Nflex Connector Utils Documentation

Release v0.2.0+0.gf653984.dirty

Nflex Connector Utils

Oct 04, 2018

Contents

Contents:

CHAPTER 1

Introduction

This python package provides a suite of tools to produce data structures when implementing NTT CMP resources connector nflex modules.

An object oriented interface is provided with `serialize` methods to convert the data into the format required by the NTT nflex connector resources API.

For example:

```
from nflex_connector_utils import Server, serialize_list


def get(event, context):
    return serialize_list([
        Server(id='server-1', name='Server 1'),
        Server(id='server-2', name='Server 2'),
    ])
```

See the *Examples* and *API* documentation for more details.

CHAPTER 2

Installation

Install it with:

```
pip install nflex-connector-utils
```

CHAPTER 3

API

nflex_connector_utils.rfc3339.**convert_datetime**(*dt*)

Convert a datetime or timestamp string to an RFC 3339 timestamp. A `None` is returned as a `None`.

Parameters **dt** (*str, datetime or None*) – Input. This can be a string timestamp to be parsed, a datetime object or a `None`.

Returns A RFC 3339 timestamp

Return type str

nflex_connector_utils.tools.**serialize_list**(*data*)

Convert all objects in a list to the CMP data structure by calling the `serialize` method on each element.

Parameters **data** (*list*) – A list of objects to be serialized

Returns A list of dicts

Return type list

nflex_connector_utils.tools.**vcr_cassette_context**(*function*)

Enclose decorated function into vcr.use_cassette context.

Parameters **function** (*function*) – inner function to enclose

Returns wrapper of inner function

Return type function

nflex_connector_utils.tasks.**set_task_percentage**(*context*, *task_id*, *percentage*)

Update the percentage of a running CMP account sync task

Parameters

- **task_id** (*str*) – The CMP uuid identifying the task. This can be found in the `task_id` key in the event.
- **percentage** (*int*) – Percentage

class nflex_connector_utils.locations.**Locations**(*locations=None*)

A representation of a list of locations. Flexible in terms of what parts can be set based on the spec defined for locations.

Parameters

- **locations** (*list*) – list of location dicts with keys as specified below:
- **location** (*dict*) – keys id, name, latitude, longitude - optional
- **city** (*dict*) – keys id, name, latitude, longitude - optional
- **state** (*dict*) – keys id, name, latitude, longitude - optional
- **country** (*dict*) – keys id, name, latitude, longitude - optional
- **region** (*dict*) – keys id, name, latitude, longitude - optional
- **type** (*str*) – one of 'customer' or 'provider' - optional

Requirement for each lower level dict is: id (str) name (str) - one of id or name is required latitude (float) - optional longitude (float) - optional

Example

This shows how to associate a paris location:

```
locations = Locations([{
    'country': {'name': 'France'},
    'city': {'name': 'Paris'},
    'location': {
        'name': 'Legendary Paris Place',
        'latitude': 48.860764,
        'longitude': 2.393646,
    }
}])

server = Server(
    id='server-2',
    name='Server with lots of details',
    locations=locations
)
```

serialize()

Serialize the contents

class nflex_connector_utils.locations.**Region**(*id=None*)

A representation of a region. When using this, the `id` is looked up in CMP and associated with a name and optional geographical data.

Parameters **id** (*str*) – id

serialize()

Serialize the contents

class nflex_connector_utils.connections.**Connections**(*type=None*, *connections=None*, *appliances=None*, *servers=None*, *networks=None*, *volumes=None*)

A representation of inter-resource associations.

