

---

**pysnow**  
*Release 0.7.5*

**Dec 02, 2018**



<b>1</b>	<b>Compatibility</b>	<b>3</b>
<b>2</b>	<b>Installing</b>	<b>5</b>
<b>3</b>	<b>Testing</b>	<b>7</b>
<b>4</b>	<b>Demo!</b>	<b>9</b>
<b>5</b>	<b>License</b>	<b>11</b>
<b>6</b>	<b>Client</b>	<b>13</b>
<b>7</b>	<b>OAuthClient</b>	<b>15</b>
<b>8</b>	<b>QueryBuilder</b>	<b>17</b>
<b>9</b>	<b>Attachment</b>	<b>19</b>
<b>10</b>	<b>Resource</b>	<b>21</b>
<b>11</b>	<b>ParamsBuilder</b>	<b>23</b>
<b>12</b>	<b>Response</b>	<b>25</b>
<b>13</b>	<b>Exceptions</b>	<b>27</b>
13.1	Generic Exceptions . . . . .	27
13.2	Response Exceptions . . . . .	27
13.3	OAuthClient Exceptions . . . . .	27
13.4	QueryBuilder Exceptions . . . . .	27
<b>14</b>	<b>The Client</b>	<b>29</b>
14.1	Using pysnow.Client . . . . .	29
14.2	Using pysnow.OAuthClient . . . . .	30
<b>15</b>	<b>Resources</b>	<b>31</b>
<b>16</b>	<b>Request parameters</b>	<b>33</b>
16.1	Client object parameters . . . . .	33
16.2	Resource object parameters . . . . .	33

<b>17 Querying</b>	<b>35</b>
17.1 Key-value . . . . .	35
17.2 Using the query builder . . . . .	35
17.3 SN Pass-through . . . . .	36
<b>18 Fetching data</b>	<b>37</b>
18.1 Multiple records . . . . .	37
18.2 First record . . . . .	38
18.3 First or none . . . . .	38
18.4 Exactly one . . . . .	38
18.5 One or none . . . . .	39
<b>19 Creating a new record</b>	<b>41</b>
<b>20 Updating a record</b>	<b>43</b>
<b>21 Deleting a record</b>	<b>45</b>
<b>22 Sorting</b>	<b>47</b>
<b>23 Using the OAuthClient</b>	<b>49</b>
<b>24 Using the QueryBuilder</b>	<b>51</b>
<b>25 Attaching files</b>	<b>53</b>
<b>26 Using threaded queries</b>	<b>55</b>
<b>27 Session with auto-retry</b>	<b>57</b>
<b>Python Module Index</b>	<b>59</b>

ServiceNow HTTP client library written in Python.



# CHAPTER 1

---

## Compatibility

---

Fully tested and verified working: Python 2.7+ and Python 3.3+





## CHAPTER 2

---

### Installing

---

```
$ pip install pysnow
```



## CHAPTER 3

---

### Testing

---

The code is automatically tested using **travis** and **nose**.

To run tests manually, move to the cloned directory and run:

```
$ nosetests --cover-package=pysnow --with-coverage --cover-erase
```



## CHAPTER 4

---

Demo!

---

This demo features the `pysnow.QueryBuilder` and shows an example of how to fetch records using the **incident** table API.



### MIT License

Copyright (c) 2018 Robert Wikman <[rbw@vault13.org](mailto:rbw@vault13.org)>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.





```
class pysnow.client.Client (instance=None, host=None, user=None, password=None,  
raise_on_empty=None, request_params=None, use_ssl=True,  
session=None)
```

User-created Client object.

#### Parameters

- **instance** – Instance name, used to construct host
- **host** – Host can be passed as an alternative to instance
- **user** – User name
- **password** – Password
- **raise\_on\_empty** – Whether or not to raise an exception on 404 (no matching records), defaults to True
- **request\_params** – Request params to send with requests globally (deprecated)
- **use\_ssl** – Enable or disable the use of SSL, defaults to True
- **session** – Optional `requests.Session` object to use instead of passing user/pass to `Client`

#### Raises

- `InvalidUsage`: On argument validation error

```
resource (api_path=None, base_path='/api/now', chunk_size=None)
```

Creates a new `Resource` object after validating paths

#### Parameters

- **api\_path** – Path to the API to operate on
- **base\_path** – (optional) Base path override
- **chunk\_size** – Response stream parser chunk size (in bytes)

