG-IP® Local Traffic Manager (LTM) policy module.
<a href="#">pool</a>	BIG-IP® Local Traffic Manager™ (LTM®) pool module.
<a href="#">rule</a>	BIG-IP® Local Traffic Manager (LTM) rule module.
<a href="#">snat</a>	BIG-IP® Local Traffic Manager (LTM) Snat module.
<a href="#">snatpool</a>	BIG-IP Local Traffic Manager (LTM) SNAT pool module.
<a href="#">snat_translation</a>	BIG-IP Local Traffic Manager (LTM) SNAT Translation module.
<a href="#">ssl</a>	This module provides some more Pythonic support for SSL.
<a href="#">virtual</a>	BIG-IP® Local Traffic Manager (LTM) virtual module.
<a href="#">virtual_address</a>	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 subsequent 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*`

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

Continued on next page

Table 3.7 – continued from previous page

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

Continued on next page

Table 3.7 – continued from previous page

<i>Wap</i> (waps)	BIG-IP® Wap monitor resource.
<i>Wmis</i> (monitor)	BIG-IP® Wmi monitor collection.
<i>Wmi</i> (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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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** (fips)

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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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` (*postgressqls*)

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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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® Sasps 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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.**Smb**s(*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 subsequent 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(smb)`

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 subsequent 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® Smtip 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 subsequent 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.Smtplib(smtplib)`

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

BIG-IP® Smtplib 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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/ltn/nat`

**GUI Path** Local Traffic --> Nat

**REST Kind** `tm:ltn:nat:*`

---

<code>Nats(ltm)</code>	BIG-IP® LTM Nat collection object
<code>Nat(nat_s)</code>	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 subsequent 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 subsequent 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:*`

---

<i>Nodes(ltm)</i>	BIG-IP® LTM node collection
<i>Node(nodes)</i>	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 subsequent 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 subsequent 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:*`

<i><a href="#">Policys</a></i> (ltm)	BIG-IP® LTM policy collection.
<i><a href="#">Policy</a></i> (policy_s)	BIG-IP® LTM policy resource.
<i><a href="#">Rules_s</a></i> (policy)	BIG-IP® LTM policy rules sub-collection.
<i><a href="#">Rules</a></i> (rules_s)	BIG-IP® LTM policy rules sub-collection resource.
<i><a href="#">Actions_s</a></i> (rules)	BIG-IP® LTM policy actions sub-collection.
<i><a href="#">Actions</a></i> (actions_s)	BIG-IP® LTM policy actions sub-collection resource.
<i><a href="#">Conditions_s</a></i> (rules)	BIG-IP® LTM policy conditions sub-collection.
<i><a href="#">Conditions</a></i> (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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 `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 subsequent 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:*`

<code>Pools(ltm)</code>	BIG-IP® LTM pool collection
<code>Pool(pool_s)</code>	BIG-IP® LTM pool resource
<code>Members_s(pool)</code>	BIG-IP® LTM pool members sub-collection
<code>Member</code>	

### 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 subsequent 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 subsequent 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 idiosyncrasy 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 subsequent 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:*`

---

<i>Rules</i> (ltm)	BIG-IP® LTM rule collection
<i>Rule</i> (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 subsequent 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 subsequent 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:\*

---

<i>Snats(ltm)</i>	BIG-IP® LTM Snat collection
<i>Snat(snat_s)</i>	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 subsequent 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, translation 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 subsequent 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/ltn/snatpool?ver=11.6.0`

**GUI Path** Local Traffic --> Address Translation --> SNAT Pool List

**REST Kind** `tm:ltn:snatpool:*`

---

<i>Snatpools</i> (ltn)	BIG-IP SNAT pool collection.
<i>Snatpool</i> (Snatpools)	BIG-IP SNAT Pool resource.

---

### Snat Collections and Resources

**class** `f5.bigip.ltm.snatpool.Snatpools` (*ltn*)

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 subsequent 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:*`

---

<code><i>Snat_Translations</i>(ltm)</code>	BIG-IP SNAT Translation collection.
<code><i>Snat_Translation</i>(Snat_Translations)</code>	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 subsequent 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 subsequent 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:\*

---

<i>Virtuals</i> (ltm)	BIG-IP® LTM virtual collection
<i>Virtual</i> (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 subsequent 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 subsequent 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 subsequent 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 subsequent 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:*`

---

<i>Virtual_Address_s</i> (ltm)	BIG-IP LTM virtual address collection.
<i>Virtual_Address</i> (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 subsequent 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 subsequent 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:\*

<a href="#"><i>arp</i></a>	BIG-IP® Network ARP module.
<a href="#"><i>fdb</i></a>	Directory: net module: fdb.
<a href="#"><i>interface</i></a>	BIG-IP® Network interface module.
<a href="#"><i>route</i></a>	BIG-IP® Network route module.
<a href="#"><i>route_domain</i></a>	Directory: net module: route-domain.
<a href="#"><i>selfip</i></a>	BIG-IP® Network self-ip module.

Continued on next page



Table 3.18 – continued from previous page

<i>tunnels</i>	BIG-IP® Network tunnels module.
<i>vlan</i>	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:*`

<i>Arps</i> (net)	BIG-IP® network ARP collection
<i>Arp</i> (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 subsequent 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 subsequent 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:\*

---

<i>Interfaces</i> (net)	BIG-IP® network interface collection
<i>Interface</i> (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 subsequent 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 subsequent 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:*`

---

<a href="#"><i>Routes</i>(net)</a>	BIG-IP® network route collection
<a href="#"><i>Route</i>(route_s)</a>	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 subsequent 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 subsequent 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:*`

---

<code>Route_Domains(net)</code>	A Collection concrete subclass docstring.
<code>Route_Domain(Route_Domains)</code>	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 subsequent 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 subsequent 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:*`

---

<code>Selfips(net)</code>	BIG-IP® network Self-IP collection
<code>Selfip(selfip_s)</code>	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 subsequent 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 subsequent 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:*`

<i>Tunnels_s</i> (net)	BIG-IP® network tunnels collection
<i>Tunnels</i> (tunnels_s)	BIG-IP® network tunnels resource (collection for GRE, Tunnel, VXLANs)
<i>Tunnel</i> (tunnels)	BIG-IP® tunnels tunnel resource
<i>Gres</i> (tunnels_s)	BIG-IP® tunnels GRE sub-collection
<i>Gre</i> (gres)	BIG-IP® tunnels GRE sub-collection resource
<i>Vxlans</i> (tunnels_s)	BIG-IP® tunnels VXLAN sub-collection
<i>Vxlan</i> (vxlan)	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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 replaced 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 subsequent 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:*`

<code>Vlans(net)</code>	BIG-IP® network Vlan collection.
<code>Vlan(vlan_s)</code>	BIG-IP® network Vlan resource.
<code>Interfaces_s(vlan)</code>	BIG-IP® network Vlan interface collection.
<code>Interfaces(interfaces_s)</code>	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 subsequent 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 subsequent 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 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 subsequent 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 `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 subsequent 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:*`

<i>Fdbs</i> (net)	A Collection concrete subclass docstring.
<i>Tunnel</i> (Tunnels)	A Resource concrete subclass.
<i>Tunnels</i> (fdb)	A Collection concrete subclass docstring.
<i>Vlans</i> (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 subsequent 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 subsequent 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 subsequent 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 subsequent 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

---

<code><i>bigip_failover_state</i></code>	BIG-IP® shared failover state module
<code><i>licensing</i></code>	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 subsequent 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 subsequent 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 subsequent 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:*`

<a href="#"><i>application</i></a>	BIG-IP® iApp (application) module
<a href="#"><i>db</i></a>	BIG-IP® db module
<a href="#"><i>failover</i></a>	BIG-IP® system failover module
<a href="#"><i>folder</i></a>	BIG-IP® system folder (partition) module
<a href="#"><i>global_settings</i></a>	BIG-IP® system global-settings module
<a href="#"><i>ntp</i></a>	BIG-IP® system ntp module
<a href="#"><i>performance</i></a>	BIG-IP® system performance 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:*`

<i>Applications</i> (sys)	BIG-IP® iApp collection.
<i>Aplscripts</i> (application)	BIG-IP® iApp script collection.
<i>Aplscript</i> (apl_script_s)	BIG-IP® iApp script resource.
<i>Customstats</i> (application)	BIG-IP® iApp custom stats sub-collection.
<i>Customstat</i> (custom_stat_s)	BIG-IP® iApp custom stats sub-collection resource.
<i>Services</i> (application)	BIG-IP® iApp service sub-collection.
<i>Service</i> (service_s)	BIG-IP® iApp service sub-collection resource
<i>Templates</i> (application)	BIG-IP® iApp template sub-collection
<i>Template</i> (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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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 subsequent 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:\*

---

<i>Db</i> s(sys)	BIG-IP® db collection
<i>Db</i> (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 subsequent 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 subsequent 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:*`

---

<i>Folders</i> (sys)	BIG-IP® system folder collection.
<i>Folder</i> (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 subsequent 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 subsequent 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:*`

<code>Ntp(sys)</code>	BIG-IP® system NTP unnamed resource
<code>Restricts(ntp)</code>	BIG-IP® system NTP restrict sub-collection
<code>Restrict(restricts)</code>	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 subsequent 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.Restrict`(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 subsequent 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 replaced 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 object 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 subsequent 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 to 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 performance stats module.

**REST URI** http://localhost/mgmt/tm/sys/performance

**GUI Path** System --> Users --> Partition List

**REST Kind** tm:sys:performance:\*

<i>Performance</i> (sys)	BIG-IP® system performace stats collection
<i>All_Stats</i> (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 subsequent 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 expected 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 subsequent 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 subsequent 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 subsequent 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.





---

**Copyright**

---

Copyright 2014-2016 F5 Networks Inc.



---

**License**

---

## **5.1 Apache V2.0**

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

## **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](mailto:Openstack_CLA@f5.com) prior to their code submission being included in this project.



## f

- f5, 219
- f5.bigip, 20
- f5.bigip.cm, 22
- f5.bigip.cm.device, 23
- f5.bigip.cm.device\_group, 25
- f5.bigip.cm.traffic\_group, 29
- f5.bigip.ltm, 32
- f5.bigip.ltm.monitor, 33
- f5.bigip.ltm.nat, 121
- f5.bigip.ltm.node, 123
- f5.bigip.ltm.policy, 125
- f5.bigip.ltm.pool, 134
- f5.bigip.ltm.rule, 139
- f5.bigip.ltm.snat, 142
- f5.bigip.ltm.snat\_translation, 147
- f5.bigip.ltm.snatpool, 144
- f5.bigip.ltm.virtual, 149
- f5.bigip.ltm.virtual\_address, 154
- f5.bigip.mixins, 217
- f5.bigip.net, 156
- f5.bigip.net.arp, 157
- f5.bigip.net.fdb, 181
- f5.bigip.net.interface, 159
- f5.bigip.net.route, 161
- f5.bigip.net.route\_domain, 164
- f5.bigip.net.selfip, 166
- f5.bigip.net.tunnels, 169
- f5.bigip.net.vlan, 177
- f5.bigip.resource, 210
- f5.bigip.shared, 186
- f5.bigip.shared.bigip\_failover\_state, 186
- f5.bigip.shared.licensing, 187
- f5.bigip.sys, 189
- f5.bigip.sys.application, 189
- f5.bigip.sys.db, 199
- f5.bigip.sys.failover, 202
- f5.bigip.sys.folder, 203
- f5.bigip.sys.global\_settings, 204
- f5.bigip.sys.ntp, 205
- f5.bigip.sys.performance, 209
- f5.common, 219
- f5.common.constants, 218
- f5.common.iapp\_parser, 218
- f5.sdk\_exception, 219



## A

Actions (class in f5.bigip.ltm.policy), 131  
 Actions\_s (class in f5.bigip.ltm.policy), 130  
 Activation (class in f5.bigip.shared.licensing), 187  
 All\_Stats (class in f5.bigip.sys.performance), 209  
 Aplscript (class in f5.bigip.sys.application), 191  
 Aplscripts (class in f5.bigip.sys.application), 191  
 Applications (class in f5.bigip.sys.application), 190  
 args (f5.common.iapp\_parser.CurlyBraceMismatchException attribute), 218  
 args (f5.common.iapp\_parser.EmptyTemplateException attribute), 218  
 args (f5.common.iapp\_parser.MalformedTCLListException attribute), 219  
 args (f5.common.iapp\_parser.NonextantSectionException attribute), 218  
 args (f5.common.iapp\_parser.NonextantTemplateNameException attribute), 218  
 Arp (class in f5.bigip.net.arp), 158  
 Arps (class in f5.bigip.net.arp), 157  
 attr\_map (f5.common.iapp\_parser.IappParser attribute), 218

## B

BigIP (class in f5.bigip), 21  
 Bigip\_Failover\_State (class in f5.bigip.shared.bigip\_failover\_state), 186

