
NSONE Python SDK Documentation

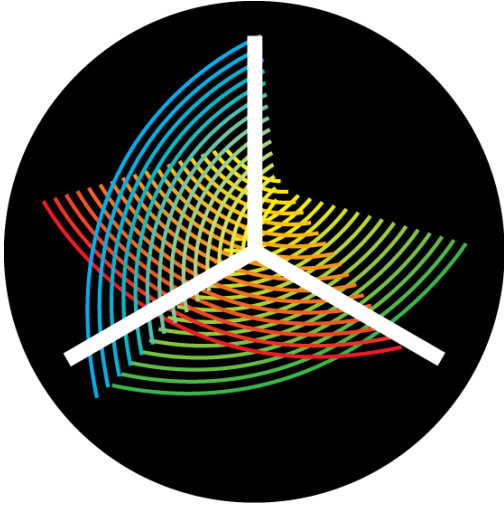
Release 0.9.11

NSONE, Inc.

May 25, 2017

Contents

1	About	3
2	Quick Start	5
3	Contributions	7
4	Contents	9
5	Reference	11
6	Indices and tables	21
	Python Module Index	23



CHAPTER 1

About

This package provides an SDK for accessing the NSONE DNS platform and includes both a simple NSONE REST API wrapper as well as a higher level interface for managing zones, records, data feeds, and more. It supports synchronous and asynchronous transports.

Both python 2.7 and 3.3 are supported.

Install with:

```
$ pip install nsone
```


CHAPTER 2

Quick Start

First, you'll need an API Key. To create one, login to [the portal](#) and click on the Account button in the top right. Select Settings & Users, then add a new API Key at the bottom.

Simple example:

```
from nsone import NSONE

nsone = NSONE(apiKey='qACMD090JXBxT7XOuRs8')
zone = nsone.createZone('example.com', nx_ttl=3600)
print(zone)
record = zone.add_A('honey', ['1.2.3.4', '5.6.7.8'])
print(record)
```

Note that all zone and record changes propagate in real time throughout the NSONE platform.

There are more examples in the [examples directory](#).

CHAPTER 3

Contributions

We welcome contributions! Please [fork on GitHub](#) and submit a Pull Request.

Features

- Extensive config system with support for multiple API keys
- High level interface to Zones and Records
- Low level REST wrapper for all other functionality
- Extendable transport system with synchronous and asynchronous transports

Configuration

Configuring the SDK can be done programmatically and/or via loading (and saving) simple JSON text configuration files. At a minimum, the NSONE API key to access the REST API must be specified.

Loading From a File

By default, configuration is loaded from the file `~/ .nsone`; that is, a file called `.nsone` in the home directory of the user calling the script.

```
# to load an explicit configuration file:
nsone = NSONE(configFile='/etc/nsone/api.json')
```

From an API Key

```
# to generate a configuration based on an api key
nsone = NSONE(apiKey='qACMD09OJXBxT7XOuRs8')
```

JSON File Format

This example shows two different API keys. Which to use can be selected at runtime, see [*nsone.config*](#)

```
{
  "default_key": "account2",
  "verbosity": 5,
  "keys": {
    "account1": {
      "key": "qACMD090JXBxT7XOuRs8",
      "desc": "account number 1",
      "writeLock": true
    },
    "account2": {
      "key": "qACMD090JXBxT7XOwv9v",
      "desc": "account number 2",
      "writeLock": false
    },
  },
  "cli": {
    "output_format": "text"
  }
}
```

More

There are more examples in the [*config.py*](#) example. For the full Config object reference API, see [*nsone.config*](#)

Usage

There are many examples of usage in the [examples](#) directory.

nsone package

nsone.NSONE

This top level object is used to initialize and coordinate access to the NSONE platform. With it, you create objects for accessing either the basic REST interface, or the high level objects such as Zone and Record.

class `nsone.NSONE` (*apiKey=None, config=None, configFile=None, keyID=None*)

Create a new top level NSONE API object

Parameters

- **apiKey** (*str*) – if given, initialize config with this API key (obtainable via creation in NSONE portal)
- **config** (`nsone.config.Config`) – if given, uses a separately constructed and configured Config object
- **configFile** (*str*) – if given, load configuration from the given json configuration file
- **keyID** (*str*) – if given, use the specified key config in the multi-key configuration file

createZone (*zone, zoneFile=None, callback=None, errback=None, **kwargs*)

Create a new zone, and return an associated high level Zone object. Several optional keyword arguments are available to configure the SOA record.