#### Returns

- Resource object

**Raises**

- InvalidUsage: If a path fails validation

```
class pysnow.oauth_client.OAuthClient (client_id=None, client_secret=None, token_updater=None, **kwargs)
    Pysnow Client with extras for oauth session and token handling.
```

**Parameters**

- **client\_id** – client\_id from ServiceNow
- **client\_secret** – client\_secret from ServiceNow
- **token\_updater** – function called when a token has been refreshed
- **kwargs** – kwargs passed along to `pysnow.Client`

```
set_token (token)
    Validate and set token
```

**Parameters** **token** – the token (dict) to set

```
resource (api_path=None, base_path='/api/now', chunk_size=None)
    Overrides resource() provided by pysnow.Client with extras for OAuth
```

**Parameters**

- **api\_path** – Path to the API to operate on
- **base\_path** – (optional) Base path override
- **chunk\_size** – Response stream parser chunk size (in bytes)

**Returns**

- Resource object

**Raises**

- `InvalidUsage`: If a path fails validation

```
generate_token (user, password)
    Takes user and password credentials and generates a new token
```

**Parameters**

- **user** – user
- **password** – password

**Returns**

- dictionary containing token data

**Raises**

- `TokenCreateError`: If there was an error generating the new token

**class** pysnow.query\_builder.**QueryBuilder**  
Query builder - for constructing advanced ServiceNow queries

**field** (*field*)  
Sets the field to operate on  
**Parameters** **field** – field (str) to operate on

**order\_descending** ()  
Sets ordering of field descending

**order\_ascending** ()  
Sets ordering of field ascending

**starts\_with** (*starts\_with*)  
Adds new STARTSWITH condition  
**Parameters** **starts\_with** – Match field starting with the provided value

**ends\_with** (*ends\_with*)  
Adds new ENDSWITH condition  
**Parameters** **ends\_with** – Match field ending with the provided value

**contains** (*contains*)  
Adds new LIKE condition  
**Parameters** **contains** – Match field containing the provided value

**not\_contains** (*not\_contains*)  
Adds new NOTLIKE condition  
**Parameters** **not\_contains** – Match field not containing the provided value

**is\_empty** ()  
Adds new ISEMPY condition

**is\_not\_empty** ()  
Adds new ISNOTEEMPTY condition

**equals** (*data*)

Adds new IN or = condition depending on if a list or string was provided

**Parameters** **data** – string or list of values

**Raise**

- QueryTypeError: if *data* is of an unexpected type

**not\_equals** (*data*)

Adds new NOT IN or = condition depending on if a list or string was provided

**Parameters** **data** – string or list of values

**Raise**

- QueryTypeError: if *data* is of an unexpected type

**greater\_than** (*greater\_than*)

Adds new > condition

**Parameters** **greater\_than** – str or datetime compatible object (naive UTC datetime or tz-aware datetime)

**Raise**

- QueryTypeError: if *greater\_than* is of an unexpected type

**less\_than** (*less\_than*)

Adds new < condition

**Parameters** **less\_than** – str or datetime compatible object (naive UTC datetime or tz-aware datetime)

**Raise**

- QueryTypeError: if *less\_than* is of an unexpected type

**between** (*start*, *end*)

Adds new BETWEEN condition

**Parameters**

- **start** – int or datetime compatible object (in SNOW user's timezone)
- **end** – int or datetime compatible object (in SNOW user's timezone)

**Raise**

- QueryTypeError: if *start* or *end* arguments is of an invalid type

**AND** ()

Adds an and-operator

**OR** ()

Adds an or-operator

**NQ** ()

Adds a NQ-operator (new query)

**class** pysnow.attachment.**Attachment** (*resource, table\_name*)  
Attachment management

**Parameters**

- **resource** – Table API resource to manage attachments for
- **table\_name** – Name of the table to use in the attachment API

**get** (*sys\_id=None, limit=100*)  
Returns a list of attachments

**Parameters**

- **sys\_id** – record sys\_id to list attachments for
- **limit** – override the default limit of 100

**Returns** list of attachments

**upload** (*sys\_id, file\_path, name=None, multipart=False*)  
Attaches a new file to the provided record

**Parameters**

- **sys\_id** – the sys\_id of the record to attach the file to
- **file\_path** – local absolute path of the file to upload
- **name** – custom name for the uploaded file (instead of basename)
- **multipart** – whether or not to use multipart

**Returns** the inserted record

**delete** (*sys\_id*)  
Deletes the provided attachment record

**Parameters** **sys\_id** – attachment sys\_id

**Returns** delete result





**class** `pysnow.resource.Resource` (*base\_url=None, base\_path=None, api\_path=None, parameters=None, \*\*kwargs*)

Creates a new *Resource* object

Resources provides a natural way of interfacing with ServiceNow APIs.