## C

Cm (class in f5.bigip.cm), 22  
 Collection (class in f5.bigip.resource), 214  
 Conditions (class in f5.bigip.ltm.policy), 133  
 Conditions\_s (class in f5.bigip.ltm.policy), 132  
 create() (f5.bigip.BigIP method), 21  
 create() (f5.bigip.cm.Cm method), 22  
 create() (f5.bigip.cm.device.Device method), 23  
 create() (f5.bigip.cm.device.Devices method), 23  
 create() (f5.bigip.cm.device\_group.Device\_Group method), 26  
 create() (f5.bigip.cm.device\_group.Device\_Groups method), 25  
 create() (f5.bigip.cm.device\_group.Devices method), 28  
 create() (f5.bigip.cm.device\_group.Devices\_s method), 27  
 create() (f5.bigip.cm.traffic\_group.Traffic\_Group method), 30  
 create() (f5.bigip.cm.traffic\_group.Traffic\_Groups method), 30  
 create() (f5.bigip.ltm.Ltm method), 32  
 create() (f5.bigip.ltm.monitor.Diameter method), 40  
 create() (f5.bigip.ltm.monitor.Diameters method), 39  
 create() (f5.bigip.ltm.monitor.Dns method), 42  
 create() (f5.bigip.ltm.monitor.Dns\_s method), 41  
 create() (f5.bigip.ltm.monitor.External method), 44  
 create() (f5.bigip.ltm.monitor.Externals method), 43  
 create() (f5.bigip.ltm.monitor.Firepass method), 46  
 create() (f5.bigip.ltm.monitor.Firepass\_s method), 46  
 create() (f5.bigip.ltm.monitor.Ftp method), 49  
 create() (f5.bigip.ltm.monitor.Ftps method), 48  
 create() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 51  
 create() (f5.bigip.ltm.monitor.Gateway\_Icmps method), 50  
 create() (f5.bigip.ltm.monitor.Http method), 35  
 create() (f5.bigip.ltm.monitor.HttpS method), 38  
 create() (f5.bigip.ltm.monitor.Https method), 35  
 create() (f5.bigip.ltm.monitor.Https\_s method), 37  
 create() (f5.bigip.ltm.monitor.Icmp method), 53  
 create() (f5.bigip.ltm.monitor.Icmps method), 52  
 create() (f5.bigip.ltm.monitor.Imap method), 55  
 create() (f5.bigip.ltm.monitor.Imaps method), 54  
 create() (f5.bigip.ltm.monitor.Inband method), 57  
 create() (f5.bigip.ltm.monitor.Inbands method), 57  
 create() (f5.bigip.ltm.monitor.Ldap method), 60  
 create() (f5.bigip.ltm.monitor.Ldaps method), 59  
 create() (f5.bigip.ltm.monitor.Module\_Score method), 62  
 create() (f5.bigip.ltm.monitor.Module\_Scores method), 61  
 create() (f5.bigip.ltm.monitor.Mssql method), 66  
 create() (f5.bigip.ltm.monitor.Mssqls method), 65  
 create() (f5.bigip.ltm.monitor.Mysql method), 64  
 create() (f5.bigip.ltm.monitor.Mysqls method), 63  
 create() (f5.bigip.ltm.monitor.Nntp method), 68

`create()` (f5.bigip.ltm.monitor.Nntps method), 68  
`create()` (f5.bigip.ltm.monitor.NONE method), 71  
`create()` (f5.bigip.ltm.monitor.Nones method), 70  
`create()` (f5.bigip.ltm.monitor.Oracle method), 73  
`create()` (f5.bigip.ltm.monitor.Oracles method), 72  
`create()` (f5.bigip.ltm.monitor.Pop3 method), 75  
`create()` (f5.bigip.ltm.monitor.Pop3s method), 74  
`create()` (f5.bigip.ltm.monitor.Postgresql method), 77  
`create()` (f5.bigip.ltm.monitor.Postgresqls method), 76  
`create()` (f5.bigip.ltm.monitor.Radius method), 79  
`create()` (f5.bigip.ltm.monitor.Radius\_Accounting method), 82  
`create()` (f5.bigip.ltm.monitor.Radius\_Accountings method), 81  
`create()` (f5.bigip.ltm.monitor.Radius\_s method), 79  
`create()` (f5.bigip.ltm.monitor.Real\_Server method), 84  
`create()` (f5.bigip.ltm.monitor.Real\_Servers method), 83  
`create()` (f5.bigip.ltm.monitor.Rpc method), 86  
`create()` (f5.bigip.ltm.monitor.Rpcs method), 85  
`create()` (f5.bigip.ltm.monitor.Sasp method), 88  
`create()` (f5.bigip.ltm.monitor.Sasps method), 87  
`create()` (f5.bigip.ltm.monitor.Scripted method), 90  
`create()` (f5.bigip.ltm.monitor.Scripteds method), 90  
`create()` (f5.bigip.ltm.monitor.Sip method), 93  
`create()` (f5.bigip.ltm.monitor.Sips method), 92  
`create()` (f5.bigip.ltm.monitor.Smb method), 95  
`create()` (f5.bigip.ltm.monitor.Smbs method), 94  
`create()` (f5.bigip.ltm.monitor.Smtp method), 97  
`create()` (f5.bigip.ltm.monitor.Smtps method), 96  
`create()` (f5.bigip.ltm.monitor.Snmp\_Dca method), 99  
`create()` (f5.bigip.ltm.monitor.Snmp\_Dca\_Base method), 101  
`create()` (f5.bigip.ltm.monitor.Snmp\_Dca\_Bases method), 101  
`create()` (f5.bigip.ltm.monitor.Snmp\_Dcas method), 98  
`create()` (f5.bigip.ltm.monitor.Soap method), 104  
`create()` (f5.bigip.ltm.monitor.Soaps method), 103  
`create()` (f5.bigip.ltm.monitor.Tcp method), 106  
`create()` (f5.bigip.ltm.monitor.Tcp\_Echo method), 108  
`create()` (f5.bigip.ltm.monitor.Tcp\_Echos method), 107  
`create()` (f5.bigip.ltm.monitor.Tcp\_Half\_Open method), 110  
`create()` (f5.bigip.ltm.monitor.Tcp\_Half\_Opens method), 109  
`create()` (f5.bigip.ltm.monitor.Tcps method), 105  
`create()` (f5.bigip.ltm.monitor.Udp method), 112  
`create()` (f5.bigip.ltm.monitor.Udps method), 112  
`create()` (f5.bigip.ltm.monitor.Virtual\_Location method), 115  
`create()` (f5.bigip.ltm.monitor.Virtual\_Locations method), 114  
`create()` (f5.bigip.ltm.monitor.Wap method), 117  
`create()` (f5.bigip.ltm.monitor.Waps method), 116  
`create()` (f5.bigip.ltm.monitor.Wmi method), 119  
`create()` (f5.bigip.ltm.monitor.Wmis method), 118  
`create()` (f5.bigip.ltm.nat.Nat method), 122  
`create()` (f5.bigip.ltm.nat.Nats method), 121  
`create()` (f5.bigip.ltm.node.Node method), 124  
`create()` (f5.bigip.ltm.node.Nodes method), 123  
`create()` (f5.bigip.ltm.policy.Actions method), 131  
`create()` (f5.bigip.ltm.policy.Actions\_s method), 130  
`create()` (f5.bigip.ltm.policy.Conditions method), 133  
`create()` (f5.bigip.ltm.policy.Conditions\_s method), 132  
`create()` (f5.bigip.ltm.policy.Policy method), 126  
`create()` (f5.bigip.ltm.policy.Policys method), 126  
`create()` (f5.bigip.ltm.policy.Rules method), 129  
`create()` (f5.bigip.ltm.policy.Rules\_s method), 128  
`create()` (f5.bigip.ltm.pool.Members method), 138  
`create()` (f5.bigip.ltm.pool.Members\_s method), 137  
`create()` (f5.bigip.ltm.pool.Pool method), 135  
`create()` (f5.bigip.ltm.pool.Pools method), 135  
`create()` (f5.bigip.ltm.rule.Rule method), 140  
`create()` (f5.bigip.ltm.rule.Rules method), 139  
`create()` (f5.bigip.ltm.snat.Snat method), 143  
`create()` (f5.bigip.ltm.snat.Snats method), 142  
`create()` (f5.bigip.ltm.snat\_translation.Snat\_Translation method), 148  
`create()` (f5.bigip.ltm.snat\_translation.Snat\_Translations method), 147  
`create()` (f5.bigip.ltm.snatpool.Snatpool method), 145  
`create()` (f5.bigip.ltm.snatpool.Snatpools method), 144  
`create()` (f5.bigip.ltm.virtual.Profiles method), 152  
`create()` (f5.bigip.ltm.virtual.Profiles\_s method), 153  
`create()` (f5.bigip.ltm.virtual.Virtual method), 150  
`create()` (f5.bigip.ltm.virtual.Virtuals method), 149  
`create()` (f5.bigip.ltm.virtual\_address.Virtual\_Address method), 155  
`create()` (f5.bigip.ltm.virtual\_address.Virtual\_Address\_s method), 154  
`create()` (f5.bigip.mixins.UnnamedResourceMixin method), 217  
`create()` (f5.bigip.net.arp.Arp method), 158  
`create()` (f5.bigip.net.arp.Arps method), 157  
`create()` (f5.bigip.net.fdb.Fdbs method), 182  
`create()` (f5.bigip.net.fdb.Tunnel method), 182  
`create()` (f5.bigip.net.fdb.Tunnels method), 184  
`create()` (f5.bigip.net.fdb.Vlans method), 185  
`create()` (f5.bigip.net.interface.Interface method), 160  
`create()` (f5.bigip.net.interface.Interfaces method), 159  
`create()` (f5.bigip.net.route.Route method), 162  
`create()` (f5.bigip.net.route.Routes method), 162  
`create()` (f5.bigip.net.route\_domain.Route\_Domain method), 165  
`create()` (f5.bigip.net.route\_domain.Route\_Domains method), 164  
`create()` (f5.bigip.net.selfip.Selfip method), 168  
`create()` (f5.bigip.net.selfip.Selfips method), 167  
`create()` (f5.bigip.net.tunnels.Gre method), 173



- create() (f5.bigip.net.tunnels.Gres method), 172
- create() (f5.bigip.net.tunnels.Tunnel method), 171
- create() (f5.bigip.net.tunnels.Tunnels method), 170
- create() (f5.bigip.net.tunnels.Tunnels\_s method), 169
- create() (f5.bigip.net.tunnels.Vxlan method), 175
- create() (f5.bigip.net.tunnels.Vxlans method), 174
- create() (f5.bigip.net.vlan.Interfaces method), 180
- create() (f5.bigip.net.vlan.Interfaces\_s method), 179
- create() (f5.bigip.net.vlan.Vlan method), 178
- create() (f5.bigip.net.vlan.Vlans method), 177
- create() (f5.bigip.resource.Collection method), 214
- create() (f5.bigip.resource.OrganizingCollection method), 213
- create() (f5.bigip.resource.Resource method), 215
- create() (f5.bigip.resource.ResourceBase method), 213
- create() (f5.bigip.shared.bigip\_failover\_state.Bigip\_Failover\_State method), 186
- create() (f5.bigip.shared.licensing.Activation method), 187
- create() (f5.bigip.shared.licensing.Registration method), 188
- create() (f5.bigip.sys.application.Aplscript method), 191
- create() (f5.bigip.sys.application.Aplscripts method), 191
- create() (f5.bigip.sys.application.Applications method), 190
- create() (f5.bigip.sys.application.Customstat method), 194
- create() (f5.bigip.sys.application.Customstats method), 193
- create() (f5.bigip.sys.application.Service method), 196
- create() (f5.bigip.sys.application.Services method), 195
- create() (f5.bigip.sys.application.Template method), 198
- create() (f5.bigip.sys.application.Templates method), 197
- create() (f5.bigip.sys.db.Db method), 200
- create() (f5.bigip.sys.db.Dbs method), 200
- create() (f5.bigip.sys.failover.Failover method), 202
- create() (f5.bigip.sys.folder.Folders method), 203
- create() (f5.bigip.sys.global\_settings.Global\_Settings method), 204
- create() (f5.bigip.sys.ntp.Ntp method), 205
- create() (f5.bigip.sys.ntp.Restrict method), 207
- create() (f5.bigip.sys.ntp.Restricts method), 206
- create() (f5.bigip.sys.performance.All\_Stats method), 210
- create() (f5.bigip.sys.performance.Performance method), 209
- CurlyBraceMismatchException, 218
- Customstat (class in f5.bigip.sys.application), 194
- Customstats (class in f5.bigip.sys.application), 193
- D**
- Db (class in f5.bigip.sys.db), 200
- Dbs (class in f5.bigip.sys.db), 199
- delete() (f5.bigip.BigIP method), 21
- delete() (f5.bigip.cm.Cm method), 22
- delete() (f5.bigip.cm.device.Device method), 24
- delete() (f5.bigip.cm.device.Devices method), 23
- delete() (f5.bigip.cm.device\_group.Device\_Group method), 26
- delete() (f5.bigip.cm.device\_group.Device\_Groups method), 25
- delete() (f5.bigip.cm.device\_group.Devices method), 28
- delete() (f5.bigip.cm.device\_group.Devices\_s method), 27
- delete() (f5.bigip.cm.traffic\_group.Traffic\_Group method), 31
- delete() (f5.bigip.cm.traffic\_group.Traffic\_Groups method), 30
- delete() (f5.bigip.ltm.Ltm method), 32
- delete() (f5.bigip.ltm.monitor.Diameter method), 40
- delete() (f5.bigip.ltm.monitor.Diameters method), 39
- delete() (f5.bigip.ltm.monitor.Dns method), 42
- delete() (f5.bigip.ltm.monitor.Dns\_s method), 41
- delete() (f5.bigip.ltm.monitor.External method), 44
- delete() (f5.bigip.ltm.monitor.Externals method), 44
- delete() (f5.bigip.ltm.monitor.Firepass method), 47
- delete() (f5.bigip.ltm.monitor.Firepass\_s method), 46
- delete() (f5.bigip.ltm.monitor.Ftp method), 49
- delete() (f5.bigip.ltm.monitor.Ftps method), 48
- delete() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 51
- delete() (f5.bigip.ltm.monitor.Gateway\_Icmps method), 50
- delete() (f5.bigip.ltm.monitor.Http method), 36
- delete() (f5.bigip.ltm.monitor.HttpS method), 38
- delete() (f5.bigip.ltm.monitor.Https method), 35
- delete() (f5.bigip.ltm.monitor.Https\_s method), 37
- delete() (f5.bigip.ltm.monitor.Icmp method), 53
- delete() (f5.bigip.ltm.monitor.Icmps method), 52
- delete() (f5.bigip.ltm.monitor.Imap method), 55
- delete() (f5.bigip.ltm.monitor.Imaps method), 55
- delete() (f5.bigip.ltm.monitor.Inband method), 58
- delete() (f5.bigip.ltm.monitor.Inbands method), 57
- delete() (f5.bigip.ltm.monitor.Ldap method), 60
- delete() (f5.bigip.ltm.monitor.Ldaps method), 59
- delete() (f5.bigip.ltm.monitor.Module\_Score method), 62
- delete() (f5.bigip.ltm.monitor.Module\_Scores method), 61
- delete() (f5.bigip.ltm.monitor.Mssql method), 66
- delete() (f5.bigip.ltm.monitor.Mssqls method), 66
- delete() (f5.bigip.ltm.monitor.Mysql method), 64
- delete() (f5.bigip.ltm.monitor.Mysqls method), 63
- delete() (f5.bigip.ltm.monitor.Nntp method), 69
- delete() (f5.bigip.ltm.monitor.Nntps method), 68
- delete() (f5.bigip.ltm.monitor.NONE method), 71
- delete() (f5.bigip.ltm.monitor.Nones method), 70
- delete() (f5.bigip.ltm.monitor.Oracle method), 73
- delete() (f5.bigip.ltm.monitor.Oracles method), 72
- delete() (f5.bigip.ltm.monitor.Pop3 method), 75