Parameters

- **type** (*str*) – The type of resource to add connections to, e.g. `server`
- **connections** (*list*) – When used together with `type`, a list of associated resource ids or dicts
- **appliances** (*list*) – an optional list of appliance ids
- **servers** (*list*) – an optional list of server ids
- **networks** (*list*) – an optional list of network ids
- **volumes** (*list*) – an optional list of volume ids

Examples

Create a bunch of connections to a server:

```
Connections(type='server', connections=['server-1', 'server-2'])
```

Create a connection to two volumes and a network:

```
Connections(networks=['network-1'], volumes=['volume-1', 'volume-2'])
```

Incrementally add connections:

```
c = Connections()
c.add(type='appliances', connections='appliance-1').add(servers=['server-1'])
c.add(networks=['network-1'])
```

add(*type=None*, *connections=None*, *appliances=None*, *servers=None*, *networks=None*, *volumes=None*)
Add a connection

Parameters

- **type** (*str*) – The type of resource to add connections to, e.g. `server`
- **connections** (*str*) – When used together with `type`, a list of associated resource ids
- **appliances** (*str*) – an optional list of appliance ids
- **servers** (*str*) – an optional list of server ids
- **networks** (*str*) – an optional list of network ids
- **volumes** (*str*) – an optional list of volume ids

Returns returns itself, so that `add` methods can be chained

Return type *nflex_connector_utils.connections.Connections*

serialize()
Serialize the contents

class nflex_connector_utils.image_detail.**ImageDetail**(*id=None*, *name=None*, *type=None*, *distribution=None*, *version=None*, *architecture=None*)
A representation of server image details. This is typically the build image or template used to deploy a server. It contains information about the OS, architecture etc.

Parameters

- **id** (*str*) – id, e.g. "ami-abcdef"
- **name** (*str*) – name, e.g. "Ubuntu Linux 16.04 LTS 64 bit"
- **type** (*str*) – `Windows` or `Linux`
- **distribution** (*str*) – distribution or subtype, e.g. `Server 2012` or `CentOS`
- **version** (*str*) – version e.g. `V2` or `16.04.1`
- **architecture** (*str*) – architecture e.g `i386`, `x86_64`

serialize()
Serialize the contents

class nflex_connector_utils.image_detail.**ImageDetailMap**(*mapping=None*)
A utility class that is able to look up common server images used for several servers using a dict mapping ids to the details.

Parameters **mapping** (*dict*) – A dict with str keys and tuple values

Example

This shows initializing a mapping and looking up images:

```
m = ImageDetailMap([
    'ubuntu16': ('Ubuntu Linux 16.04 LTS 64 bit', 'Linux', 'Ubuntu', '16.04', 'x64
↪'),
    'w2012R2': ('Windows Server 2012R2 Standard', 'Windows', 'Server 2012', 'R2',
↪'x64'),
])

m.get('no-match')                          # Returns an Image with None
m.get('ubuntu16')                          # Matches the image with "ubuntu16"
m.get('ubuntu16', architecture='i386')  # Matches the image with "ubuntu16" and␣
↪overrides the architecture
```

get(*id=None*, *name=None*, *version=None*, *type=None*, *architecture=None*, *distribution=None*)
Lookup an image. If none is found, an image is returned with no data in it. Use the arguments to override individual details.

Parameters

- **name** (*str*) – optional name
- **type** (*str*) – optional type
- **distribution** (*str*) – optional distribution
- **version** (*str*) – optional version
- **architecture** (*str*) – optional architecture

Returns: *nflex_connector_utils.image_detail.ImageDetail*

class nflex_connector_utils.metadata.**Metadata**(*values=None*, *default_namespace=None*)
A representation of an resource metadata. Metadata can be set directly by using `Metadata()` or added piece by piece by using the `add()` method.

Parameters

- **values** (*list*) – An optional list of 2 or 3 element tuples. 2-element tuples have `(key, value)` and 3-element tuples have `(namespace, key, value)`.
- **default_namespace** (*str*) – Optional default namespace. This defaults to `provider_specific`.

Examples

Create two metadata key/values in the default `provider_specific` namespace:

```
Metadata([('key1', 'value1'), ('key2', 'value2')])
```

Create two metadata key/values in an `alt-ns` namespace:

```
Metadata([('alt-ns', 'key1', 'value1'), ('alt-ns', 'key2', 'value2')])
```

Add metadata using the `add()` method:

```
m = Metadata()
m.add('key1', 'value1').add('key2', 'value2', namespace='alt-ns')
```

add(*key*, *value*, *namespace=None*)

Add metadata

Parameters

- **key** (*str*) – key
- **value** (*str*) – value
- **namespace** (*str*) – Optional namespace. If not included, the `default_namespace` is used which defaults to `provider_specific`.