### Parameters

- **base\_path** – Base path
- **api\_path** – API path
- **chunk\_size** – Response stream parser chunk size (in bytes)
- **\*\*kwargs** – Arguments to pass along to Request

### path

Get current path relative to base URL

**Returns** resource path

### attachments

Provides an Attachment API for this resource. Enables easy listing, deleting and creating new attachments.

**Returns** Attachment object

### get\_record\_link

 (*sys\_id*)

Provides full URL to the provided *sys\_id*

**Parameters** **sys\_id** – *sys\_id* to generate URL for

**Returns** full *sys\_id* URL

### get

 (*\*args, \*\*kwargs*)

Queries the API resource

### Parameters

- **args** –

- **param query** Dictionary, string or `QueryBuilder` object defaults to empty dict (all)
- **kwargs** –
  - **param limit** Limits the number of records returned
  - **param fields** List of fields to include in the response created\_on in descending order.
  - **param offset** Number of records to skip before returning records
  - **param stream** Whether or not to use streaming / generator response interface

**Returns**

- Response object

**create** (*payload*)

Creates a new record in the API resource

**Parameters** **payload** – Dictionary containing key-value fields of the new record

**Returns**

- Dictionary of the inserted record

**update** (*query, payload*)

Updates a record in the API resource

**Parameters**

- **query** – Dictionary, string or `QueryBuilder` object
- **payload** – Dictionary containing key-value fields of the record to be updated

**Returns**

- Dictionary of the updated record

**delete** (*query*)

Deletes matching record

**Parameters** **query** – Dictionary, string or `QueryBuilder` object

**Returns**

- Dictionary containing information about deletion result

**request** (*method, path\_append=None, headers=None, \*\*kwargs*)

Create a custom request

**Parameters**

- **method** – HTTP method to use
- **path\_append** – (optional) relative to `api_path`
- **headers** – (optional) Dictionary of headers to add or override
- **kwargs** – kwargs to pass along to `requests.Request`

**Returns**

- Response object

**class** pysnow.params\_builder.ParamsBuilder

Provides an interface for setting / getting common ServiceNow sysparms.

**static stringify\_query** (*query*)

Stringifies the query (dict or QueryBuilder) into a ServiceNow-compatible format

**Returns**

- ServiceNow-compatible string-type query

**add\_custom** (*params*)

Adds new custom parameter after making sure it's of type dict.

**Parameters** *params* – Dictionary containing one or more parameters

**custom\_params**

Returns a dictionary of added custom parameters

**display\_value**

Maps to sysparm\_display\_value

**query**

Maps to sysparm\_query

**limit**

Maps to sysparm\_limit

**offset**

Maps to sysparm\_offset

**fields**

Maps to sysparm\_fields

**exclude\_reference\_link**

Maps to sysparm\_exclude\_reference\_link

**suppress\_pagination\_header**

Maps to sysparm\_suppress\_pagination\_header

**as\_dict()**

Constructs query params compatible with `requests.Request`

**Returns**

- Dictionary containing query parameters

**class** `pysnow.response.Response` (*response*, *resource*, *chunk\_size=8192*, *stream=False*)

Takes a `requests.Response` object and performs deserialization and validation.

**Parameters**

- **response** – `requests.Response` object
- **resource** – parent `resource.Resource` object
- **chunk\_size** – Read and return up to this size (in bytes) in the stream parser

**all** ()

Returns a chained generator response containing all matching records

**Returns**

- Iterable response

**first** ()

Return the first record or raise an exception if the result doesn't contain any data

**Returns**

- Dictionary containing the first item in the response content

**Raise**

- `NoResults`: If no results were found

**first\_or\_none** ()

Return the first record or `None`

**Returns**

- Dictionary containing the first item or `None`

**one** ()

Return exactly one record or raise an exception.