`delete()` (f5.bigip.ltm.monitor.Pop3s method), 74  
`delete()` (f5.bigip.ltm.monitor.Postgresql method), 77  
`delete()` (f5.bigip.ltm.monitor.Postgresqls method), 77  
`delete()` (f5.bigip.ltm.monitor.Radius method), 80  
`delete()` (f5.bigip.ltm.monitor.Radius\_Accounting method), 82  
`delete()` (f5.bigip.ltm.monitor.Radius\_Accountings method), 81  
`delete()` (f5.bigip.ltm.monitor.Radius\_s method), 79  
`delete()` (f5.bigip.ltm.monitor.Real\_Server method), 84  
`delete()` (f5.bigip.ltm.monitor.Real\_Servers method), 83  
`delete()` (f5.bigip.ltm.monitor.Rpc method), 86  
`delete()` (f5.bigip.ltm.monitor.Rpcs method), 85  
`delete()` (f5.bigip.ltm.monitor.Sasp method), 88  
`delete()` (f5.bigip.ltm.monitor.Sasps method), 88  
`delete()` (f5.bigip.ltm.monitor.Scripted method), 91  
`delete()` (f5.bigip.ltm.monitor.Scripteds method), 90  
`delete()` (f5.bigip.ltm.monitor.Sip method), 93  
`delete()` (f5.bigip.ltm.monitor.Sips method), 92  
`delete()` (f5.bigip.ltm.monitor.Smb method), 95  
`delete()` (f5.bigip.ltm.monitor.Smbs method), 94  
`delete()` (f5.bigip.ltm.monitor.Smtp method), 97  
`delete()` (f5.bigip.ltm.monitor.Smtps method), 96  
`delete()` (f5.bigip.ltm.monitor.Snmp\_Dca method), 99  
`delete()` (f5.bigip.ltm.monitor.Snmp\_Dca\_Base method), 102  
`delete()` (f5.bigip.ltm.monitor.Snmp\_Dca\_Bases method), 101  
`delete()` (f5.bigip.ltm.monitor.Snmp\_Dcas method), 99  
`delete()` (f5.bigip.ltm.monitor.Soap method), 104  
`delete()` (f5.bigip.ltm.monitor.Soaps method), 103  
`delete()` (f5.bigip.ltm.monitor.Tcp method), 106  
`delete()` (f5.bigip.ltm.monitor.Tcp\_Echo method), 108  
`delete()` (f5.bigip.ltm.monitor.Tcp\_Echos method), 107  
`delete()` (f5.bigip.ltm.monitor.Tcp\_Half\_Open method), 110  
`delete()` (f5.bigip.ltm.monitor.Tcp\_Half\_Opens method), 110  
`delete()` (f5.bigip.ltm.monitor.Tcps method), 105  
`delete()` (f5.bigip.ltm.monitor.Udp method), 113  
`delete()` (f5.bigip.ltm.monitor.Udps method), 112  
`delete()` (f5.bigip.ltm.monitor.Virtual\_Location method), 115  
`delete()` (f5.bigip.ltm.monitor.Virtual\_Locations method), 114  
`delete()` (f5.bigip.ltm.monitor.Wap method), 117  
`delete()` (f5.bigip.ltm.monitor.Waps method), 116  
`delete()` (f5.bigip.ltm.monitor.Wmi method), 120  
`delete()` (f5.bigip.ltm.monitor.Wmis method), 118  
`delete()` (f5.bigip.ltm.nat.Nat method), 122  
`delete()` (f5.bigip.ltm.nat.Nats method), 121  
`delete()` (f5.bigip.ltm.node.Node method), 124  
`delete()` (f5.bigip.ltm.node.Nodes method), 123  
`delete()` (f5.bigip.ltm.policy.Actions method), 131  
`delete()` (f5.bigip.ltm.policy.Actions\_s method), 130  
`delete()` (f5.bigip.ltm.policy.Conditions method), 133  
`delete()` (f5.bigip.ltm.policy.Conditions\_s method), 132  
`delete()` (f5.bigip.ltm.policy.Policy method), 127  
`delete()` (f5.bigip.ltm.policy.Policys method), 126  
`delete()` (f5.bigip.ltm.policy.Rules method), 129  
`delete()` (f5.bigip.ltm.policy.Rules\_s method), 128  
`delete()` (f5.bigip.ltm.pool.Members method), 139  
`delete()` (f5.bigip.ltm.pool.Members\_s method), 137  
`delete()` (f5.bigip.ltm.pool.Pool method), 136  
`delete()` (f5.bigip.ltm.pool.Pools method), 135  
`delete()` (f5.bigip.ltm.rule.Rule method), 140  
`delete()` (f5.bigip.ltm.rule.Rules method), 140  
`delete()` (f5.bigip.ltm.snat.Snat method), 143  
`delete()` (f5.bigip.ltm.snat.Snats method), 142  
`delete()` (f5.bigip.ltm.snat\_translation.Snat\_Translation method), 148  
`delete()` (f5.bigip.ltm.snat\_translation.Snat\_Translations method), 147  
`delete()` (f5.bigip.ltm.snatpool.Snatpool method), 145  
`delete()` (f5.bigip.ltm.snatpool.Snatpools method), 145  
`delete()` (f5.bigip.ltm.virtual.Profiles method), 152  
`delete()` (f5.bigip.ltm.virtual.Profiles\_s method), 153  
`delete()` (f5.bigip.ltm.virtual.Virtual method), 150  
`delete()` (f5.bigip.ltm.virtual.Virtuals method), 149  
`delete()` (f5.bigip.ltm.virtual\_address.Virtual\_Address method), 155  
`delete()` (f5.bigip.ltm.virtual\_address.Virtual\_Address\_s method), 154  
`delete()` (f5.bigip.mixins.UnnamedResourceMixin method), 217  
`delete()` (f5.bigip.net.arp.Arp method), 158  
`delete()` (f5.bigip.net.arp.Arps method), 157  
`delete()` (f5.bigip.net.fdb.Fdbs method), 182  
`delete()` (f5.bigip.net.fdb.Tunnel method), 183  
`delete()` (f5.bigip.net.fdb.Tunnels method), 184  
`delete()` (f5.bigip.net.fdb.Vlans method), 185  
`delete()` (f5.bigip.net.interface.Interface method), 160  
`delete()` (f5.bigip.net.interface.Interfaces method), 159  
`delete()` (f5.bigip.net.route.Route method), 163  
`delete()` (f5.bigip.net.route.Routes method), 162  
`delete()` (f5.bigip.net.route\_domain.Route\_Domain method), 165  
`delete()` (f5.bigip.net.route\_domain.Route\_Domains method), 164  
`delete()` (f5.bigip.net.selfip.Selfip method), 168  
`delete()` (f5.bigip.net.selfip.Selfips method), 167  
`delete()` (f5.bigip.net.tunnels.Gre method), 173  
`delete()` (f5.bigip.net.tunnels.Gres method), 172  
`delete()` (f5.bigip.net.tunnels.Tunnel method), 171  
`delete()` (f5.bigip.net.tunnels.Tunnels method), 170  
`delete()` (f5.bigip.net.tunnels.Tunnels\_s method), 169  
`delete()` (f5.bigip.net.tunnels.Vxlan method), 175  
`delete()` (f5.bigip.net.tunnels.Vxlans method), 175

- delete() (f5.bigip.net.vlan.Interfaces method), 180
- delete() (f5.bigip.net.vlan.Interfaces\_s method), 179
- delete() (f5.bigip.net.vlan.Vlan method), 178
- delete() (f5.bigip.net.vlan.Vlans method), 177
- delete() (f5.bigip.resource.Collection method), 214
- delete() (f5.bigip.resource.OrganizingCollection method), 214
- delete() (f5.bigip.resource.Resource method), 216
- delete() (f5.bigip.resource.ResourceBase method), 213
- delete() (f5.bigip.shared.bigip\_failover\_state.Bigip\_Failover\_State method), 186
- delete() (f5.bigip.shared.licensing.Activation method), 188
- delete() (f5.bigip.shared.licensing.Registration method), 188
- delete() (f5.bigip.sys.application.Aplscript method), 192
- delete() (f5.bigip.sys.application.Aplscripts method), 191
- delete() (f5.bigip.sys.application.Applications method), 190
- delete() (f5.bigip.sys.application.Customstat method), 194
- delete() (f5.bigip.sys.application.Customstats method), 193
- delete() (f5.bigip.sys.application.Service method), 196
- delete() (f5.bigip.sys.application.Services method), 195
- delete() (f5.bigip.sys.application.Template method), 198
- delete() (f5.bigip.sys.application.Templates method), 197
- delete() (f5.bigip.sys.db.Db method), 200
- delete() (f5.bigip.sys.db.Dbs method), 200
- delete() (f5.bigip.sys.failover.Failover method), 202
- delete() (f5.bigip.sys.folder.Folders method), 203
- delete() (f5.bigip.sys.global\_settings.Global\_Settings method), 204
- delete() (f5.bigip.sys.ntp.Ntp method), 206
- delete() (f5.bigip.sys.ntp.Restrict method), 207
- delete() (f5.bigip.sys.ntp.Restricts method), 207
- delete() (f5.bigip.sys.performance.All\_Stats method), 210
- delete() (f5.bigip.sys.performance.Performance method), 209
- Device (class in f5.bigip.cm.device), 23
- Device\_Group (class in f5.bigip.cm.device\_group), 26
- Device\_Groups (class in f5.bigip.cm.device\_group), 25
- DeviceProvidesIncompatibleKey, 211
- Devices (class in f5.bigip.cm.device), 23
- Devices (class in f5.bigip.cm.device\_group), 28
- Devices\_s (class in f5.bigip.cm.device\_group), 27
- Diameter (class in f5.bigip.ltm.monitor), 40
- Diameters (class in f5.bigip.ltm.monitor), 39
- Dns (class in f5.bigip.ltm.monitor), 42
- Dns\_s (class in f5.bigip.ltm.monitor), 41
- ExclusiveAttributesMixin (class in f5.bigip.mixins), 217
- exists() (f5.bigip.cm.device.Device method), 24
- exists() (f5.bigip.cm.device\_group.Device\_Group method), 26
- exists() (f5.bigip.cm.device\_group.Devices method), 28
- exists() (f5.bigip.cm.traffic\_group.Traffic\_Group method), 31
- exists() (f5.bigip.ltm.monitor.Diameter method), 40
- exists() (f5.bigip.ltm.monitor.Dns method), 42
- exists() (f5.bigip.ltm.monitor.External method), 45
- exists() (f5.bigip.ltm.monitor.Firepass method), 47
- exists() (f5.bigip.ltm.monitor.Ftp method), 49
- exists() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 51
- exists() (f5.bigip.ltm.monitor.Http method), 36
- exists() (f5.bigip.ltm.monitor.HttpS method), 38
- exists() (f5.bigip.ltm.monitor.Icmp method), 53
- exists() (f5.bigip.ltm.monitor.Imap method), 56
- exists() (f5.bigip.ltm.monitor.Inband method), 58
- exists() (f5.bigip.ltm.monitor.Ldap method), 60
- exists() (f5.bigip.ltm.monitor.Module\_Score method), 62
- exists() (f5.bigip.ltm.monitor.Mssql method), 67
- exists() (f5.bigip.ltm.monitor.Mysql method), 64
- exists() (f5.bigip.ltm.monitor.Nntp method), 69
- exists() (f5.bigip.ltm.monitor.NONE method), 71
- exists() (f5.bigip.ltm.monitor.Oracle method), 73
- exists() (f5.bigip.ltm.monitor.Pop3 method), 75
- exists() (f5.bigip.ltm.monitor.Postgresql method), 78
- exists() (f5.bigip.ltm.monitor.Radius method), 80
- exists() (f5.bigip.ltm.monitor.Radius\_Accounting method), 82
- exists() (f5.bigip.ltm.monitor.Real\_Server method), 85
- exists() (f5.bigip.ltm.monitor.Rpc method), 86
- exists() (f5.bigip.ltm.monitor.Sasp method), 89
- exists() (f5.bigip.ltm.monitor.Scripted method), 91
- exists() (f5.bigip.ltm.monitor.Sip method), 93
- exists() (f5.bigip.ltm.monitor.Smb method), 95
- exists() (f5.bigip.ltm.monitor.Smtp method), 97
- exists() (f5.bigip.ltm.monitor.Snmp\_Dca method), 100
- exists() (f5.bigip.ltm.monitor.Snmp\_Dca\_Base method), 102
- exists() (f5.bigip.ltm.monitor.Soap method), 104
- exists() (f5.bigip.ltm.monitor.Tcp method), 106
- exists() (f5.bigip.ltm.monitor.Tcp\_Echo method), 108
- exists() (f5.bigip.ltm.monitor.Tcp\_Half\_Open method), 111
- exists() (f5.bigip.ltm.monitor.Udp method), 113
- exists() (f5.bigip.ltm.monitor.Virtual\_Location method), 115
- exists() (f5.bigip.ltm.monitor.Wap method), 117
- exists() (f5.bigip.ltm.monitor.Wmi method), 120
- exists() (f5.bigip.ltm.nat.Nat method), 122
- exists() (f5.bigip.ltm.node.Node method), 124
- exists() (f5.bigip.ltm.policy.Actions method), 131
- exists() (f5.bigip.ltm.policy.Conditions method), 133