Returns returns itself, so that `add` methods can be chained

Return type *nflex_connector_utils.metadata.Metadata*

serialize()

Serialize the contents

class nflex_connector_utils.ip_address.**IpAddress**(*ip_address=None*, *network_id=None*, *network_name=None*, *description=None*)

A representation of an IP address

Parameters

- **ip_address** (*str*) – IPv4 or IPv6 address
- **description** (*str*) – Description of the ip address. Useful when there is no network, e.g. if the IP address is public.
- **network_id** (*str*) – Optional network id. This isn't used yet, but may be used in the future to associate an IP address with a network.
- **network_name** (*str*) – Optional network name. This isn't used yet.

serialize()

Serialize the contents

class nflex_connector_utils.resource.**Resource**(*id=None*, *name=None*, *type=None*, *region=None*, *locations=None*, *provider_created_at=None*, *metadata=None*, *native_portal_link=None*, *connections=None*)

A representation of a resource. This contains all data common to all resources.

Parameters

- **id** (*str*) – Unique identifier of this resource type. The `(id, type)` tuple uniquely identifies a resource.
- **type** (*str*) – Type of a resource, e.g. `server`, `network`, `volume`, ...
- **name** (*str*) – Human readable name of the resource
- **provider_created_at** (*str or datetime*) – An optional string or datetime object when the resource was created. This should never change.
- **native_portal_link** (*str*) – An optional url to a page on a provider portal with details of the resource.
- **region** (*nflex_connector_utils.locations.Region*) – An optional *nflex_connector_utils.locations.Region* object that associates the resource with a CMP location.
- **locations** (*nflex_connector_utils.locations.Locations*) – An optional *nflex_connector_utils.locations.Locations* objects that provides extended location information.
- **connections** (*nflex_connector_utils.connections.Connections*) – An optional *nflex_connector_utils.connections.Connections* object
- **metadata** (*nflex_connector_utils.metadata.Metadata*) – An optional *nflex_connector_utils.metadata.Metadata* object

serialize()

Serialize the contents

class nflex_connector_utils.appliance.**Appliance**(*size_b=None*, *type_id=None*, ***kwargs*)

A representation of an appliance.

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **type_id** (*str*) – one of `firewall`, `load_balancer`, `router`, `switch`, `storage`, `kvm`, `unknown`

serialize()

Serialize the contents

class nflex_connector_utils.network.**Network**(***kwargs*)

A representation of a network

Parameters **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.

serialize()

Serialize the contents

class nflex_connector_utils.server.**Server**(*cpu_hz=None*, *cpu_cores=None*, *ram_b=None*, *volumes_b=None*, *state=None*, *provider_state=None*, *image_detail=None*, *instance_type=None*, *ip_addresses=None*, *is_virtual=None*, ***kwargs*)

A representation of a server

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **cpu_hz** (*int*) – Optional CPU Speed in Hz
- **cpu_cores** (*int*) – Optional number of CPU cores
- **ram_b** (*int*) – Optional RAM size in bytes
- **volumes_b** – Optional total volume size in bytes
- **state** – CMP state. Must be one of: `unknown`, `pending`, `running`, `shutting`, `terminated`, `stopping`, `stopped`,
- **provider_state** (*str*) – Optional provider description of the state, can be any text.
- **instance_type** (*str*) – Optional text value describing the instance type
- **is_virtual** (*bool*) – Optional, set to `True` if the server is physical. Defaults to `False`.
- **image_detail** (*nflex_connector_utils.image_detail.ImageDetail*) – An optional *nflex_connector_utils.image_detail.ImageDetail* object
- **ip_addresses** (*nflex_connector_utils.ip_address.IpAddress*) – An optional *nflex_connector_utils.ip_address.IpAddress* object

serialize()

Serialize the contents

class nflex_connector_utils.volume.**Volume**(*size_b=None*, *encrypted=None*, *iops=None*, *zone_name=None*, ***kwargs*)

A representation of a volume. See

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **size_b** (*int*) – Optional size in bytes
- **encrypted** (*bool*) – Optional, set to true if the volume is encrypted
- **iops** (*int*) – Optional iops
- **zone_name** (*str*) – Optional zone name

serialize()