**Returns**

- Dictionary containing the only item in the response content

**Raise**

- **MultipleResults**: If more than one records are present in the content
- **NoResults**: If the result is empty

**one\_or\_none** ()

Return at most one record or raise an exception.

**Returns**

- Dictionary containing the matching record or None

**Raise**

- **MultipleResults**: If more than one records are present in the content

**update** (*payload*)

Convenience method for updating a fetched record

**Parameters** **payload** – update payload

**Returns** update response object

**delete** ()

Convenience method for deleting a fetched record

**Returns** delete response object

**upload** (*\*args, \*\*kwargs*)

Convenience method for attaching files to a fetched record

**Parameters**

- **args** – args to pass along to `Attachment.upload`
- **kwargs** – kwargs to pass along to `Attachment.upload`

**Returns** upload response object

### 13.1 Generic Exceptions

```
class pysnow.exceptions.InvalidUsage
```

### 13.2 Response Exceptions

```
class pysnow.exceptions.ResponseError(error)
class pysnow.exceptions.MissingResult
class pysnow.exceptions.UnexpectedResponseFormat
class pysnow.exceptions.ReportUnavailable
class pysnow.exceptions.NoResults
class pysnow.exceptions.MultipleResults
```

### 13.3 OAuthClient Exceptions

```
class pysnow.exceptions.MissingToken
class pysnow.exceptions.TokenCreateError(error, description, status_code)
```

### 13.4 QueryBuilder Exceptions

```
class pysnow.exceptions.QueryTypeError
class pysnow.exceptions.QueryMissingField
```

```
class pysnow.exceptions.QueryEmpty  
class pysnow.exceptions.QueryExpressionError  
class pysnow.exceptions.QueryMultipleExpressions
```



### The Client comes in two forms:

- The regular `pysnow.Client` - use if you're authenticating with password credentials or wish to pass an already created session object.
- The `pysnow.OAuthClient` - use if you wish to do OAuth with an OAuth2 enabled ServiceNow instance.

## 14.1 Using `pysnow.Client`

This shows some examples of how to create the `pysnow.Client` using username and password or a custom session object

See the `pysnow.Client` documentation for details.

### 14.1.1 With username and password

```
s = pysnow.Client(instance='myinstance',
                 user='myusername',
                 password='mypassword')
```

### 14.1.2 With a custom session object

You can pass a custom session object to `pysnow.Client`. In this example password credentials are used, but with SSL verification disabled.

```
s = requests.Session()
s.verify = False
s.auth = requests.auth.HTTPBasicAuth('myusername', 'mypassword')
```

(continues on next page)

```
sn = pysnow.Client(instance='myinstance', session=s)
```

## 14.2 Using pysnow.OAuthClient

Pysnow provides the `pysnow.OAuthClient` to simplify the process of obtaining initial tokens, refreshing tokens and keeping tokens in sync with your storage.

Should the `pysnow.OAuthClient` not be sufficient for your requirements some reason, you can always create a custom `Requests` compatible OAuth session and pass along to `pysnow.Client()`

Enabling OAuth in ServiceNow is fairly simple but beyond the scope of this document. Details on how to do this can be found in the [official ServiceNow documentation](#).

### 14.2.1 Getting initial tokens

In order to use the `pysnow.OAuthClient` you first need to obtain a new token from ServiceNow. Creating a new token bound to a certain user is easy. Simply call `pysnow.OAuthClient.generate_token()` and keep it in your storage (e.g. in session or database)

```
s = pysnow.OAuthClient(client_id='<client_id_from_servicenow>', client_secret='
↳<client_secret_from_servicenow>', instance='<instance_name>')

if not session['token']:
    # No previous token exists. Generate new.
    session['token'] = s.generate_token('<username>', '<password>')
```

### 14.2.2 Using tokens

Once an initial token has been obtained it will be refreshed automatically upon usage, provided its `refresh_token` hasn't expired.

After a token has been refreshed, the provided `token_updater()` function will be called with the refreshed token as first argument.

```
def updater(new_token):
    print("OAuth token refreshed!")
    session['token'] = new_token

s = pysnow.OAuthClient(client_id='<client_id_from_servicenow>', client_secret='
↳<client_secret_from_servicenow>', token_updater=updater, instance='<instance_name>')
s.set_token(session['token'])
```

`pysnow.Resource` provides an interface to easily and efficiently create, read, update and delete records in ServiceNow. Additionally, it's also equipped with helpers for common operations, such as attaching files.