If zoneFile is specified, upload the specific zone definition file to populate the zone with.

Parameters

- **zone** (*str*) – zone name, like ‘example.com’
- **zoneFile** (*str*) – absolute path of a zone file
- **retry** (*int*) – retry time
- **refresh** (*int*) – refresh ttl

- **expiry** (*int*) – expiry ttl
- **nx_ttl** (*int*) – nxdomain TTL

Return type *nsone.zones.Zone*

datafeed ()

Return a new raw REST interface to feed resources

Return type *nsone.rest.data.Feed*

datasource ()

Return a new raw REST interface to datasource resources

Return type *nsone.rest.data.Source*

loadRecord (*domain*, *type*, *zone=None*, *callback=None*, *errback=None*, ***kwargs*)

Load an existing record into a high level Record object.

Parameters

- **domain** (*str*) – domain name of the record in the zone, for example ‘myrecord’. You may leave off the zone, since it must be specified in the zone parameter
- **type** (*str*) – record type, such as ‘A’, ‘MX’, ‘AAAA’, etc.
- **zone** (*str*) – zone name, like ‘example.com’

Return type *nsone.records*

loadZone (*zone*, *callback=None*, *errback=None*)

Load an existing zone into a high level Zone object.

Parameters **zone** (*str*) – zone name, like ‘example.com’

Return type *nsone.zones.Zone*

monitors ()

Return a new raw REST interface to monitors resources

Return type *nsone.rest.monitoring.Monitors*

plan ()

Return a new raw REST interface to account plan

Return type *nsone.rest.account.Plan*

records ()

Return a new raw REST interface to record resources

Return type *nsone.rest.records.Records*

stats ()

Return a new raw REST interface to stats resources

Return type *nsone.rest.stats.Stats*

zones ()

Return a new raw REST interface to zone resources

Return type *nsone.rest.zones.Zones*

nsone.config

This object is used to configure the SDK and REST client. It handles multiple API keys via a simple selection mechanism (keyID).

Sample:

```
{
  "default_key": "account2",
  "verbosity": 5,
  "keys": {
    "account1": {
      "key": "qACMD090JXBxT7XOuRs8",
      "desc": "account number 1",
      "writeLock": true
    },
    "account2": {
      "key": "qACMD090JXBxT7XOwv9v",
      "desc": "account number 2",
      "writeLock": false
    }
  },
  "cli": {
    "output_format": "text"
  }
}
```

class nsone.config.Config (path=None)

A simple object for accessing and manipulating config files. These contains options and credentials for accessing the NSONE REST API. Config files are simple JSON text files. To set or retrieve vales, use the object like a dict.

Parameters path (*str*) – optional path. if given, try to load the given config file

API_VERSION = 'v1'

DEFAULT_CONFIG_FILE = '~/nsone'

ENDPOINT = 'api.nsone.net'

PORT = 443

createFromAPIKey (apikey, maybeWriteDefault=False)

Create a basic config from a single API key

Parameters

- **apikey** (*str*) – NSONE API Key, as created in the NSONE portal
- **maybeWriteDefault** (*bool*) – If True and DEFAULT_CONFIG_FILE doesn't exist write out the resulting config there.

get (item, default=None)

Retrieve a value from the config object.

Parameters

- **item** (*str*) – Key to lookup
- **default** – Default value to return if the requested item doesn't exist

Returns Requested value, or *default* if it didn't exist

getAPIKey (*keyID=None*)

Retrieve the NSONE API Key for the given keyID

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns API Key for the given keyID

getCurrentKeyID ()

Retrieve the current keyID in use.

Returns current keyID in use

getEndpoint ()

Retrieve the NSONE API Endpoint URL that will be used for requests.

Returns URL of the NSONE API that will be used for requests

getKeyConfig (*keyID=None*)

Get key configuration specified by *keyID*, or current keyID.

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns a dict of the request (or current) key config

isKeyWriteLocked (*keyID=None*)

Determine if a key config is write locked.

Parameters **keyID** (*str*) – optional keyID to retrieve, or current if not passed

Returns True if the given (or current) keyID is writeLocked

loadFromDict (*d*)

Load config data from the given dictionary

Parameters **d** (*dict*) – Python dictionary containing configuration items

loadFromFile (*path*)

Load JSON config file from disk at the given path

Parameters **path** (*str*) – path to config file

loadFromString (*body*)

Load config data (i.e. JSON text) from the given string

Parameters **body** (*str*) – config data in JSON format

useKeyID (*keyID*)

Use the given API key config specified by *keyID* during subsequent API calls

Parameters **keyID** (*str*) – an index into the ‘keys’ maintained in this config

write (*path=None*)

Write config data to disk. If this config object already has a path, it will write to it. If it doesn’t, one must be passed during this call.