Serialize the contents

size_b

Size in bytes

class nflex_connector_utils.account.**Account**(*id=None*, *metadata=None*)

A representation of an Account

serialize()

Serialize the contents

update(*context*, *account_id*)
Updates CMP account

class nflex_connector_utils.service_offering.**ServiceOffering**(*type_id=None*, ***kwargs*)

A representation of a service offering

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **type_id** – Type of service offering. Free text.

serialize()
Serialize the contents

class nflex_connector_utils.compute_pool.**ComputePool**(*cpu_hz=None*, *memory_b=None*, *storage_b=None*, *billing_tag=None*, ***kwargs*)

A representation of a compute pool.

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **cpu_hz** (*int*) – cpu in Hz (optional)
- **memory_b** (*int*) – memory in bytes (optional)
- **storage_b** (*int*) – storage in bytes (optional)
- **billing_tag** (*str*) – billing tag for the compute pool (optional)

serialize()
Serialize the contents

class nflex_connector_utils.saas_user.**SaasUser**(*avatar_url=None*, *phone=None*, *address=None*, *language=None*, *is_active=None*, *disk_quota_b=None*, *disk_used_b=None*, *email=None*, *country=None*, ***kwargs*)

A representation of an Saas User

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **user_id** (*str*) – Id of the Saas User
- **avatar_url** (*str*) – Avatal URL of the user
- **phone** (*str*) – Phone number of the user
- **address** (*str*) – Address of the Saas User
- **language** (*str*) – Preferred language of the user
- **is_active** (*Boolean*) – To check if the user is active or not
- **disk_quota_b** (*str*) – The Storage allocated to the user
- **disk_used_b** (*str*) – The Storage used by the user

serialize()
Serialize the contents

class nflex_connector_utils.colo_space.**ColoSpace**(*power_allocation_w=None*, *type_id=None*, *colo_space_location=None*, *customer_name=None*, *customer_label=None*, *customer_description=None*, *combination=None*, ***kwargs*)

A representation of a colo space.

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **power_allocation_w** (*int*) – Power Allocation in W (optional)
- **type_id** (*str*) – Type ID (optional)
- **colo_space_location** (*str*) – Location (optional)
- **customer_name** (*str*) – Customer Name (optional)
- **customer_label** (*str*) – Customer Label (optional)
- **customer_description** (*str*) – Customer Description (optional)
- **combination** (*str*) – Combination (optional)

serialize()
Serialize the contents

class nflex_connector_utils.circuit.**Circuit**(*type_id=None*, *carrier=None*, *reference=None*, *endpoint_a=None*, *endpoint_b=None*, ***kwargs*)

A representation of a circuit.

Parameters

- **base** (*base*) – See *nflex_connector_utils.resource.Resource* for common resource args.
- **type_id** (*str*) – Type ID (optional)
- **carrier** (*str*) – Carrier (optional)
- **reference** (*str*) – Circuit reference (optional)
- **endpoint_a** (*str*) – One endpoint for the circuit (optional)
- **endpoint_b** (*str*) – The other endpoint for the circuit (optional)

serialize()
Serialize the contents

class nflex_connector_utils.parser.**CaseExpressionParser**
Takes care of parsing (nested) SQL CASE statements with variables

class nflex_connector_utils.parser.**ExpressionParser**
Takes care of parsing simple math expressions with variables

class nflex_connector_utils.parser.**ParsedEntry**(*name*, *unit*, *counter*, *specialisation*, *conversion=None*)

Represents a parsed mapping entry

convert(***kwargs*)
Different syntax for *value()*

Evaluate metric based on initial value:

```
metric = mapping['metric1']
try:
    # 10 * 10 = 100
    value = metric.convert(
        value=10,
    )
except VariableLookupError as e:
    raise e

name = metric.name()
unit = metric.unit()
counter = metric.counter()
specialisation = metric.specialisation()
```

counter()
Retrieves metric's counter

name()
Retrieves metric's name

specialisation()
Retrieves metric's specialisation

unit()
Retrieves metric's unit

value(***kwargs*)
Evaluates conversion expression if it has one based on kwargs input. :raises VariableLookupError: failed to evaluate expression based on variables given.