Check out the [Resource API documentation](#) for more info.

```
incident = client.resource(api_path='/table/incident')
incident.parameters.display_value = True

record = incident.get(query={'number': 'INC012345'}).one()

sys_id = record['sys_id']
incident.attachments.upload(sys_id, file_path='/tmp/document.txt')

updated = incident.update(query={'sys_id': sys_id},
                          payload={
                              'short_description': 'Uh-uh',
                              'description': 'I fear I might be getting deleted.'
                          })

incident.delete(query={'sys_id': sys_id})
```



---

## Request parameters

---

Request parameters (sysparms in ServiceNow) are key-values passed in the query string for GET requests. Default parameters can be set on both the `pysnow.Client` and the `pysnow.Resource` using the `parameters` property. Parameters set on **Client** are automatically inherited by **Resource**, but can of course be overridden.

Please see the API documentation for more info on this.

### 16.1 Client object parameters

```
client = pysnow.Client(instance=instance,
                       user=username,
                       password=password)

client.parameters.display_value = False
client.parameters.exclude_reference_link = True
client.parameters.add_custom({'foo': 'bar'})
```

### 16.2 Resource object parameters

```
incident = client.resource(api_path='/custom/api')
incident.parameters.add_custom({'foo': 'bar'})
```



There are three different ways to create queries using the pysnow library.

### 17.1 Key-value

Simple. And sufficient in many cases.

```
content = incident.get(query={'NUMBER': 'INC012345'}).one()
```

### 17.2 Using the query builder

The recommended way to create advanced queries.

See the `pysnow.QueryBuilder()` documentation for details.

```
# Set start and end range
start = datetime(1970, 1, 1)
end = datetime.now() - timedelta(days=20)

# Query incident records with number starting with 'INC0123', created between 1970-01-
↳ 01 and 20 days back in time
qb = (
    pysnow.QueryBuilder()
        .field('number').starts_with('INC0123')
        .AND()
        .field('sys_created_on').between(start, end)
        .AND()
        .field('sys_updated_on').order_descending()
)

iterable_content = incident.get(query=qb).all()
```

## 17.3 SN Pass-through

It's recommended to use the query builder for complex queries, as it offers error handling and a cleaner way of creating queries.

However, you can still use SN pass-through queries should the query builder not satisfy your needs for some reason.

Below is the pass-through equivalent of the QB in the previous example. You decide ;)

```
# Set start and end range
start = datetime(1970, 1, 1)
end = datetime.now() - timedelta(days=20)

# Query incident records with number starting with 'INC0123', created between 1970-01-
↳ 01 and 20 days back in time
iterable_content = incident.get(query='numberSTARTSWITHINC0150^sys_created_
↳ onBETWEENjavascript:gs.dateGenerate("%s")@javascript:gs.dateGenerate("%s")' %
↳ (start, end)).all()
```



---

## Fetching data

---

The `pysnow.Resource.get()` returns an instance of `pysnow.Response`, which exposes an interface to the various methods available for getting the data you're after.

---

**Note:** Fetching large amounts of data? Use the incremental stream parser by passing `stream=True` to `get()`, this will return a memory-friendly generator instead of a buffered result.

---

### 18.1 Multiple records

In this example, `pysnow.Response.all()` returns a generator (`stream=True`), which is iterated on in chunks of 8192 bytes by default.

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Query for incidents with state 1
response = incident.get(query={'state': 1}, stream=True)

# Iterate over the result and print out `sys_id` of the matching records.
for record in response.all():
    print(record['sys_id'])
```

## 18.2 First record

The `pysnow.Response.first()` returns the first record in a result containing one or more records. An exception is raised if the result doesn't contain any records.

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Query for incidents with state 3
response = incident.get(query={'state': 3})

# Print out the first match
print(response.first())
```

## 18.3 First or none

The `pysnow.Response.first_or_none()` returns the first record in a result containing one or more records. None is returned if the result doesn't contain any records.