Parameters **path** (*str*) – path to config file

exception `nsone.config.ConfigException`

Bases: `exceptions.Exception`

nsone.zones

Object representing a single DNS zone.

Note: Answers to a record (the *answers* kwarg) should be passed as one of the following four structures, depending on how advanced the configuration for the answer needs to be:

1. A single string that is coerced to a single answer with no other fields e.g. meta. For example: "1.1.1.1"
2. A list of single strings that is coerced to several answers with no other fields e.g. meta. For example: ["1.1.1.1", "2.2.2.2"]
3. A list of lists. In this case there will be as many answers as are in the outer list, and the answers themselves are used verbatim from the inner list (e.g. may have MX style [10, '1.1.1.1']), but no other fields e.g. meta. You must use this form for MX records, and if there is only one answer it still must be wrapped in an outer list.
4. A list of dicts. In this case it expects the full rest model and passes it along unchanged. You must use this form for any advanced record config like meta data or data feeds.

```
# Example of an advanced answer configuration (list of dicts)
record = yield zone.add_A('record',
    [{ 'answer': ['1.1.1.1'],
      'meta': {
        'up': False
      }
    },
    { 'answer': ['9.9.9.9'],
      'meta': {
        'up': True
      }
    }
  ],
  filters=[{'up': {}}])
```

class nsone.zones.Zone (config, zone)

Bases: object

High level object representing a Zone. In addition to the documented methods, there are magic methods allowing easy creation of records in this zone. Simply can 'add_TYPE' where TYPE is a valid DNS record type, such as add_A(). See examples for more information.

Create a new high level Zone object

Parameters

- **config** (nsone.config.Config) – config object
- **zone** (str) – zone name

cloneRecord (existing_domain, new_domain, rtype, zone=None, callback=None, errback=None)

Clone the given record to a new record such that their configs are identical.

Parameters

- **existing_domain** (str) – The existing record to clone
- **new_domain** (str) – The name of the new cloned record
- **rtype** (str) – DNS record type
- **zone** (str) – Optional zone name, if the new record should exist in a different zone than the original record.

Return type *nsone.records.Record*

Returns new Record

create (*zoneFile=None, callback=None, errback=None, **kwargs*)

Create a new zone. Pass a list of keywords and their values to configure. For the list of keywords available for zone configuration, see *nsone.rest.zones.Zones.INT_FIELDS* and *nsone.rest.zones.Zones.PASSTHRU_FIELDS*. If *zoneFile* is passed, it should be a zone text file on the local disk that will be used to populate the created zone file.

createLinkToSelf (*new_zone, callback=None, errback=None, **kwargs*)

Create a new linked zone, linking to ourselves. All records in this zone will then be available as “linked records” in the new zone.

Parameters *new_zone* (*str*) – the new zone name to link to this one

Returns new Zone

delete (*callback=None, errback=None*)

Delete the zone and ALL records it contains.

linkRecord (*existing_domain, new_domain, rtype, callback=None, errback=None, **kwargs*)

Create a new linked record in this zone. These records use the configuration (answers, ttl, filters, etc) from an existing record in the NSONE platform.

Parameters

- **existing_domain** (*str*) – FQDN of the target record whose config should be used. Does not have to be in the same zone.
- **new_domain** (*str*) – Name of the new (linked) record. Zone name is appended automatically.
- **rtype** (*str*) – DNS record type, which must match the target record.

Return type *nsone.records.Record*

Returns new Record

load (*callback=None, errback=None, reload=False*)

Load zone data from the API.

loadRecord (*domain, rtype, callback=None, errback=None*)

Load a high level Record object from a domain within this Zone.

Parameters

- **domain** (*str*) – The name of the record to load
- **rtype** (*str*) – The DNS record type

Return type *nsone.records.Record*

Returns new Record

qps (*callback=None, errback=None*)

Return the current QPS for this zone

Return type *dict*

Returns QPS information

reload (*callback=None, errback=None*)

Reload zone data from the API.

update (*callback=None, errback=None, **kwargs*)

Update zone configuration. Pass a list of keywords and their values to update. For the list of keywords available for zone configuration, see `nsone.rest.zones.Zones.INT_FIELDS` and `nsone.rest.zones.Zones.PASSTHRU_FIELDS`

usage (*callback=None, errback=None, **kwargs*)

Return the current usage information for this zone

Return type `dict`

Returns usage information

exception `nsone.zones.ZoneException`

Bases: `exceptions.Exception`

nsone.records

Object representing a single DNS record in a zone of a specific type.