Returns value: evaluated value or original value if definition has

no conversion expression

Parameters

- ****kwargs** (*dict*) –
- **specified definition. The original value should be expressed as** (*of*) –
- **kwargs['value']** –

Examples

See *load_metric_mapping()* for load example

Evaluate metric based on initial value:

```
metric = mapping['metric1']
try:
    # 10 * 10 = 100
    value = metric.value(
        value=10,
    )
```

(continues on next page)

(continued from previous page)

```
except VariableLookupError as e:
    raise e

name = metric.name()
unit = metric.unit()
counter = metric.counter()
```

class nflex_connector_utils.parser.**SimpleExpressionParser**
Takes care of parsing only python ternary expressions and simple math expressions with variables

class nflex_connector_utils.parser.**TernaryExpressionParser**
Takes care of parsing Python ternary statements with variables

exception nflex_connector_utils.parser.**VariableLookupError**
Occurs when a variable lookup fails

nflex_connector_utils.parser.**load_metric_mapping**(*file_path='mapping.yaml'*)
Loads a selection of metric definition mappings using SimpleExpressionParser. It does so by mapping values per metric key to ParsedEntry value which looks for conversion_expr in each metric definition and runs the logic on that field per evaluate mechanism.

Returns mapping: mapping of metric with conversion expression ready to

be evaluated. :rtype: dict of str: *ParsedEntry*

Parameters **file_path** (*str*) – path to file (optional)

Examples

read mappings from mappings.yaml from current directory (or absolute path):

metric1: name: metric1 unit: "%" conversion_expr: value * 10

Get original value:

```
value = 10
```

Load mapping:

```
# can specify custom path by passing argument file_path
# example file_path='my_project/definition.yaml'
mapping = load_metric_mapping()
```

Evaluate metric based on initial value:

See *ParsedEntry*

class nflex_connector_utils.logger.**Logger**(*context*, *customer_id*, *account_id*, *resource_id=None*)
Nflex Logger is simple wrapper for context.log which is supposed to be used in nflex module. This logger is achieves the following:

- Each log message has header including customer_id, account_id, resource_id (optional).
- The output destination can be changed easily between context.log and print
- When you execute via flexer or GUI for test, you need to use "print" so you can see log from the result of flexer api(GUI used this api implicitly) you need to use "context.log" so you can see log at module status page. Through the context.module_id we dynamically know when it should use the logs API as this parameter is only set on production environments.

Parameters

- **context** (*dict*) – nflex module's context
- **customer_id** (*str*) – event's customer id
- **account_id** (*str*) – event's account id
- **resource_id** (*str*) – event's resource id (nullable)

Examples

Set up logger:

```
# Nflex Module Context
context= {}

# Nflex
event = {
    'customer_id': 'fake-id',
    'account_id': 'fake-id',
    'resource': {
        'id': 'fake-resource-id'
    }
}

logger = Logger(
    context,
    customer_id=event.get('customer_id', ''),
    account_id=event.get('account_id', ''),
    resource_id=event['resource']['id']
)
```

Log information with severity info, warn or error:

```
logger.info('logging info %s' % ('something'))
logger.warn('logging warn %s' % ('something'))
logger.error('logging error %s' % ('something'))
```

nflex_connector_utils.time.**setup_time_interval**(*event*, *backfill_time=None*, *initial_interval=0*, *skew=0*, *time_format='%Y-%m-%dT%H:%M:%S.%fZ'*, *logger=None*)

sets up a time interval based on a combination of event data (last_update) and provider restriction (skew logic)

Returns

- start_time (datetime): interval start
- end_time (datetime): interval end

Return type (tuple)

Parameters

- **event** (*dict*) – module's event that contains:
  - last_update (datetime): event's last_update which specifies the last time when data was fetched from provider

  - poll_interval (int): event's poll-interval in seconds which describes the time interval on when to fetch data from the provider (time of the execution - poll_interval)