## E

EmptyTemplateException, 218

`exists()` (f5.bigip.ltm.policy.Policy method), 127  
`exists()` (f5.bigip.ltm.policy.Rules method), 129  
`exists()` (f5.bigip.ltm.pool.Members method), 138  
`exists()` (f5.bigip.ltm.pool.Pool method), 136  
`exists()` (f5.bigip.ltm.rule.Rule method), 141  
`exists()` (f5.bigip.ltm.snat.Snat method), 143  
`exists()` (f5.bigip.ltm.snat\_translation.Snat\_Translation method), 148  
`exists()` (f5.bigip.ltm.snatpool.Snatpool method), 146  
`exists()` (f5.bigip.ltm.virtual.Profiles method), 152  
`exists()` (f5.bigip.ltm.virtual.Virtual method), 151  
`exists()` (f5.bigip.ltm.virtual\_address.Virtual\_Address method), 155  
`exists()` (f5.bigip.net.arp.Arp method), 158  
`exists()` (f5.bigip.net.fdb.Tunnel method), 183  
`exists()` (f5.bigip.net.interface.Interface method), 160  
`exists()` (f5.bigip.net.route.Route method), 163  
`exists()` (f5.bigip.net.route\_domain.Route\_Domain method), 165  
`exists()` (f5.bigip.net.selfip.Selfip method), 168  
`exists()` (f5.bigip.net.tunnels.Gre method), 173  
`exists()` (f5.bigip.net.tunnels.Tunnel method), 171  
`exists()` (f5.bigip.net.tunnels.Vxlan method), 176  
`exists()` (f5.bigip.net.vlan.Interfaces method), 180  
`exists()` (f5.bigip.net.vlan.Vlan method), 178  
`exists()` (f5.bigip.resource.Resource method), 216  
`exists()` (f5.bigip.shared.bigip\_failover\_state.Bigip\_Failover\_State method), 186  
`exists()` (f5.bigip.shared.licensing.Activation method), 188  
`exists()` (f5.bigip.shared.licensing.Registration method), 188  
`exists()` (f5.bigip.sys.application.Aplscript method), 192  
`exists()` (f5.bigip.sys.application.Customstat method), 194  
`exists()` (f5.bigip.sys.application.Service method), 196  
`exists()` (f5.bigip.sys.application.Template method), 198  
`exists()` (f5.bigip.sys.db.Db method), 201  
`exists()` (f5.bigip.sys.failover.Failover method), 202  
`exists()` (f5.bigip.sys.global\_settings.Global\_Settings method), 204  
`exists()` (f5.bigip.sys.ntp.Ntp method), 206  
`exists()` (f5.bigip.sys.ntp.Restrict method), 208  
`exists()` (f5.bigip.sys.performance.All\_Stats method), 210  
`External` (class in f5.bigip.ltm.monitor), 44  
`Externals` (class in f5.bigip.ltm.monitor), 43  
`f5.bigip.ltm` (module), 32  
`f5.bigip.ltm.monitor` (module), 33  
`f5.bigip.ltm.nat` (module), 121  
`f5.bigip.ltm.node` (module), 123  
`f5.bigip.ltm.policy` (module), 125  
`f5.bigip.ltm.pool` (module), 134  
`f5.bigip.ltm.rule` (module), 139  
`f5.bigip.ltm.snat` (module), 142  
`f5.bigip.ltm.snat_translation` (module), 147  
`f5.bigip.ltm.snatpool` (module), 144  
`f5.bigip.ltm.virtual` (module), 149  
`f5.bigip.ltm.virtual_address` (module), 154  
`f5.bigip.mixins` (module), 217  
`f5.bigip.net` (module), 156  
`f5.bigip.net.arp` (module), 157  
`f5.bigip.net.fdb` (module), 181  
`f5.bigip.net.interface` (module), 159  
`f5.bigip.net.route` (module), 161  
`f5.bigip.net.route_domain` (module), 164  
`f5.bigip.net.selfip` (module), 166  
`f5.bigip.net.tunnels` (module), 169  
`f5.bigip.net.vlan` (module), 177  
`f5.bigip.resource` (module), 210  
`f5.bigip.shared` (module), 186  
`f5.bigip.shared.bigip_failover_state` (module), 186  
`f5.bigip.shared.licensing` (module), 187  
`f5.bigip.sys` (module), 189  
`f5.bigip.sys.application` (module), 189  
`f5.bigip.sys.db` (module), 199  
`f5.bigip.sys.failover` (module), 202  
`f5.bigip.sys.folder` (module), 203  
`f5.bigip.sys.global_settings` (module), 204  
`f5.bigip.sys.ntp` (module), 205  
`f5.bigip.sys.performance` (module), 209  
`f5.common` (module), 219  
`f5.common.constants` (module), 218  
`f5.common.iapp_parser` (module), 218  
`f5.sdk_exception` (module), 219  
`F5SDKError`, 219  
`Failover` (class in f5.bigip.sys.failover), 202  
`Fdbs` (class in f5.bigip.net.fdb), 182  
`Firepass` (class in f5.bigip.ltm.monitor), 46  
`Firepass_s` (class in f5.bigip.ltm.monitor), 46  
`Folders` (class in f5.bigip.sys.folder), 203  
`Ftp` (class in f5.bigip.ltm.monitor), 49  
`Ftps` (class in f5.bigip.ltm.monitor), 48

## F

`f5` (module), 219  
`f5.bigip` (module), 20  
`f5.bigip.cm` (module), 22  
`f5.bigip.cm.device` (module), 23  
`f5.bigip.cm.device_group` (module), 25  
`f5.bigip.cm.traffic_group` (module), 29

## G

`Gateway_Icmp` (class in f5.bigip.ltm.monitor), 51  
`Gateway_Icmps` (class in f5.bigip.ltm.monitor), 50  
`GenerationMismatch`, 212  
`get_collection()` (f5.bigip.BigIP method), 21  
`get_collection()` (f5.bigip.cm.Cm method), 22  
`get_collection()` (f5.bigip.cm.device.Devices method), 23



`get_collection()` (f5.bigip.cm.device\_group.Device\_Groups method), 25  
`get_collection()` (f5.bigip.cm.device\_group.Devices\_s method), 27  
`get_collection()` (f5.bigip.cm.traffic\_group.Traffic\_Groups method), 30  
`get_collection()` (f5.bigip.ltm.Ltm method), 32  
`get_collection()` (f5.bigip.ltm.monitor.Diameters method), 39  
`get_collection()` (f5.bigip.ltm.monitor.Dns\_s method), 41  
`get_collection()` (f5.bigip.ltm.monitor.Externals method), 44  
`get_collection()` (f5.bigip.ltm.monitor.Firepass\_s method), 46  
`get_collection()` (f5.bigip.ltm.monitor.Ftps method), 48  
`get_collection()` (f5.bigip.ltm.monitor.Gateway\_Icmps method), 50  
`get_collection()` (f5.bigip.ltm.monitor.Https method), 35  
`get_collection()` (f5.bigip.ltm.monitor.Https\_s method), 37  
`get_collection()` (f5.bigip.ltm.monitor.Icmps method), 52  
`get_collection()` (f5.bigip.ltm.monitor.Imaps method), 55  
`get_collection()` (f5.bigip.ltm.monitor.Inbands method), 57  
`get_collection()` (f5.bigip.ltm.monitor.Ldaps method), 59  
`get_collection()` (f5.bigip.ltm.monitor.Module\_Scores method), 61  
`get_collection()` (f5.bigip.ltm.monitor.Mssqls method), 66  
`get_collection()` (f5.bigip.ltm.monitor.Mysqls method), 63  
`get_collection()` (f5.bigip.ltm.monitor.Nntps method), 68  
`get_collection()` (f5.bigip.ltm.monitor.Nones method), 70  
`get_collection()` (f5.bigip.ltm.monitor.Oracles method), 72  
`get_collection()` (f5.bigip.ltm.monitor.Pop3s method), 74  
`get_collection()` (f5.bigip.ltm.monitor.Postgresqls method), 77  
`get_collection()` (f5.bigip.ltm.monitor.Radius\_Accountings method), 81  
`get_collection()` (f5.bigip.ltm.monitor.Radius\_s method), 79  
`get_collection()` (f5.bigip.ltm.monitor.Real\_Servers method), 83  
`get_collection()` (f5.bigip.ltm.monitor.Rpcs method), 85  
`get_collection()` (f5.bigip.ltm.monitor.Sasps method), 88  
`get_collection()` (f5.bigip.ltm.monitor.Scripteds method), 90  
`get_collection()` (f5.bigip.ltm.monitor.Sips method), 92  
`get_collection()` (f5.bigip.ltm.monitor.Smbs method), 94  
`get_collection()` (f5.bigip.ltm.monitor.Smtps method), 96  
`get_collection()` (f5.bigip.ltm.monitor.Snmp\_Dca\_Bases method), 101  
`get_collection()` (f5.bigip.ltm.monitor.Snmp\_Dcas method), 99  
`get_collection()` (f5.bigip.ltm.monitor.Soaps method), 103  
`get_collection()` (f5.bigip.ltm.monitor.Tcp\_Echos method), 107  
`get_collection()` (f5.bigip.ltm.monitor.Tcp\_Half\_Opens method), 110  
`get_collection()` (f5.bigip.ltm.monitor.Tcps method), 105  
`get_collection()` (f5.bigip.ltm.monitor.Udps method), 112  
`get_collection()` (f5.bigip.ltm.monitor.Virtual\_Locations method), 114  
`get_collection()` (f5.bigip.ltm.monitor.Waps method), 116  
`get_collection()` (f5.bigip.ltm.monitor.Wmis method), 118  
`get_collection()` (f5.bigip.ltm.nat.Nats method), 121  
`get_collection()` (f5.bigip.ltm.node.Nodes method), 123  
`get_collection()` (f5.bigip.ltm.policy.Actions\_s method), 130  
`get_collection()` (f5.bigip.ltm.policy.Conditions\_s method), 132  
`get_collection()` (f5.bigip.ltm.policy.Policys method), 126  
`get_collection()` (f5.bigip.ltm.policy.Rules\_s method), 128  
`get_collection()` (f5.bigip.ltm.pool.Members\_s method), 137  
`get_collection()` (f5.bigip.ltm.pool.Pools method), 135  
`get_collection()` (f5.bigip.ltm.rule.Rules method), 140  
`get_collection()` (f5.bigip.ltm.snat.Snats method), 142  
`get_collection()` (f5.bigip.ltm.snat\_translation.Snat\_Translations method), 147  
`get_collection()` (f5.bigip.ltm.snatpool.Snatpools method), 145  
`get_collection()` (f5.bigip.ltm.virtual.Profiles\_s method), 153  
`get_collection()` (f5.bigip.ltm.virtual.Virtuals method), 150  
`get_collection()` (f5.bigip.ltm.virtual\_address.Virtual\_Address\_s method), 154  
`get_collection()` (f5.bigip.net.arp.Arps method), 157  
`get_collection()` (f5.bigip.net.fdb.Fdbs method), 182  
`get_collection()` (f5.bigip.net.fdb.Tunnels method), 184  
`get_collection()` (f5.bigip.net.fdb.Vlans method), 185  
`get_collection()` (f5.bigip.net.interface.Interfaces method), 160  
`get_collection()` (f5.bigip.net.route.Routes method), 162  
`get_collection()` (f5.bigip.net.route\_domain.Route\_Domains method), 164  
`get_collection()` (f5.bigip.net.selfip.Selfips method), 167  
`get_collection()` (f5.bigip.net.tunnels.Gres method), 172  
`get_collection()` (f5.bigip.net.tunnels.Tunnels method), 170  
`get_collection()` (f5.bigip.net.tunnels.Tunnels\_s method), 169  
`get_collection()` (f5.bigip.net.tunnels.Vxlans method),

175  
get\_collection() (f5.bigip.net.vlan.Interfaces\_s method),  
179  
get\_collection() (f5.bigip.net.vlan.Vlans method), 177  
get\_collection() (f5.bigip.resource.Collection method),  
214  
get\_collection() (f5.bigip.resource.OrganizingCollection  
method), 213  
get\_collection() (f5.bigip.sys.application.Aplscripts  
method), 191  
get\_collection() (f5.bigip.sys.application.Applications  
method), 190  
get\_collection() (f5.bigip.sys.application.Customstats  
method), 193  
get\_collection() (f5.bigip.sys.application.Services  
method), 195  
get\_collection() (f5.bigip.sys.application.Templates  
method), 197  
get\_collection() (f5.bigip.sys.db.Dbs method), 200  
get\_collection() (f5.bigip.sys.folder.Folders method), 203  
get\_collection() (f5.bigip.sys.ntp.Restrict method), 207  
get\_collection() (f5.bigip.sys.performance.Performance  
method), 209  
Global\_Settings (class in f5.bigip.sys.global\_settings),  
204  
Gre (class in f5.bigip.net.tunnels), 173  
Gres (class in f5.bigip.net.tunnels), 172