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Query for incidents with state 3
response = incident.get(query={'state': 3})

# Print out the first match, or `None`
print(response.first_or_none())
```

## 18.4 Exactly one

The `pysnow.Response.one()` returns exactly one record. An exception is raised if the result is empty or contains multiple records.

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Query for incident with number INC012345
```

(continues on next page)

(continued from previous page)

```
response = incident.get(query={'number': 'INC012345'})  
  
# Print out the matching record  
print(response.one())
```

## 18.5 One or none

The `pysnow.Response.one_or_none()` returns one record, or `None` if no matching records were found. An exception is raised if the result contains multiple records

```
import pysnow  
  
# Create client object  
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')  
  
# Create a new resource for the incident table API  
incident = c.resource(api_path='/table/incident')  
  
# Query for incident with number INC012345  
response = incident.get(query={'number': 'INC012345'})  
  
# Print out the matching record, or `None` if no matches were found.  
print(response.one_or_none())
```



---

## Creating a new record

---

The `Client.resource.create()` takes a dictionary payload with key-values of the record to be created.

**Note:** This method calls `pysnow.Resource.one()` if the record was created successfully, returning a dictionary of the created record.

---

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Set the payload
new_record = {
    'short_description': 'Pysnow created incident',
    'description': 'This is awesome'
}

# Create a new incident record
result = incident.create(payload=new_record)
```



## CHAPTER 20

---

### Updating a record

---

The `Client.resource.update()` takes a **payload** and **query** to perform an update.

---

**Note:** This method returns the updated record (dict) if the operation was successful. Refer to `Client.resource.custom()` if you want a **Response** object back.

---

---

**Note:** Updating multiple records is **not supported**.

---

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

update = {'short_description': 'New short description', 'state': 5}

# Update 'short_description' and 'state' for 'INC012345'
updated_record = incident.update(query={'number': 'INC012345'}, payload=update)

# Print out the updated record
print(updated_record)
```





## CHAPTER 21

---

### Deleting a record

---

Deletes the queried record and returns the result (dict).

---

**Note:** Deletion of multiple records is **not supported**.

---

```
import pysnow

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# Define a resource, here we'll use the incident table API
incident = c.resource(api_path='/table/incident')

# Delete incident with number 'INC012345'
result = incident.delete(query={'number': 'INC012345'})
```



Sorting can be done over multiple fields in ascending or descending order.

Example showing how sorting can be dynamically created from a comma-separated string of fields.

```
query = pysnow.QueryBuilder()
fields = '-created_at,priority'.split(',')

query.field('priority').equals(['3', '4']).AND()

for (idx, field) in enumerate(fields):
    if field[:1] == '-':
        query.field(field[1:]).order_descending()
    else:
        query.field(field).order_ascending()

    if idx != len(fields) - 1:
        query.AND()

print(str(query)) # priorityIN3,4^ORDERBYDESCcreated_at^ORDERBYpriority
```



---

## Using the OAuthClient

---

Example showing how tokens can be obtained, stored and refreshed using the OAuthClient.

In this example a basic dictionary is used as store, which offers no persistence, meaning that `OAuthClient.generate_token` will be called every time this code executes, which introduces an overhead. The store here could be a database table, file, session or whatever you want.

```
import pysnow

store = {'token': None}

# Takes care of refreshing the token storage if needed
def updater(new_token):
    print("OAuth token refreshed!")
    store['token'] = new_token

# Create the OAuthClient with the ServiceNow provided `client_id` and `client_secret`,
↪ and a `token_updater`
# function which takes care of refreshing local token storage.
s = pysnow.OAuthClient(client_id='<client_id_from_servicenow>',
                       client_secret='<client_secret_from_servicenow>',
                       token_updater=updater, instance='<instance_name>')

if not store['token']:
    # No previous token exists. Generate new.
    store['token'] = s.generate_token('<username>', '<password>')

# Set the access / refresh tokens
s.set_token(store['token'])

# We should now be good to go. Let's define a `Resource` for the incident API.
incident_resource = s.resource(api_path='/table/incident')

# Fetch the first record in the response
record = incident_resource.get(stream=True).first()
```

(continues on next page)

(continued from previous page)

```
# Print it  
print(record)
```

---

## Using the QueryBuilder

---

Example showing how the QueryBuilder can be used to construct a query using the Python datetime library.

```
import pysnow
from datetime import datetime, timedelta

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

today = datetime.today()
sixty_days_ago = today - timedelta(days=60)

# Query incident records with number starting with 'INC0123', created between 60 days_
↪ago and today.
qb = (
    pysnow.QueryBuilder()
    .field('number').starts_with('INC0123')
    .AND()
    .field('sys_created_on').between(sixty_days_ago, today)
)

incident = c.resource(api_path='/table/incident')

response = incident.get(query=qb)

# Iterate over the matching records and print out number
for record in response.all():
    print(record['number'])
```





---

## Attaching files

---

Shows how to upload a file using `Resource.attachments`.