- **initial_interval** (*int*) – additional interval parameter in seconds to add to existing poll interval when generating interval start time (optional)
- **backfill_time** (*int*) – the time required in seconds to backfill to ensure no data is missed (optional)
- **time_format** (*str*) – the time format used to parse last_update datetime optional and defaults to %Y-%m-%dT%H:%M:%S.%fZ format
- **skew** (*int*) – time in seconds used to discount from end time (optional)
- **logger** (*option*) – nflex connector util's *logger* that allows to log info regarding usage of last_update and backfill_time

Examples

set up time interval:

```
event = {
    'last_update':datetime.utcnow(),
    'resource': {
        'poll_interval': 15*60,
    },
}
```

(optional) set up logger:

```
# get these values from nflex handler which has event and context
logger = Logger(
    context=context,
    customer_id=customer_id
    account_id=account_id,
    resource_id=resource_id,
)
```

setup time interval:

```
output = setup_time_interval(
    event=event,
    backfill_time=3600,
    initial_interval=5*60,
    skew=5*60,
    logger=logger,
)
```

parse and use the values:

```
start_time, end_time = output
```

Exceptions predefined here will be handled in spend collector.

exception nflex_connector_utils.errors.**InsufficientPermission**(*msg=None*)

Exception for insufficient permission. Raise this if the account does not have permission to access billing data

Parameters **msg** (*str*) – customised error msg.

Returns Raise an exception with either default message or given message

exception nflex_connector_utils.errors.**InvalidCredentials**(*msg=None*)
Exception for invalid account credentials

Parameters **msg** (*str*) – customised error msg.

Returns Raise an exception with either default message or given message

exception nflex_connector_utils.errors.**NoBillingReport**(*msg=None*)
Exception for no billing report file found

Parameters **msg** (*str*) – customised error msg.

Returns Raise an exception with either default message or given message

exception nflex_connector_utils.errors.**SpendError**
Exception for any other kind of spend errors

CHAPTER 4

Examples

Here is a full example of some nflex resource connector code:

```
def get(event, context):
    resources = serialize_list([
        Appliance(id='appliance-1', name='Network 1', type_id='firewall'),
        Network(id='network-1', name='Network 1'),
        Server(id='server-1', name='Server 1'),
        Volume(id='volume-1', name='Volume 1'),

        # A volume with some more details
        Volume(
            id='volume-2',
            name='Volume 2',
            size_b=1024 * 1024 * 1024 * 1024,  # 1 TB disks
            encrypted=False,
            iops=None,
            zone_name='eu-west-1',
        ),

        # A server with some more details
        Server(
            id='server-2',
            name='Server with lots of details',
            provider_created_at=datetime(year=2017, month=2, day=4),

            # Associate the server with the volume and network
            connections=Connections(
                volumes=['volume-2'],
                networks=['network-1']),

            # Add some metadata
            metadata=Metadata([('key1', 'value1'), ('key2', 'value2')]),

            # Set a location by matching a region id with locations
```

(continues on next page)

(continued from previous page)

```
            # in the CMP database. This is a legacy procedure which
            # requires data to be added to the CMP database by the CMP
            # development team.
            region=Region('region-1'),

            # An example of a single location association
            locations = Locations([{
                'country': {'name': 'France'},
                'city': {'name': 'Paris'},
                'location': {
                    'name': 'Legendary Paris Place',
                    'latitude': 48.860764,
                    'longitude': 2.393646,
                }
            }]),

            native_portal_link='http://www.example.com/servers/server-2',
            state='stopped',
            provider_state='powered off',

            image_detail=ImageDetail(id='image-1',
                                     name='ubuntu 16.04',
                                     type='Linux',
                                     distribution='Ubuntu',
                                     version='16.04',
                                     architecture='64'),

            cpu_cores=2,
            cpu_hz=2500000,                       # 2.5 GHz
            ram_b=1024 * 1024 * 32,               # 1 GB RAM
            volumes_b=1024 * 1024 * 1024 * 1024,  # 1 TB disks

            ip_addresses=[IpAddress(
                ip_address='192.168.0.1',
                network_id='network-1',
                network_name='Network 1',
            )]
        ),
    ])

    return resources
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

n

Index