## H

Http (class in f5.bigip.ltm.monitor), 35  
Https (class in f5.bigip.ltm.monitor), 35  
Https\_s (class in f5.bigip.ltm.monitor), 37

## I

IappParser (class in f5.common.iapp\_parser), 218  
Icmp (class in f5.bigip.ltm.monitor), 53  
Icmps (class in f5.bigip.ltm.monitor), 52  
Imap (class in f5.bigip.ltm.monitor), 55  
Imaps (class in f5.bigip.ltm.monitor), 54  
Inband (class in f5.bigip.ltm.monitor), 57  
Inbands (class in f5.bigip.ltm.monitor), 57  
Interface (class in f5.bigip.net.interface), 160  
Interfaces (class in f5.bigip.net.interface), 159  
Interfaces (class in f5.bigip.net.vlan), 180  
Interfaces\_s (class in f5.bigip.net.vlan), 179  
InvalidForceType, 212  
InvalidResource, 211

## K

KindTypeMismatch, 211

## L

LazyAttributeMixin (class in f5.bigip.mixins), 217

LazyAttributesRequired, 217  
Ldap (class in f5.bigip.ltm.monitor), 60  
Ldaps (class in f5.bigip.ltm.monitor), 59  
Licensing (class in f5.bigip.shared.licensing), 187  
load() (f5.bigip.cm.device.Device method), 24  
load() (f5.bigip.cm.device\_group.Device\_Group  
method), 27  
load() (f5.bigip.cm.device\_group.Devices method), 29  
load() (f5.bigip.cm.traffic\_group.Traffic\_Group method),  
31  
load() (f5.bigip.ltm.monitor.Diameter method), 41  
load() (f5.bigip.ltm.monitor.Dns method), 43  
load() (f5.bigip.ltm.monitor.External method), 45  
load() (f5.bigip.ltm.monitor.Firepass method), 47  
load() (f5.bigip.ltm.monitor.Ftp method), 49  
load() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 52  
load() (f5.bigip.ltm.monitor.Http method), 36  
load() (f5.bigip.ltm.monitor.HttpS method), 38  
load() (f5.bigip.ltm.monitor.Icmp method), 54  
load() (f5.bigip.ltm.monitor.Imap method), 56  
load() (f5.bigip.ltm.monitor.Inband method), 58  
load() (f5.bigip.ltm.monitor.Ldap method), 60  
load() (f5.bigip.ltm.monitor.Module\_Score method), 63  
load() (f5.bigip.ltm.monitor.Mssql method), 67  
load() (f5.bigip.ltm.monitor.Mysql method), 65  
load() (f5.bigip.ltm.monitor.Nntp method), 69  
load() (f5.bigip.ltm.monitor.NONE method), 71  
load() (f5.bigip.ltm.monitor.Oracle method), 74  
load() (f5.bigip.ltm.monitor.Pop3 method), 76  
load() (f5.bigip.ltm.monitor.Postgresql method), 78  
load() (f5.bigip.ltm.monitor.Radius method), 80  
load() (f5.bigip.ltm.monitor.Radius\_Accounting method),  
82  
load() (f5.bigip.ltm.monitor.Real\_Server method), 85  
load() (f5.bigip.ltm.monitor.Rpc method), 87  
load() (f5.bigip.ltm.monitor.Sasp method), 89  
load() (f5.bigip.ltm.monitor.Scripted method), 91  
load() (f5.bigip.ltm.monitor.Sip method), 93  
load() (f5.bigip.ltm.monitor.Smb method), 96  
load() (f5.bigip.ltm.monitor.Smtp method), 98  
load() (f5.bigip.ltm.monitor.Snmp\_Dca method), 100  
load() (f5.bigip.ltm.monitor.Snmp\_Dca\_Base method),  
102  
load() (f5.bigip.ltm.monitor.Soap method), 104  
load() (f5.bigip.ltm.monitor.Tcp method), 107  
load() (f5.bigip.ltm.monitor.Tcp\_Echo method), 109  
load() (f5.bigip.ltm.monitor.Tcp\_Half\_Open method),  
111  
load() (f5.bigip.ltm.monitor.Udp method), 113  
load() (f5.bigip.ltm.monitor.Virtual\_Location method),  
115  
load() (f5.bigip.ltm.monitor.Wap method), 118  
load() (f5.bigip.ltm.monitor.Wmi method), 120  
load() (f5.bigip.ltm.nat.Nat method), 122

load() (f5.bigip.ltm.node.Node method), 125  
 load() (f5.bigip.ltm.policy.Actions method), 131  
 load() (f5.bigip.ltm.policy.Conditions method), 134  
 load() (f5.bigip.ltm.policy.Policy method), 127  
 load() (f5.bigip.ltm.policy.Rules method), 129  
 load() (f5.bigip.ltm.pool.Members method), 139  
 load() (f5.bigip.ltm.pool.Pool method), 136  
 load() (f5.bigip.ltm.rule.Rule method), 141  
 load() (f5.bigip.ltm.snat.Snat method), 144  
 load() (f5.bigip.ltm.snat\_translation.Snat\_Translation method), 148  
 load() (f5.bigip.ltm.snatpool.Snatpool method), 146  
 load() (f5.bigip.ltm.virtual.Profiles method), 152  
 load() (f5.bigip.ltm.virtual.Virtual method), 151  
 load() (f5.bigip.ltm.virtual\_address.Virtual\_Address method), 156  
 load() (f5.bigip.net.arp.Arp method), 158  
 load() (f5.bigip.net.fdb.Tunnel method), 183  
 load() (f5.bigip.net.interface.Interface method), 161  
 load() (f5.bigip.net.route.Route method), 163  
 load() (f5.bigip.net.route\_domain.Route\_Domain method), 165  
 load() (f5.bigip.net.selfip.Selfip method), 168  
 load() (f5.bigip.net.tunnels.Gre method), 174  
 load() (f5.bigip.net.tunnels.Tunnel method), 172  
 load() (f5.bigip.net.tunnels.Vxlan method), 176  
 load() (f5.bigip.net.vlan.Interfaces method), 181  
 load() (f5.bigip.net.vlan.Vlan method), 178  
 load() (f5.bigip.resource.Resource method), 215  
 load() (f5.bigip.sys.application.Aplscript method), 192  
 load() (f5.bigip.sys.application.Customstat method), 194  
 load() (f5.bigip.sys.application.Service method), 197  
 load() (f5.bigip.sys.application.Template method), 199  
 load() (f5.bigip.sys.db.Db method), 201  
 load() (f5.bigip.sys.ntp.Restrict method), 208  
 Ltm (class in f5.bigip.ltm), 32

## M

MalformedTCLListException, 218  
 Members (class in f5.bigip.ltm.pool), 138  
 Members\_s (class in f5.bigip.ltm.pool), 137  
 message (f5.common.iapp\_parser.CurlyBraceMismatchException attribute), 218  
 message (f5.common.iapp\_parser.EmptyTemplateException attribute), 218  
 message (f5.common.iapp\_parser.MalformedTCLListException attribute), 219  
 message (f5.common.iapp\_parser.NonextantSectionException attribute), 218  
 message (f5.common.iapp\_parser.NonextantTemplateNameException attribute), 218  
 MissingRequiredCreationParameter, 212  
 MissingRequiredReadParameter, 212  
 Module\_Score (class in f5.bigip.ltm.monitor), 62

Module\_Scores (class in f5.bigip.ltm.monitor), 61  
 Mssql (class in f5.bigip.ltm.monitor), 66  
 Mssqls (class in f5.bigip.ltm.monitor), 65  
 Mysql (class in f5.bigip.ltm.monitor), 64  
 Mysqls (class in f5.bigip.ltm.monitor), 63

## N

Nat (class in f5.bigip.ltm.nat), 122  
 Nats (class in f5.bigip.ltm.nat), 121  
 Nntp (class in f5.bigip.ltm.monitor), 68  
 Nntps (class in f5.bigip.ltm.monitor), 68  
 Node (class in f5.bigip.ltm.node), 124  
 Nodes (class in f5.bigip.ltm.node), 123  
 NONE (class in f5.bigip.ltm.monitor), 71  
 Nones (class in f5.bigip.ltm.monitor), 70  
 NonextantSectionException, 218  
 NonextantTemplateNameException, 218  
 Ntp (class in f5.bigip.sys.ntp), 205

## O

Oracle (class in f5.bigip.ltm.monitor), 73  
 Oracles (class in f5.bigip.ltm.monitor), 72  
 OrganizingCollection (class in f5.bigip.resource), 213

## P

parse\_template() (f5.common.iapp\_parser.IappParser method), 218  
 PathElement (class in f5.bigip.resource), 212  
 Performance (class in f5.bigip.sys.performance), 209  
 Policy (class in f5.bigip.ltm.policy), 126  
 Policys (class in f5.bigip.ltm.policy), 126  
 Pool (class in f5.bigip.ltm.pool), 135  
 Pools (class in f5.bigip.ltm.pool), 135  
 Pop3 (class in f5.bigip.ltm.monitor), 75  
 Pop3s (class in f5.bigip.ltm.monitor), 74  
 Postgresql (class in f5.bigip.ltm.monitor), 77  
 Postgresqls (class in f5.bigip.ltm.monitor), 76  
 Profiles (class in f5.bigip.ltm.virtual), 151  
 Profiles\_s (class in f5.bigip.ltm.virtual), 153

## R

Radius (class in f5.bigip.ltm.monitor), 79  
 Radius\_Accounting (class in f5.bigip.ltm.monitor), 82  
 Radius\_Accountings (class in f5.bigip.ltm.monitor), 81  
 Radius\_s (class in f5.bigip.ltm.monitor), 79  
 raw (f5.bigip.BigIP attribute), 21  
 raw (f5.bigip.cm.Cm attribute), 22  
 raw (f5.bigip.cm.device.Device attribute), 24  
 raw (f5.bigip.cm.device.Devices attribute), 23  
 raw (f5.bigip.cm.device\_group.Device\_Group attribute), 27  
 raw (f5.bigip.cm.device\_group.Device\_Groups attribute), 25