Note: Answers to a record (the *answers* kwarg) should be passed as one of the following four structures, depending on how advanced the configuration for the answer needs to be:

1. A single string that is coerced to a single answer with no other fields e.g. meta. For example: `"1.1.1.1"`
2. A list of single strings that is coerced to several answers with no other fields e.g. meta. For example: `["1.1.1.1", "2.2.2.2"]`
3. A list of lists. In this case there will be as many answers as are in the outer list, and the answers themselves are used verbatim from the inner list (e.g. may have MX style `[10, '1.1.1.1']`), but no other fields e.g. meta. You must use this form for MX records, and if there is only one answer it still must be wrapped in an outer list.
4. A list of dicts. In this case it expects the full rest model and passes it along unchanged. You must use this form for any advanced record config like meta data or data feeds.

```
# Example of an advanced answer configuration (list of dicts)
record = yield zone.add_A('record',
    [{ 'answer': ['1.1.1.1'],
      'meta': {
        'up': False
      }
    },
    { 'answer': ['9.9.9.9'],
      'meta': {
        'up': True
      }
    }
  ],
  filters=[{ 'up': {} }])
```

class `nsone.records.Record` (*parentZone, domain, type*)

Bases: `object`

High level object representing a Record

Create a new high level Record

Parameters

- **parentZone** (`nsone.zones.Zone`) – the high level Zone parent object

- **domain** (*str*) – full domain name this record represents. if the domain does not end with the zone name, it is appended.
- **type** (*str*) – The DNS record type (A, MX, etc)

addAnswers (*answers*, *callback=None*, *errback=None*, ***kwargs*)
Add answers to the record.

Parameters **answers** – answers structure. See the class note on answer format.

create (*callback=None*, *errback=None*, ***kwargs*)
Create new record. Pass a list of keywords and their values to config. For the list of keywords available for zone configuration, see `nsone.rest.records.Records.INT_FIELDS`, `nsone.rest.records.Records.PASSTHRU_FIELDS`, `nsone.rest.records.Records.BOOL_FIELDS`

delete (*callback=None*, *errback=None*)
Delete the record from the zone, including all advanced configuration, meta data, etc.

load (*callback=None*, *errback=None*, *reload=False*)
Load record data from the API.

qps (*callback=None*, *errback=None*)
Return the current QPS for this record

Return type `dict`

Returns QPS information

reload (*callback=None*, *errback=None*)
Reload record data from the API.

update (*callback=None*, *errback=None*, ***kwargs*)
Update record configuration. Pass list of keywords and their values to update. For the list of keywords available for zone configuration, see `nsone.rest.records.Records.INT_FIELDS`, `nsone.rest.records.Records.PASSTHRU_FIELDS`, `nsone.rest.records.Records.BOOL_FIELDS`

usage (*callback=None*, *errback=None*, ***kwargs*)
Return the current usage information for this record

Return type `dict`

Returns usage information

exception `nsone.records.RecordException`
Bases: `exceptions.Exception`

nsone.rest

A thin layer over the NSONE REST API

exception `nsone.rest.errors.AuthException` (*message*, *response=None*, *body=None*)
Bases: `nsone.rest.errors.ResourceException`

exception `nsone.rest.errors.RateLimitException` (*message*, *response=None*, *body=None*)
Bases: `nsone.rest.errors.ResourceException`

exception `nsone.rest.errors.ResourceException` (*message*, *response=None*, *body=None*)
Bases: `exceptions.Exception`

class `nsone.rest.resource.BaseResource` (*config*)