Check out the Attachment API documentation for more info.

### Using `Response.upload()` convenience method

```
# Create a resource
incidents = client.resource(api_path='/table/incident')

# Fetch and attach
incident = incidents.get(query={'number': 'INC01234'})
incident.upload(file_path='/tmp/attachment.txt')
```

### Using `Resource.attachments.upload()`

```
# Create a resource
incidents = client.resource(api_path='/table/incident')

# Uploads file '/tmp/attachment.txt' to the provided incident
incidents.attachments.upload(sys_id='9b9dd196dbc91f005ab1f58dbf96192b',
                             file_path='/tmp/attachment.txt')
```



---

## Using threaded queries

---

This is an example of multiple threads doing simple fetches.

**Note:** This example uses `concurrent.futures` and expects you to be familiar with `pysnow.Resource.get()`.

---

```
import concurrent.futures
import pysnow

def just_print(client, query):
    # Run the query
    response = client.get(query=query)

    # Iterate over the result and print out `sys_id` and `state` of the matching_
    ↪ records.
    for record in response.all():
        print(record['sys_id'], record['state'])

# Create client object
c = pysnow.Client(instance='myinstance', user='myusername', password='mypassword')

# list of simple items to query
queries = ({'api': '/table/incident', 'q': {'state': 1}}, {'api': '/table/incident',
    ↪ 'q': {'state': 3}})

# build taskqueue
with concurrent.futures.ThreadPoolExecutor(max_workers=4) as taskpool:
    for query in queries:
        connection = c.resource(api_path=query['api'])
        taskpool.submit(just_print, connection, query['q'])
```



---

## Session with auto-retry

---

You might run into issues if you're creating too many requests against the ServiceNow API. Fortunately, the `requests` library enables users to create their own transport adapter with a retry mechanism from the `urllib3` library.

**You can read more about transport adapters and the retry mechanism here:**

- <http://docs.python-requests.org/en/master/user/advanced/#transport-adapters>
- <https://urllib3.readthedocs.io/en/latest/reference/urllib3.util.html#module-urllib3.util.retry>

This example shows how to automatically retry on an error for about 2 seconds and then fall back to the default error handling.

```
import requests
import pysnow

from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry

s = requests.Session()
s.auth = requests.auth.HTTPBasicAuth('<username>', '<password>')

# set auto retry for about 2 seconds on some common errors
adapter = HTTPAdapter(
    max_retries=Retry(
        total=3,
        backoff_factor=0.3,
        status_forcelist=(401, 408, 429, 431, 500, 502, 503, 504, 511)
    )
)

s.mount('https://', adapter)

sn = pysnow.Client(instance='<instance>', session=s)
```



**p**

`pysnow.attachment`, 19  
`pysnow.client`, 13  
`pysnow.exceptions`, 27  
`pysnow.oauth_client`, 15  
`pysnow.params_builder`, 23  
`pysnow.query_builder`, 17  
`pysnow.resource`, 21  
`pysnow.response`, 25





**A**

add\_custom() (pysnow.params\_builder.ParamsBuilder method), 23  
 all() (pysnow.response.Response method), 25  
 AND() (pysnow.query\_builder.QueryBuilder method), 18  
 as\_dict() (pysnow.params\_builder.ParamsBuilder method), 23  
 Attachment (class in pysnow.attachment), 19  
 attachments (pysnow.resource.Resource attribute), 21

**B**

between() (pysnow.query\_builder.QueryBuilder method), 18

**C**

Client (class in pysnow.client), 13  
 contains() (pysnow.query\_builder.QueryBuilder method), 17  
 create() (pysnow.resource.Resource method), 22  
 custom\_params (pysnow.params\_builder.ParamsBuilder attribute), 23