raw (f5.bigip.cm.device\_group.Devices attribute), 29  
raw (f5.bigip.cm.device\_group.Devices\_s attribute), 28  
raw (f5.bigip.cm.traffic\_group.Traffic\_Group attribute), 31  
raw (f5.bigip.cm.traffic\_group.Traffic\_Groups attribute), 30  
raw (f5.bigip.ltm.Ltm attribute), 32  
raw (f5.bigip.ltm.monitor.Diameter attribute), 41  
raw (f5.bigip.ltm.monitor.Diameters attribute), 39  
raw (f5.bigip.ltm.monitor.Dns attribute), 43  
raw (f5.bigip.ltm.monitor.Dns\_s attribute), 42  
raw (f5.bigip.ltm.monitor.External attribute), 45  
raw (f5.bigip.ltm.monitor.Externals attribute), 44  
raw (f5.bigip.ltm.monitor.Firepass attribute), 47  
raw (f5.bigip.ltm.monitor.Firepass\_s attribute), 46  
raw (f5.bigip.ltm.monitor.Ftp attribute), 50  
raw (f5.bigip.ltm.monitor.Ftps attribute), 48  
raw (f5.bigip.ltm.monitor.Gateway\_Icmp attribute), 52  
raw (f5.bigip.ltm.monitor.Gateway\_Icmps attribute), 50  
raw (f5.bigip.ltm.monitor.Http attribute), 36  
raw (f5.bigip.ltm.monitor.HttpS attribute), 39  
raw (f5.bigip.ltm.monitor.Https attribute), 35  
raw (f5.bigip.ltm.monitor.Https\_s attribute), 37  
raw (f5.bigip.ltm.monitor.Icmp attribute), 54  
raw (f5.bigip.ltm.monitor.Icmps attribute), 53  
raw (f5.bigip.ltm.monitor.Imap attribute), 56  
raw (f5.bigip.ltm.monitor.Imaps attribute), 55  
raw (f5.bigip.ltm.monitor.Inband attribute), 58  
raw (f5.bigip.ltm.monitor.Inbands attribute), 57  
raw (f5.bigip.ltm.monitor.Ldap attribute), 61  
raw (f5.bigip.ltm.monitor.Ldaps attribute), 59  
raw (f5.bigip.ltm.monitor.Module\_Score attribute), 63  
raw (f5.bigip.ltm.monitor.Module\_Scores attribute), 61  
raw (f5.bigip.ltm.monitor.Mssql attribute), 67  
raw (f5.bigip.ltm.monitor.Mssqls attribute), 66  
raw (f5.bigip.ltm.monitor.Mysql attribute), 65  
raw (f5.bigip.ltm.monitor.Mysqls attribute), 64  
raw (f5.bigip.ltm.monitor.Nntp attribute), 69  
raw (f5.bigip.ltm.monitor.Nntps attribute), 68  
raw (f5.bigip.ltm.monitor.NONE attribute), 72  
raw (f5.bigip.ltm.monitor.Nones attribute), 70  
raw (f5.bigip.ltm.monitor.Oracle attribute), 74  
raw (f5.bigip.ltm.monitor.Oracles attribute), 72  
raw (f5.bigip.ltm.monitor.Pop3 attribute), 76  
raw (f5.bigip.ltm.monitor.Pop3s attribute), 75  
raw (f5.bigip.ltm.monitor.Postgresql attribute), 78  
raw (f5.bigip.ltm.monitor.Postgresqls attribute), 77  
raw (f5.bigip.ltm.monitor.Radius attribute), 80  
raw (f5.bigip.ltm.monitor.Radius\_Accounting attribute), 83  
raw (f5.bigip.ltm.monitor.Radius\_Accountings attribute), 81  
raw (f5.bigip.ltm.monitor.Radius\_s attribute), 79  
raw (f5.bigip.ltm.monitor.Real\_Server attribute), 85  
raw (f5.bigip.ltm.monitor.Real\_Servers attribute), 83  
raw (f5.bigip.ltm.monitor.Rpc attribute), 87  
raw (f5.bigip.ltm.monitor.Rpcs attribute), 86  
raw (f5.bigip.ltm.monitor.Sasp attribute), 89  
raw (f5.bigip.ltm.monitor.Sasps attribute), 88  
raw (f5.bigip.ltm.monitor.Scripted attribute), 91  
raw (f5.bigip.ltm.monitor.Scripteds attribute), 90  
raw (f5.bigip.ltm.monitor.Sip attribute), 94  
raw (f5.bigip.ltm.monitor.Sips attribute), 92  
raw (f5.bigip.ltm.monitor.Smb attribute), 96  
raw (f5.bigip.ltm.monitor.Smbfs attribute), 94  
raw (f5.bigip.ltm.monitor.Smtp attribute), 98  
raw (f5.bigip.ltm.monitor.Smtps attribute), 97  
raw (f5.bigip.ltm.monitor.Snmp\_Dca attribute), 100  
raw (f5.bigip.ltm.monitor.Snmp\_Dca\_Base attribute), 102  
raw (f5.bigip.ltm.monitor.Snmp\_Dca\_Bases attribute), 101  
raw (f5.bigip.ltm.monitor.Snmp\_Dcas attribute), 99  
raw (f5.bigip.ltm.monitor.Soap attribute), 105  
raw (f5.bigip.ltm.monitor.Soaps attribute), 103  
raw (f5.bigip.ltm.monitor.Tcp attribute), 107  
raw (f5.bigip.ltm.monitor.Tcp\_Echo attribute), 109  
raw (f5.bigip.ltm.monitor.Tcp\_Echos attribute), 108  
raw (f5.bigip.ltm.monitor.Tcp\_Half\_Open attribute), 111  
raw (f5.bigip.ltm.monitor.Tcp\_Half\_Opens attribute), 110  
raw (f5.bigip.ltm.monitor.Tcps attribute), 105  
raw (f5.bigip.ltm.monitor.Udp attribute), 113  
raw (f5.bigip.ltm.monitor.Udps attribute), 112  
raw (f5.bigip.ltm.monitor.Virtual\_Location attribute), 116  
raw (f5.bigip.ltm.monitor.Virtual\_Locations attribute), 114  
raw (f5.bigip.ltm.monitor.Wap attribute), 118  
raw (f5.bigip.ltm.monitor.Waps attribute), 116  
raw (f5.bigip.ltm.monitor.Wmi attribute), 120  
raw (f5.bigip.ltm.monitor.Wmis attribute), 119  
raw (f5.bigip.ltm.nat.Nat attribute), 122  
raw (f5.bigip.ltm.nat.Nats attribute), 121  
raw (f5.bigip.ltm.node.Node attribute), 125  
raw (f5.bigip.ltm.node.Nodes attribute), 123  
raw (f5.bigip.ltm.policy.Actions attribute), 132  
raw (f5.bigip.ltm.policy.Actions\_s attribute), 130  
raw (f5.bigip.ltm.policy.Conditions attribute), 134  
raw (f5.bigip.ltm.policy.Conditions\_s attribute), 133  
raw (f5.bigip.ltm.policy.Policy attribute), 127  
raw (f5.bigip.ltm.policy.Policys attribute), 126  
raw (f5.bigip.ltm.policy.Rules attribute), 129  
raw (f5.bigip.ltm.policy.Rules\_s attribute), 128  
raw (f5.bigip.ltm.pool.Members attribute), 139  
raw (f5.bigip.ltm.pool.Members\_s attribute), 137  
raw (f5.bigip.ltm.pool.Pool attribute), 136  
raw (f5.bigip.ltm.pool.Pools attribute), 135  
raw (f5.bigip.ltm.rule.Rule attribute), 141



- raw (f5.bigip.ltm.rule.Rules attribute), 140
- raw (f5.bigip.ltm.snat.Snat attribute), 144
- raw (f5.bigip.ltm.snat.Snats attribute), 142
- raw (f5.bigip.ltm.snat\_translation.Snat\_Translation attribute), 149
- raw (f5.bigip.ltm.snat\_translation.Snat\_Translations attribute), 147
- raw (f5.bigip.ltm.snatpool.Snatpool attribute), 146
- raw (f5.bigip.ltm.snatpool.Snatpools attribute), 145
- raw (f5.bigip.ltm.virtual.Profiles attribute), 152
- raw (f5.bigip.ltm.virtual.Profiles\_s attribute), 153
- raw (f5.bigip.ltm.virtual.Virtual attribute), 151
- raw (f5.bigip.ltm.virtual.Virtuals attribute), 150
- raw (f5.bigip.ltm.virtual\_address.Virtual\_Address attribute), 156
- raw (f5.bigip.ltm.virtual\_address.Virtual\_Address\_s attribute), 154
- raw (f5.bigip.net.arp.Arp attribute), 159
- raw (f5.bigip.net.arp.Arps attribute), 157
- raw (f5.bigip.net.fdb.Fdb attribute), 182
- raw (f5.bigip.net.fdb.Tunnel attribute), 183
- raw (f5.bigip.net.fdb.Tunnels attribute), 184
- raw (f5.bigip.net.fdb.Vlans attribute), 185
- raw (f5.bigip.net.interface.Interface attribute), 161
- raw (f5.bigip.net.interface.Interfaces attribute), 160
- raw (f5.bigip.net.route.Route attribute), 163
- raw (f5.bigip.net.route.Routes attribute), 162
- raw (f5.bigip.net.route\_domain.Route\_Domain attribute), 166
- raw (f5.bigip.net.route\_domain.Route\_Domains attribute), 164
- raw (f5.bigip.net.selfip.Selfip attribute), 168
- raw (f5.bigip.net.selfip.Selfips attribute), 167
- raw (f5.bigip.net.tunnels.Gre attribute), 174
- raw (f5.bigip.net.tunnels.Gres attribute), 173
- raw (f5.bigip.net.tunnels.Tunnel attribute), 172
- raw (f5.bigip.net.tunnels.Tunnels attribute), 170
- raw (f5.bigip.net.tunnels.Tunnels\_s attribute), 170
- raw (f5.bigip.net.tunnels.Vxlan attribute), 176
- raw (f5.bigip.net.tunnels.Vxlans attribute), 175
- raw (f5.bigip.net.vlan.Interfaces attribute), 181
- raw (f5.bigip.net.vlan.Interfaces\_s attribute), 180
- raw (f5.bigip.net.vlan.Vlan attribute), 179
- raw (f5.bigip.net.vlan.Vlans attribute), 177
- raw (f5.bigip.resource.Collection attribute), 215
- raw (f5.bigip.resource.OrganizingCollection attribute), 214
- raw (f5.bigip.resource.PathElement attribute), 212
- raw (f5.bigip.resource.Resource attribute), 216
- raw (f5.bigip.resource.ResourceBase attribute), 213
- raw (f5.bigip.shared.bigip\_failover\_state.Bigip\_Failover\_State attribute), 187
- raw (f5.bigip.shared.licensing.Activation attribute), 188
- raw (f5.bigip.shared.licensing.Licensing attribute), 187
- raw (f5.bigip.shared.licensing.Registration attribute), 189
- raw (f5.bigip.sys.application.Aplscript attribute), 192
- raw (f5.bigip.sys.application.Aplscripts attribute), 191
- raw (f5.bigip.sys.application.Applications attribute), 190
- raw (f5.bigip.sys.application.Customstat attribute), 195
- raw (f5.bigip.sys.application.Customstats attribute), 193
- raw (f5.bigip.sys.application.Service attribute), 197
- raw (f5.bigip.sys.application.Services attribute), 195
- raw (f5.bigip.sys.application.Template attribute), 199
- raw (f5.bigip.sys.application.Templates attribute), 198
- raw (f5.bigip.sys.db.Db attribute), 201
- raw (f5.bigip.sys.db.Dbs attribute), 200
- raw (f5.bigip.sys.failover.Failover attribute), 203
- raw (f5.bigip.sys.folder.Folders attribute), 204
- raw (f5.bigip.sys.global\_settings.Global\_Settings attribute), 205
- raw (f5.bigip.sys.ntp.Ntp attribute), 206
- raw (f5.bigip.sys.ntp.Restrict attribute), 208
- raw (f5.bigip.sys.ntp.Restricts attribute), 207
- raw (f5.bigip.sys.performance.All\_Stats attribute), 210
- raw (f5.bigip.sys.performance.Performance attribute), 209
- Real\_Server (class in f5.bigip.ltm.monitor), 84
- Real\_Servers (class in f5.bigip.ltm.monitor), 83
- refresh() (f5.bigip.BigIP method), 21
- refresh() (f5.bigip.cm.Cm method), 22
- refresh() (f5.bigip.cm.device.Device method), 24
- refresh() (f5.bigip.cm.device.Devices method), 23
- refresh() (f5.bigip.cm.device\_group.Device\_Group method), 27
- refresh() (f5.bigip.cm.device\_group.Device\_Groups method), 25
- refresh() (f5.bigip.cm.device\_group.Devices method), 29
- refresh() (f5.bigip.cm.device\_group.Devices\_s method), 28
- refresh() (f5.bigip.cm.traffic\_group.Traffic\_Group method), 31
- refresh() (f5.bigip.cm.traffic\_group.Traffic\_Groups method), 30
- refresh() (f5.bigip.ltm.Ltm method), 33
- refresh() (f5.bigip.ltm.monitor.Diameter method), 41
- refresh() (f5.bigip.ltm.monitor.Diameters method), 40
- refresh() (f5.bigip.ltm.monitor.Dns method), 43
- refresh() (f5.bigip.ltm.monitor.Dns\_s method), 42
- refresh() (f5.bigip.ltm.monitor.External method), 45
- refresh() (f5.bigip.ltm.monitor.Externals method), 44
- refresh() (f5.bigip.ltm.monitor.Firepass method), 47
- refresh() (f5.bigip.ltm.monitor.Firepass\_s method), 46
- refresh() (f5.bigip.ltm.monitor.Ftp method), 50
- refresh() (f5.bigip.ltm.monitor.Ftps method), 48
- refresh() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 52
- refresh() (f5.bigip.ltm.monitor.Gateway\_Icmps method), 51