Parameters **config** (`nsone.config.Config`) – config object used to build requests

```

    BOOL_FIELDS = []
    DEFAULT_TRANSPORT = 'requests'
    INT_FIELDS = []
    PASSTHRU_FIELDS = []

class nsone.rest.data.Feed(config)
    Bases: nsone.rest.resource.BaseResource

        Parameters config (nsone.config.Config) – config object used to build requests
    PASSTHRU_FIELDS = ['name', 'config']
    ROOT = 'data/feeds'

    create (sourceid, name, config, callback=None, errback=None, **kwargs)
    delete (sourceid, feedid, callback=None, errback=None)
    list (sourceid, callback=None, errback=None)
    retrieve (sourceid, feedid, callback=None, errback=None)
    update (sourceid, feedid, callback=None, errback=None, **kwargs)

class nsone.rest.data.Source(config)
    Bases: nsone.rest.resource.BaseResource

        Parameters config (nsone.config.Config) – config object used to build requests
    PASSTHRU_FIELDS = ['name', 'config']
    ROOT = 'data/sources'

    create (name, sourcetype, callback=None, errback=None, **kwargs)
    delete (sourceid, callback=None, errback=None)
    list (callback=None, errback=None)
    publish (sourceid, data, callback=None, errback=None)
    retrieve (sourceid, callback=None, errback=None)
    update (sourceid, callback=None, errback=None, **kwargs)

class nsone.rest.stats.Stats(config)
    Bases: nsone.rest.resource.BaseResource

        Parameters config (nsone.config.Config) – config object used to build requests
    ROOT = 'stats'

    qps (zone=None, domain=None, type=None, callback=None, errback=None)
    usage (zone=None, domain=None, type=None, callback=None, errback=None, **kwargs)

class nsone.rest.records.Records(config)
    Bases: nsone.rest.resource.BaseResource

        Parameters config (nsone.config.Config) – config object used to build requests
    BOOL_FIELDS = ['use_csubnet', 'override_ttl']
    INT_FIELDS = ['ttl']
    PASSTHRU_FIELDS = ['networks', 'meta', 'regions', 'link']

```

```

ROOT = 'zones'

create (zone, domain, type, callback=None, errback=None, **kwargs)
create_raw (zone, domain, type, body, callback=None, errback=None, **kwargs)
delete (zone, domain, type, callback=None, errback=None)
retrieve (zone, domain, type, callback=None, errback=None)
update (zone, domain, type, callback=None, errback=None, **kwargs)

class nsone.rest.zones.Zones (config)
    Bases: nsone.rest.resource.BaseResource

        Parameters config (nsone.config.Config) – config object used to build requests

    INT_FIELDS = ['retry', 'refresh', 'expiry', 'nx_ttl']
    PASSTHRU_FIELDS = ['secondary', 'hostmaster', 'meta', 'networks', 'link']
    ROOT = 'zones'

    create (zone, callback=None, errback=None, **kwargs)
    delete (zone, callback=None, errback=None)
    import_file (zone, zoneFile, callback=None, errback=None, **kwargs)
    list (callback=None, errback=None)
    retrieve (zone, callback=None, errback=None)
    update (zone, callback=None, errback=None, **kwargs)

```


CHAPTER 6

Indices and tables

- `genindex`
- `modindex`
- `search`

n

- `nsone`, [11](#)
- `nsone.config`, [13](#)
- `nsone.records`, [17](#)
- `nsone.rest.data`, [19](#)
- `nsone.rest.errors`, [18](#)
- `nsone.rest.records`, [19](#)
- `nsone.rest.resource`, [18](#)
- `nsone.rest.stats`, [19](#)
- `nsone.rest.zones`, [20](#)
- `nsone.zones`, [15](#)

A

addAnswers() (nsone.records.Record method), 18
API_VERSION (nsone.config.Config attribute), 13
AuthException, 18

B

BaseResource (class in nsone.rest.resource), 18
BOOL_FIELDS (nsone.rest.records.Records attribute), 19
BOOL_FIELDS (nsone.rest.resource.BaseResource attribute), 18

C

cloneRecord() (nsone.zones.Zone method), 15
Config (class in nsone.config), 13
ConfigException, 14
create() (nsone.records.Record method), 18
create() (nsone.rest.data.Feed method), 19
create() (nsone.rest.data.Source method), 19
create() (nsone.rest.records.Records method), 20
create() (nsone.rest.zones.Zones method), 20
create() (nsone.zones.Zone method), 16
create_raw() (nsone.rest.records.Records method), 20
createFromAPIKey() (nsone.config.Config method), 13
createLinkToSelf() (nsone.zones.Zone method), 16
createZone() (nsone.NSONE method), 11

D

datafeed() (nsone.NSONE method), 12
datasource() (nsone.NSONE method), 12
DEFAULT_CONFIG_FILE (nsone.config.Config attribute), 13
DEFAULT_TRANSPORT (nsone.rest.resource.BaseResource attribute), 19
delete() (nsone.records.Record method), 18
delete() (nsone.rest.data.Feed method), 19
delete() (nsone.rest.data.Source method), 19
delete() (nsone.rest.records.Records method), 20

delete() (nsone.rest.zones.Zones method), 20
delete() (nsone.zones.Zone method), 16