**D**

delete() (pysnow.attachment.Attachment method), 19  
 delete() (pysnow.resource.Resource method), 22  
 delete() (pysnow.response.Response method), 26  
 display\_value (pysnow.params\_builder.ParamsBuilder attribute), 23

**E**

ends\_with() (pysnow.query\_builder.QueryBuilder method), 17  
 equals() (pysnow.query\_builder.QueryBuilder method), 17  
 exclude\_reference\_link (pysnow.params\_builder.ParamsBuilder attribute), 23

**F**

field() (pysnow.query\_builder.QueryBuilder method), 17

fields (pysnow.params\_builder.ParamsBuilder attribute), 23  
 first() (pysnow.response.Response method), 25  
 first\_or\_none() (pysnow.response.Response method), 25

**G**

generate\_token() (pysnow.oauth\_client.OAuthClient method), 15  
 get() (pysnow.attachment.Attachment method), 19  
 get() (pysnow.resource.Resource method), 21  
 get\_record\_link() (pysnow.resource.Resource method), 21  
 greater\_than() (pysnow.query\_builder.QueryBuilder method), 18

**I**

InvalidUsage (class in pysnow.exceptions), 27  
 is\_empty() (pysnow.query\_builder.QueryBuilder method), 17  
 is\_not\_empty() (pysnow.query\_builder.QueryBuilder method), 17

**L**

less\_than() (pysnow.query\_builder.QueryBuilder method), 18  
 limit (pysnow.params\_builder.ParamsBuilder attribute), 23

**M**

MissingResult (class in pysnow.exceptions), 27  
 MissingToken (class in pysnow.exceptions), 27  
 MultipleResults (class in pysnow.exceptions), 27

**N**

NoResults (class in pysnow.exceptions), 27  
 not\_contains() (pysnow.query\_builder.QueryBuilder method), 17  
 not\_equals() (pysnow.query\_builder.QueryBuilder method), 18

NQ() (pysnow.query\_builder.QueryBuilder method), 18

## O

OAuthClient (class in pysnow.oauth\_client), 15

offset (pysnow.params\_builder.ParamsBuilder attribute), 23

one() (pysnow.response.Response method), 25

one\_or\_none() (pysnow.response.Response method), 26

OR() (pysnow.query\_builder.QueryBuilder method), 18

order\_ascending() (pysnow.query\_builder.QueryBuilder method), 17

order\_descending() (pysnow.query\_builder.QueryBuilder method), 17

## P

ParamsBuilder (class in pysnow.params\_builder), 23

path (pysnow.resource.Resource attribute), 21

pysnow.attachment (module), 19

pysnow.client (module), 13

pysnow.exceptions (module), 27

pysnow.oauth\_client (module), 15

pysnow.params\_builder (module), 23

pysnow.query\_builder (module), 17

pysnow.resource (module), 21

pysnow.response (module), 25

## Q

query (pysnow.params\_builder.ParamsBuilder attribute), 23

QueryBuilder (class in pysnow.query\_builder), 17

QueryEmpty (class in pysnow.exceptions), 27

QueryExpressionError (class in pysnow.exceptions), 28

QueryMissingField (class in pysnow.exceptions), 27

QueryMultipleExpressions (class in pysnow.exceptions), 28

QueryTypeError (class in pysnow.exceptions), 27

## R

ReportUnavailable (class in pysnow.exceptions), 27

request() (pysnow.resource.Resource method), 22

Resource (class in pysnow.resource), 21

resource() (pysnow.client.Client method), 13

resource() (pysnow.oauth\_client.OAuthClient method), 15

Response (class in pysnow.response), 25

ResponseError (class in pysnow.exceptions), 27

## S

set\_token() (pysnow.oauth\_client.OAuthClient method), 15

starts\_with() (pysnow.query\_builder.QueryBuilder method), 17

stringify\_query() (pysnow.params\_builder.ParamsBuilder static method), 23

suppress\_pagination\_header (pysnow.params\_builder.ParamsBuilder attribute), 23

## T

TokenCreateError (class in pysnow.exceptions), 27

## U

UnexpectedResponseFormat (class in pysnow.exceptions), 27

update() (pysnow.resource.Resource method), 22

update() (pysnow.response.Response method), 26

upload() (pysnow.attachment.Attachment method), 19

upload() (pysnow.response.Response method), 26