`refresh()` (`f5.bigip.ltm.monitor.Http` method), 36  
`refresh()` (`f5.bigip.ltm.monitor.HttpS` method), 39  
`refresh()` (`f5.bigip.ltm.monitor.Https` method), 35  
`refresh()` (`f5.bigip.ltm.monitor.Https_s` method), 37  
`refresh()` (`f5.bigip.ltm.monitor.Icmp` method), 54  
`refresh()` (`f5.bigip.ltm.monitor.Icmps` method), 53  
`refresh()` (`f5.bigip.ltm.monitor.Imap` method), 56  
`refresh()` (`f5.bigip.ltm.monitor.Imaps` method), 55  
`refresh()` (`f5.bigip.ltm.monitor.Inband` method), 58  
`refresh()` (`f5.bigip.ltm.monitor.Inbands` method), 57  
`refresh()` (`f5.bigip.ltm.monitor.Ldap` method), 61  
`refresh()` (`f5.bigip.ltm.monitor.Ldaps` method), 59  
`refresh()` (`f5.bigip.ltm.monitor.Module_Score` method), 63  
`refresh()` (`f5.bigip.ltm.monitor.Module_Scores` method), 62  
`refresh()` (`f5.bigip.ltm.monitor.Mssql` method), 67  
`refresh()` (`f5.bigip.ltm.monitor.Mssqls` method), 66  
`refresh()` (`f5.bigip.ltm.monitor.Mysql` method), 65  
`refresh()` (`f5.bigip.ltm.monitor.Mysqls` method), 64  
`refresh()` (`f5.bigip.ltm.monitor.Nntp` method), 69  
`refresh()` (`f5.bigip.ltm.monitor.Nntps` method), 68  
`refresh()` (`f5.bigip.ltm.monitor.NONE` method), 72  
`refresh()` (`f5.bigip.ltm.monitor.Nones` method), 70  
`refresh()` (`f5.bigip.ltm.monitor.Oracle` method), 74  
`refresh()` (`f5.bigip.ltm.monitor.Oracles` method), 73  
`refresh()` (`f5.bigip.ltm.monitor.Pop3` method), 76  
`refresh()` (`f5.bigip.ltm.monitor.Pop3s` method), 75  
`refresh()` (`f5.bigip.ltm.monitor.Postgresql` method), 78  
`refresh()` (`f5.bigip.ltm.monitor.Postgresqls` method), 77  
`refresh()` (`f5.bigip.ltm.monitor.Radius` method), 80  
`refresh()` (`f5.bigip.ltm.monitor.Radius_Accounting` method), 83  
`refresh()` (`f5.bigip.ltm.monitor.Radius_Accountings` method), 81  
`refresh()` (`f5.bigip.ltm.monitor.Radius_s` method), 79  
`refresh()` (`f5.bigip.ltm.monitor.Real_Server` method), 85  
`refresh()` (`f5.bigip.ltm.monitor.Real_Servers` method), 84  
`refresh()` (`f5.bigip.ltm.monitor.Rpc` method), 87  
`refresh()` (`f5.bigip.ltm.monitor.Rpcs` method), 86  
`refresh()` (`f5.bigip.ltm.monitor.Sasp` method), 89  
`refresh()` (`f5.bigip.ltm.monitor.Sasps` method), 88  
`refresh()` (`f5.bigip.ltm.monitor.Scripted` method), 91  
`refresh()` (`f5.bigip.ltm.monitor.Scripteds` method), 90  
`refresh()` (`f5.bigip.ltm.monitor.Sip` method), 94  
`refresh()` (`f5.bigip.ltm.monitor.Sips` method), 92  
`refresh()` (`f5.bigip.ltm.monitor.Smb` method), 96  
`refresh()` (`f5.bigip.ltm.monitor.Smbs` method), 95  
`refresh()` (`f5.bigip.ltm.monitor.Smtp` method), 98  
`refresh()` (`f5.bigip.ltm.monitor.Smtps` method), 97  
`refresh()` (`f5.bigip.ltm.monitor.Snmp_Dca` method), 100  
`refresh()` (`f5.bigip.ltm.monitor.Snmp_Dca_Base` method), 102  
`refresh()` (`f5.bigip.ltm.monitor.Snmp_Dca_Bases` method), 101  
`refresh()` (`f5.bigip.ltm.monitor.Snmp_Dcas` method), 99  
`refresh()` (`f5.bigip.ltm.monitor.Soap` method), 105  
`refresh()` (`f5.bigip.ltm.monitor.Soaps` method), 103  
`refresh()` (`f5.bigip.ltm.monitor.Tcp` method), 107  
`refresh()` (`f5.bigip.ltm.monitor.Tcp_Echo` method), 109  
`refresh()` (`f5.bigip.ltm.monitor.Tcp_Echos` method), 108  
`refresh()` (`f5.bigip.ltm.monitor.Tcp_Half_Open` method), 111  
`refresh()` (`f5.bigip.ltm.monitor.Tcp_Half_Opens` method), 110  
`refresh()` (`f5.bigip.ltm.monitor.Tcps` method), 106  
`refresh()` (`f5.bigip.ltm.monitor.Udp` method), 113  
`refresh()` (`f5.bigip.ltm.monitor.Udps` method), 112  
`refresh()` (`f5.bigip.ltm.monitor.Virtual_Location` method), 116  
`refresh()` (`f5.bigip.ltm.monitor.Virtual_Locations` method), 114  
`refresh()` (`f5.bigip.ltm.monitor.Wap` method), 118  
`refresh()` (`f5.bigip.ltm.monitor.Waps` method), 117  
`refresh()` (`f5.bigip.ltm.monitor.Wmi` method), 120  
`refresh()` (`f5.bigip.ltm.monitor.Wmis` method), 119  
`refresh()` (`f5.bigip.ltm.nat.Nat` method), 123  
`refresh()` (`f5.bigip.ltm.nat.Nats` method), 121  
`refresh()` (`f5.bigip.ltm.node.Node` method), 125  
`refresh()` (`f5.bigip.ltm.node.Nodes` method), 124  
`refresh()` (`f5.bigip.ltm.policy.Actions` method), 132  
`refresh()` (`f5.bigip.ltm.policy.Actions_s` method), 130  
`refresh()` (`f5.bigip.ltm.policy.Conditions` method), 134  
`refresh()` (`f5.bigip.ltm.policy.Conditions_s` method), 133  
`refresh()` (`f5.bigip.ltm.policy.Policy` method), 127  
`refresh()` (`f5.bigip.ltm.policy.Policys` method), 126  
`refresh()` (`f5.bigip.ltm.policy.Rules` method), 130  
`refresh()` (`f5.bigip.ltm.policy.Rules_s` method), 128  
`refresh()` (`f5.bigip.ltm.pool.Members` method), 139  
`refresh()` (`f5.bigip.ltm.pool.Members_s` method), 137  
`refresh()` (`f5.bigip.ltm.pool.Pool` method), 136  
`refresh()` (`f5.bigip.ltm.pool.Pools` method), 135  
`refresh()` (`f5.bigip.ltm.rule.Rule` method), 141  
`refresh()` (`f5.bigip.ltm.rule.Rules` method), 140  
`refresh()` (`f5.bigip.ltm.snat.Snat` method), 144  
`refresh()` (`f5.bigip.ltm.snat.Snats` method), 142  
`refresh()` (`f5.bigip.ltm.snat_translation.Snat_Translation` method), 149  
`refresh()` (`f5.bigip.ltm.snat_translation.Snat_Translations` method), 147  
`refresh()` (`f5.bigip.ltm.snatpool.Snatpool` method), 146  
`refresh()` (`f5.bigip.ltm.snatpool.Snatpools` method), 145  
`refresh()` (`f5.bigip.ltm.virtual.Profiles` method), 153  
`refresh()` (`f5.bigip.ltm.virtual.Profiles_s` method), 153  
`refresh()` (`f5.bigip.ltm.virtual.Virtual` method), 151  
`refresh()` (`f5.bigip.ltm.virtual.Virtuals` method), 150

- `refresh()` (`f5.bigip.ltm.virtual_address.Virtual_Address` method), 156
  - `refresh()` (`f5.bigip.ltm.virtual_address.Virtual_Address_s` method), 155
  - `refresh()` (`f5.bigip.net.arp.Arp` method), 159
  - `refresh()` (`f5.bigip.net.arp.Arps` method), 157
  - `refresh()` (`f5.bigip.net.fdb.Fdb` method), 182
  - `refresh()` (`f5.bigip.net.fdb.Tunnel` method), 183
  - `refresh()` (`f5.bigip.net.fdb.Tunnels` method), 184
  - `refresh()` (`f5.bigip.net.fdb.Vlans` method), 185
  - `refresh()` (`f5.bigip.net.interface.Interface` method), 161
  - `refresh()` (`f5.bigip.net.interface.Interfaces` method), 160
  - `refresh()` (`f5.bigip.net.route.Route` method), 163
  - `refresh()` (`f5.bigip.net.route.Routes` method), 162
  - `refresh()` (`f5.bigip.net.route_domain.Route_Domain` method), 166
  - `refresh()` (`f5.bigip.net.route_domain.Route_Domains` method), 164
  - `refresh()` (`f5.bigip.net.selfip.Selfip` method), 169
  - `refresh()` (`f5.bigip.net.selfip.Selfips` method), 167
  - `refresh()` (`f5.bigip.net.tunnels.Gre` method), 174
  - `refresh()` (`f5.bigip.net.tunnels.Gres` method), 173
  - `refresh()` (`f5.bigip.net.tunnels.Tunnel` method), 172
  - `refresh()` (`f5.bigip.net.tunnels.Tunnels` method), 171
  - `refresh()` (`f5.bigip.net.tunnels.Tunnels_s` method), 170
  - `refresh()` (`f5.bigip.net.tunnels.Vxlan` method), 176
  - `refresh()` (`f5.bigip.net.tunnels.Vxlans` method), 175
  - `refresh()` (`f5.bigip.net.vlan.Interfaces` method), 181
  - `refresh()` (`f5.bigip.net.vlan.Interfaces_s` method), 180
  - `refresh()` (`f5.bigip.net.vlan.Vlan` method), 179
  - `refresh()` (`f5.bigip.net.vlan.Vlans` method), 177
  - `refresh()` (`f5.bigip.resource.Collection` method), 215
  - `refresh()` (`f5.bigip.resource.OrganizingCollection` method), 214
  - `refresh()` (`f5.bigip.resource.Resource` method), 216
  - `refresh()` (`f5.bigip.resource.ResourceBase` method), 213
  - `refresh()` (`f5.bigip.shared.bigip_failover_state.Bigip_Failover_State` method), 187
  - `refresh()` (`f5.bigip.shared.licensing.Activation` method), 188
  - `refresh()` (`f5.bigip.shared.licensing.Registration` method), 189
  - `refresh()` (`f5.bigip.sys.application.Aplscript` method), 192
  - `refresh()` (`f5.bigip.sys.application.Aplscripts` method), 191
  - `refresh()` (`f5.bigip.sys.application.Applications` method), 190
  - `refresh()` (`f5.bigip.sys.application.Customstat` method), 195
  - `refresh()` (`f5.bigip.sys.application.Customstats` method), 193
  - `refresh()` (`f5.bigip.sys.application.Service` method), 197
  - `refresh()` (`f5.bigip.sys.application.Services` method), 196
  - `refresh()` (`f5.bigip.sys.application.Template` method), 199
  - `refresh()` (`f5.bigip.sys.application.Templates` method), 198
  - `refresh()` (`f5.bigip.sys.db.Db` method), 201
  - `refresh()` (`f5.bigip.sys.db.Dbs` method), 200
  - `refresh()` (`f5.bigip.sys.failover.Failover` method), 203
  - `refresh()` (`f5.bigip.sys.folder.Folders` method), 204
  - `refresh()` (`f5.bigip.sys.global_settings.Global_Settings` method), 205
  - `refresh()` (`f5.bigip.sys.ntp.Ntp` method), 206
  - `refresh()` (`f5.bigip.sys.ntp.Restrict` method), 208
  - `refresh()` (`f5.bigip.sys.ntp.Restricts` method), 207
  - `refresh()` (`f5.bigip.sys.performance.All_Stats` method), 210
  - `refresh()` (`f5.bigip.sys.performance.Performance` method), 209
  - `Registration` (class in `f5.bigip.shared.licensing`), 188
  - `Resource` (class in `f5.bigip.resource`), 215
  - `ResourceBase` (class in `f5.bigip.resource`), 212
  - `Restrict` (class in `f5.bigip.sys.ntp`), 207
  - `Restricts` (class in `f5.bigip.sys.ntp`), 206
  - `Route` (class in `f5.bigip.net.route`), 162
  - `Route_Domain` (class in `f5.bigip.net.route_domain`), 165
  - `Route_Domains` (class in `f5.bigip.net.route_domain`), 164
  - `Routes` (class in `f5.bigip.net.route`), 161
  - `Rpc` (class in `f5.bigip.ltm.monitor`), 86
  - `Rpcs` (class in `f5.bigip.ltm.monitor`), 85
  - `Rule` (class in `f5.bigip.ltm.rule`), 140
  - `Rules` (class in `f5.bigip.ltm.policy`), 128
  - `Rules` (class in `f5.bigip.ltm.rule`), 139
  - `Rules_s` (class in `f5.bigip.ltm.policy`), 128
- ## S
- `Sasp` (class in `f5.bigip.ltm.monitor`), 88
  - `Sasps` (class in `f5.bigip.ltm.monitor`), 87
  - `Scripted` (class in `f5.bigip.ltm.monitor`), 90
  - `Scripteds` (class in `f5.bigip.ltm.monitor`), 90
  - `section_map` (`f5.common.iapp_parser.IappParser` attribute), 218
  - `sections_not_required` (`f5.common.iapp_parser.IappParser` attribute), 218
  - `Selfip` (class in `f5.bigip.net.selfip`), 167
  - `Selfips` (class in `f5.bigip.net.selfip`), 166
  - `Service` (class in `f5.bigip.sys.application`), 196
  - `Services` (class in `f5.bigip.sys.application`), 195
  - `Sip` (class in `f5.bigip.ltm.monitor`), 93
  - `Sips` (class in `f5.bigip.ltm.monitor`), 92
  - `Smb` (class in `f5.bigip.ltm.monitor`), 95
  - `Smb`s (class in `f5.bigip.ltm.monitor`), 94
  - `Smt`p (class in `f5.bigip.ltm.monitor`), 97
  - `Smt`ps (class in `f5.bigip.ltm.monitor`), 96
  - `Snat` (class in `f5.bigip.ltm.snat`), 143
  - `Snat_Translation` (class in `f5.bigip.ltm.snat_translation`), 148

Snat\_Translations (class in f5.bigip.ltm.snat\_translation), 147  
Snatpool (class in f5.bigip.ltm.snatpool), 145  
Snatpools (class in f5.bigip.ltm.snatpool), 144  
Snats (class in f5.bigip.ltm.snat), 142  
Snmp\_Dca (class in f5.bigip.ltm.monitor), 99  
Snmp\_Dca\_Base (class in f5.bigip.ltm.monitor), 101  
Snmp\_Dca\_Bases (class in f5.bigip.ltm.monitor), 101  
Snmp\_Dcas (class in f5.bigip.ltm.monitor), 98  
Soap (class in f5.bigip.ltm.monitor), 104  
Soaps (class in f5.bigip.ltm.monitor), 103  
sync() (f5.bigip.cm.Cm method), 22  
sync() (f5.bigip.cm.device\_group.Device\_Group method), 26

## T

tcl\_list\_for\_attr\_re (f5.common.iapp\_parser.IappParser attribute), 218  
tcl\_list\_for\_section\_re (f5.common.iapp\_parser.IappParser attribute), 218  
tcl\_list\_patterns (f5.common.iapp\_parser.IappParser attribute), 218  
Tcp (class in f5.bigip.ltm.monitor), 106  
Tcp\_Echo (class in f5.bigip.ltm.monitor), 108  
Tcp\_Echos (class in f5.bigip.ltm.monitor), 107  
Tcp\_Half\_Open (class in f5.bigip.ltm.monitor), 110  
Tcp\_Half\_Opens (class in f5.bigip.ltm.monitor), 109  
Tcps (class in f5.bigip.ltm.monitor), 105  
Template (class in f5.bigip.sys.application), 198  
template\_attrs (f5.common.iapp\_parser.IappParser attribute), 218  
template\_sections (f5.common.iapp\_parser.IappParser attribute), 218  
Templates (class in f5.bigip.sys.application), 197  
ToDictMixin (class in f5.bigip.mixins), 217  
toggle\_standby() (f5.bigip.sys.failover.Failover method), 202  
Traffic\_Group (class in f5.bigip.cm.traffic\_group), 30  
Traffic\_Groups (class in f5.bigip.cm.traffic\_group), 30  
Tunnel (class in f5.bigip.net.fdb), 182  
Tunnel (class in f5.bigip.net.tunnels), 171  
Tunnels (class in f5.bigip.net.fdb), 184  
Tunnels (class in f5.bigip.net.tunnels), 170  
Tunnels\_s (class in f5.bigip.net.tunnels), 169