E

ENDPOINT (nsone.config.Config attribute), 13

F

Feed (class in nsone.rest.data), 19

G

get() (nsone.config.Config method), 13
getAPIKey() (nsone.config.Config method), 13
getCurrentKeyID() (nsone.config.Config method), 14
getEndpoint() (nsone.config.Config method), 14
getKeyConfig() (nsone.config.Config method), 14

I

import_file() (nsone.rest.zones.Zones method), 20
INT_FIELDS (nsone.rest.records.Records attribute), 19
INT_FIELDS (nsone.rest.resource.BaseResource attribute), 19
INT_FIELDS (nsone.rest.zones.Zones attribute), 20
isKeyWriteLocked() (nsone.config.Config method), 14

L

linkRecord() (nsone.zones.Zone method), 16
list() (nsone.rest.data.Feed method), 19
list() (nsone.rest.data.Source method), 19
list() (nsone.rest.zones.Zones method), 20
load() (nsone.records.Record method), 18
load() (nsone.zones.Zone method), 16
loadFromDict() (nsone.config.Config method), 14
loadFromFile() (nsone.config.Config method), 14
loadFromString() (nsone.config.Config method), 14
loadRecord() (nsone.NSONE method), 12
loadRecord() (nsone.zones.Zone method), 16
loadZone() (nsone.NSONE method), 12

M

monitors() (nsone.NSONE method), 12

N

NSONE (class in nsone), 11
 nsone (module), 11
 nsone.config (module), 13
 nsone.records (module), 17
 nsone.rest.data (module), 19
 nsone.rest.errors (module), 18
 nsone.rest.records (module), 19
 nsone.rest.resource (module), 18
 nsone.rest.stats (module), 19
 nsone.rest.zones (module), 20
 nsone.zones (module), 15

P

PASSTHRU_FIELDS (nsone.rest.data.Feed attribute), 19
 PASSTHRU_FIELDS (nsone.rest.data.Source attribute), 19
 PASSTHRU_FIELDS (nsone.rest.records.Records attribute), 19
 PASSTHRU_FIELDS (nsone.rest.resource.BaseResource attribute), 19
 PASSTHRU_FIELDS (nsone.rest.zones.Zones attribute), 20
 plan() (nsone.NSONE method), 12
 PORT (nsone.config.Config attribute), 13
 publish() (nsone.rest.data.Source method), 19

Q

qps() (nsone.records.Record method), 18
 qps() (nsone.rest.stats.Stats method), 19
 qps() (nsone.zones.Zone method), 16

R

RateLimitException, 18
 Record (class in nsone.records), 17
 RecordException, 18
 Records (class in nsone.rest.records), 19
 records() (nsone.NSONE method), 12
 reload() (nsone.records.Record method), 18
 reload() (nsone.zones.Zone method), 16
 ResourceException, 18
 retrieve() (nsone.rest.data.Feed method), 19
 retrieve() (nsone.rest.data.Source method), 19
 retrieve() (nsone.rest.records.Records method), 20
 retrieve() (nsone.rest.zones.Zones method), 20
 ROOT (nsone.rest.data.Feed attribute), 19
 ROOT (nsone.rest.data.Source attribute), 19
 ROOT (nsone.rest.records.Records attribute), 19
 ROOT (nsone.rest.stats.Stats attribute), 19
 ROOT (nsone.rest.zones.Zones attribute), 20

S

Source (class in nsone.rest.data), 19

Stats (class in nsone.rest.stats), 19
 stats() (nsone.NSONE method), 12

U

update() (nsone.records.Record method), 18
 update() (nsone.rest.data.Feed method), 19
 update() (nsone.rest.data.Source method), 19
 update() (nsone.rest.records.Records method), 20
 update() (nsone.rest.zones.Zones method), 20
 update() (nsone.zones.Zone method), 16
 usage() (nsone.records.Record method), 18
 usage() (nsone.rest.stats.Stats method), 19
 usage() (nsone.zones.Zone method), 17
 useKeyID() (nsone.config.Config method), 14

W

write() (nsone.config.Config method), 14

Z

Zone (class in nsone.zones), 15
 ZoneException, 17
 Zones (class in nsone.rest.zones), 20
 zones() (nsone.NSONE method), 12