## U

Udp (class in f5.bigip.ltm.monitor), 112  
Udps (class in f5.bigip.ltm.monitor), 112  
UnnamedResourceMixin (class in f5.bigip.mixins), 217  
UnregisteredKind, 212  
UnsupportedOperation, 212  
update() (f5.bigip.BigIP method), 21  
update() (f5.bigip.cm.Cm method), 22  
update() (f5.bigip.cm.device.Device method), 25

update() (f5.bigip.cm.device.Devices method), 23  
update() (f5.bigip.cm.device\_group.Device\_Group method), 27  
update() (f5.bigip.cm.device\_group.Device\_Groups method), 26  
update() (f5.bigip.cm.device\_group.Devices method), 29  
update() (f5.bigip.cm.device\_group.Devices\_s method), 28  
update() (f5.bigip.cm.traffic\_group.Traffic\_Group method), 31  
update() (f5.bigip.cm.traffic\_group.Traffic\_Groups method), 30  
update() (f5.bigip.ltm.Ltm method), 33  
update() (f5.bigip.ltm.monitor.Diameter method), 41  
update() (f5.bigip.ltm.monitor.Diameters method), 40  
update() (f5.bigip.ltm.monitor.Dns method), 43  
update() (f5.bigip.ltm.monitor.Dns\_s method), 42  
update() (f5.bigip.ltm.monitor.External method), 45  
update() (f5.bigip.ltm.monitor.Externals method), 44  
update() (f5.bigip.ltm.monitor.Firepass method), 48  
update() (f5.bigip.ltm.monitor.Firepass\_s method), 46  
update() (f5.bigip.ltm.monitor.Ftp method), 50  
update() (f5.bigip.ltm.monitor.Ftps method), 48  
update() (f5.bigip.ltm.monitor.Gateway\_Icmp method), 52  
update() (f5.bigip.ltm.monitor.Gateway\_Icmps method), 51  
update() (f5.bigip.ltm.monitor.Http method), 37  
update() (f5.bigip.ltm.monitor.HttpS method), 39  
update() (f5.bigip.ltm.monitor.Https method), 35  
update() (f5.bigip.ltm.monitor.Https\_s method), 37  
update() (f5.bigip.ltm.monitor.Icmp method), 54  
update() (f5.bigip.ltm.monitor.Icmps method), 53  
update() (f5.bigip.ltm.monitor.Imap method), 56  
update() (f5.bigip.ltm.monitor.Imaps method), 55  
update() (f5.bigip.ltm.monitor.Inband method), 59  
update() (f5.bigip.ltm.monitor.Inbands method), 57  
update() (f5.bigip.ltm.monitor.Ldap method), 61  
update() (f5.bigip.ltm.monitor.Ldaps method), 59  
update() (f5.bigip.ltm.monitor.Module\_Score method), 63  
update() (f5.bigip.ltm.monitor.Module\_Scores method), 62  
update() (f5.bigip.ltm.monitor.Mssql method), 67  
update() (f5.bigip.ltm.monitor.Mssqls method), 66  
update() (f5.bigip.ltm.monitor.Mysql method), 65  
update() (f5.bigip.ltm.monitor.Mysqls method), 64  
update() (f5.bigip.ltm.monitor.Nntp method), 70  
update() (f5.bigip.ltm.monitor.Nntps method), 68  
update() (f5.bigip.ltm.monitor.NONE method), 72  
update() (f5.bigip.ltm.monitor.Nones method), 70  
update() (f5.bigip.ltm.monitor.Oracle method), 74  
update() (f5.bigip.ltm.monitor.Oracles method), 73  
update() (f5.bigip.ltm.monitor.Pop3 method), 76



update() (f5.bigip.ltm.monitor.Pop3s method), 75  
 update() (f5.bigip.ltm.monitor.Postgresql method), 78  
 update() (f5.bigip.ltm.monitor.Postgresqls method), 77  
 update() (f5.bigip.ltm.monitor.Radius method), 81  
 update() (f5.bigip.ltm.monitor.Radius\_Accounting method), 83  
 update() (f5.bigip.ltm.monitor.Radius\_Accountings method), 81  
 update() (f5.bigip.ltm.monitor.Radius\_s method), 79  
 update() (f5.bigip.ltm.monitor.Real\_Server method), 84  
 update() (f5.bigip.ltm.monitor.Real\_Servers method), 84  
 update() (f5.bigip.ltm.monitor.Rpc method), 87  
 update() (f5.bigip.ltm.monitor.Rpcs method), 86  
 update() (f5.bigip.ltm.monitor.Sasp method), 89  
 update() (f5.bigip.ltm.monitor.Sasps method), 88  
 update() (f5.bigip.ltm.monitor.Scripted method), 92  
 update() (f5.bigip.ltm.monitor.Scripteds method), 90  
 update() (f5.bigip.ltm.monitor.Sip method), 94  
 update() (f5.bigip.ltm.monitor.Sips method), 92  
 update() (f5.bigip.ltm.monitor.Smb method), 96  
 update() (f5.bigip.ltm.monitor.Smbs method), 95  
 update() (f5.bigip.ltm.monitor.Smtp method), 98  
 update() (f5.bigip.ltm.monitor.Smtps method), 97  
 update() (f5.bigip.ltm.monitor.Snmp\_Dca method), 100  
 update() (f5.bigip.ltm.monitor.Snmp\_Dca\_Base method), 103  
 update() (f5.bigip.ltm.monitor.Snmp\_Dca\_Bases method), 101  
 update() (f5.bigip.ltm.monitor.Snmp\_Dcas method), 99  
 update() (f5.bigip.ltm.monitor.Soap method), 105  
 update() (f5.bigip.ltm.monitor.Soaps method), 103  
 update() (f5.bigip.ltm.monitor.Tcp method), 107  
 update() (f5.bigip.ltm.monitor.Tcp\_Echo method), 109  
 update() (f5.bigip.ltm.monitor.Tcp\_Echos method), 108  
 update() (f5.bigip.ltm.monitor.Tcp\_Half\_Open method), 111  
 update() (f5.bigip.ltm.monitor.Tcp\_Half\_Opens method), 110  
 update() (f5.bigip.ltm.monitor.Tcps method), 106  
 update() (f5.bigip.ltm.monitor.Udp method), 114  
 update() (f5.bigip.ltm.monitor.Udps method), 112  
 update() (f5.bigip.ltm.monitor.Virtual\_Location method), 116  
 update() (f5.bigip.ltm.monitor.Virtual\_Locations method), 114  
 update() (f5.bigip.ltm.monitor.Wap method), 118  
 update() (f5.bigip.ltm.monitor.Waps method), 117  
 update() (f5.bigip.ltm.monitor.Wmi method), 119  
 update() (f5.bigip.ltm.monitor.Wmis method), 119  
 update() (f5.bigip.ltm.nat.Nats method), 121  
 update() (f5.bigip.ltm.node.Node method), 124  
 update() (f5.bigip.ltm.node.Nodes method), 124  
 update() (f5.bigip.ltm.policy.Actions method), 132  
 update() (f5.bigip.ltm.policy.Actions\_s method), 131  
 update() (f5.bigip.ltm.policy.Conditions method), 134  
 update() (f5.bigip.ltm.policy.Conditions\_s method), 133  
 update() (f5.bigip.ltm.policy.Policy method), 127  
 update() (f5.bigip.ltm.policy.Policys method), 126  
 update() (f5.bigip.ltm.policy.Rules method), 130  
 update() (f5.bigip.ltm.policy.Rules\_s method), 128  
 update() (f5.bigip.ltm.pool.Members method), 138  
 update() (f5.bigip.ltm.pool.Members\_s method), 138  
 update() (f5.bigip.ltm.pool.Pool method), 137  
 update() (f5.bigip.ltm.pool.Pools method), 135  
 update() (f5.bigip.ltm.rule.Rule method), 141  
 update() (f5.bigip.ltm.rule.Rules method), 140  
 update() (f5.bigip.ltm.snat.Snat method), 144  
 update() (f5.bigip.ltm.snat.Snats method), 143  
 update() (f5.bigip.ltm.snat\_translation.Snat\_Translation method), 149  
 update() (f5.bigip.ltm.snat\_translation.Snat\_Translations method), 148  
 update() (f5.bigip.ltm.snatpool.Snatpool method), 146  
 update() (f5.bigip.ltm.snatpool.Snatpools method), 145  
 update() (f5.bigip.ltm.virtual.Profiles method), 153  
 update() (f5.bigip.ltm.virtual.Profiles\_s method), 154  
 update() (f5.bigip.ltm.virtual.Virtual method), 151  
 update() (f5.bigip.ltm.virtual.Virtuals method), 150  
 update() (f5.bigip.ltm.virtual\_address.Virtual\_Address method), 156  
 update() (f5.bigip.ltm.virtual\_address.Virtual\_Address\_s method), 155  
 update() (f5.bigip.net.arp.Arp method), 159  
 update() (f5.bigip.net.arp.Arps method), 158  
 update() (f5.bigip.net.fdb.Fdb method), 182  
 update() (f5.bigip.net.fdb.Tunnel method), 184  
 update() (f5.bigip.net.fdb.Tunnels method), 185  
 update() (f5.bigip.net.fdb.Vlans method), 185  
 update() (f5.bigip.net.interface.Interface method), 161  
 update() (f5.bigip.net.interface.Interfaces method), 160  
 update() (f5.bigip.net.route.Route method), 163  
 update() (f5.bigip.net.route.Routes method), 162  
 update() (f5.bigip.net.route\_domain.Route\_Domain method), 166  
 update() (f5.bigip.net.route\_domain.Route\_Domains method), 165  
 update() (f5.bigip.net.selfip.Selfip method), 169  
 update() (f5.bigip.net.selfip.Selfips method), 167  
 update() (f5.bigip.net.tunnels.Gre method), 174  
 update() (f5.bigip.net.tunnels.Gres method), 173  
 update() (f5.bigip.net.tunnels.Tunnel method), 172  
 update() (f5.bigip.net.tunnels.Tunnels method), 171  
 update() (f5.bigip.net.tunnels.Tunnels\_s method), 170  
 update() (f5.bigip.net.tunnels.Vxlan method), 176  
 update() (f5.bigip.net.tunnels.Vxlans method), 175  
 update() (f5.bigip.net.vlan.Interfaces method), 181  
 update() (f5.bigip.net.vlan.Interfaces\_s method), 180  
 update() (f5.bigip.net.vlan.Vlan method), 179

[update\(\)](#) (f5.bigip.net.vlan.Vlans method), 178  
[update\(\)](#) (f5.bigip.resource.Collection method), 215  
[update\(\)](#) (f5.bigip.resource.OrganizingCollection method), 214  
[update\(\)](#) (f5.bigip.resource.Resource method), 216  
[update\(\)](#) (f5.bigip.resource.ResourceBase method), 213  
[update\(\)](#) (f5.bigip.shared.bigip\_failover\_state.Bigip\_Failover\_State method), 186  
[update\(\)](#) (f5.bigip.shared.licensing.Activation method), 187  
[update\(\)](#) (f5.bigip.shared.licensing.Registration method), 188  
[update\(\)](#) (f5.bigip.sys.application.Aplscript method), 193  
[update\(\)](#) (f5.bigip.sys.application.Aplscripts method), 191  
[update\(\)](#) (f5.bigip.sys.application.Applications method), 191  
[update\(\)](#) (f5.bigip.sys.application.Customstat method), 195  
[update\(\)](#) (f5.bigip.sys.application.Customstats method), 193  
[update\(\)](#) (f5.bigip.sys.application.Service method), 196  
[update\(\)](#) (f5.bigip.sys.application.Services method), 196  
[update\(\)](#) (f5.bigip.sys.application.Template method), 199  
[update\(\)](#) (f5.bigip.sys.application.Templates method), 198  
[update\(\)](#) (f5.bigip.sys.db.Db method), 201  
[update\(\)](#) (f5.bigip.sys.db.Dbs method), 200  
[update\(\)](#) (f5.bigip.sys.failover.Failover method), 202  
[update\(\)](#) (f5.bigip.sys.folder.Folders method), 204  
[update\(\)](#) (f5.bigip.sys.global\_settings.Global\_Settings method), 205  
[update\(\)](#) (f5.bigip.sys.ntp.Ntp method), 206  
[update\(\)](#) (f5.bigip.sys.ntp.Restrict method), 208  
[update\(\)](#) (f5.bigip.sys.ntp.Restricts method), 207  
[update\(\)](#) (f5.bigip.sys.performance.All\_Stats method), 209  
[update\(\)](#) (f5.bigip.sys.performance.Performance method), 209  
[URICreationCollision](#), 212

## V

[Virtual](#) (class in f5.bigip.ltm.virtual), 150  
[Virtual\\_Address](#) (class in f5.bigip.ltm.virtual\_address), 155  
[Virtual\\_Address\\_s](#) (class in f5.bigip.ltm.virtual\_address), 154  
[Virtual\\_Location](#) (class in f5.bigip.ltm.monitor), 115  
[Virtual\\_Locations](#) (class in f5.bigip.ltm.monitor), 114  
[Virtuals](#) (class in f5.bigip.ltm.virtual), 149  
[Vlan](#) (class in f5.bigip.net.vlan), 178  
[Vlans](#) (class in f5.bigip.net.fdb), 185  
[Vlans](#) (class in f5.bigip.net.vlan), 177  
[Vxlan](#) (class in f5.bigip.net.tunnels), 175

[Vxlans](#) (class in f5.bigip.net.tunnels), 174

## W

[Wap](#) (class in f5.bigip.ltm.monitor), 117  
[Waps](#) (class in f5.bigip.ltm.monitor), 116  
[Wmi](#) (class in f5.bigip.ltm.monitor), 119  
[Wmns](#) (class in f5.bigip.ltm.monitor), 118