

---

# **SteelConnection Documentation**

***Release 1.1.8***

**Greg Mueller**

**Jun 25, 2020**



---

## SteelConnection User's Guide:

---

<b>1</b>	<b>License</b>	<b>3</b>
<b>2</b>	<b>Getting Help</b>	<b>5</b>
2.1	Getting Help with SteelConneciton: . . . . .	5
2.2	SteelConnect CX API Reference: . . . . .	5
<b>3</b>	<b>Getting Started</b>	<b>7</b>
3.1	Prerequisites . . . . .	7
3.2	Create an Object: . . . . .	7
3.3	Quickstart . . . . .	7
<b>4</b>	<b>Installation</b>	<b>9</b>
4.1	Install . . . . .	9
4.2	Upgrade to latest version . . . . .	9
<b>5</b>	<b>Tutorial</b>	<b>11</b>
5.1	Finding the realm and org name . . . . .	11
5.2	Create the SteelConnection object . . . . .	11
5.3	Find the ID for our Organization . . . . .	11
5.4	Create a new site . . . . .	12
5.5	Set uplink to static IP . . . . .	12
5.6	Create virtual gateway . . . . .	13
5.7	Assign Port to Zone . . . . .	13
5.8	Download Virtual Gateway image . . . . .	14
5.9	Fin . . . . .	14
<b>6</b>	<b>Authentication</b>	<b>15</b>
6.1	Unattended Mode . . . . .	15
6.1.1	Specifying authentication credentials . . . . .	15
6.1.2	Using environment variables . . . . .	15
6.1.3	Using a .netrc file . . . . .	16
6.2	Interactive login . . . . .	16
6.2.1	Connection attempts . . . . .	16
<b>7</b>	<b>API Guide</b>	<b>17</b>
7.1	Realms and Organizations . . . . .	17
7.2	Understanding the API . . . . .	17

7.3	Available Methods . . . . .	17
7.4	A Tale of Two APIs . . . . .	18
7.5	Crafting your API calls . . . . .	18
7.5.1	Model Schema (Data Payload): . . . . .	19
7.6	Retrieving Data . . . . .	19
<b>8</b>	<b>Errors and Exceptions</b>	<b>21</b>
8.1	Specific Exceptions: . . . . .	21
8.2	Alternate Error Behavior . . . . .	21
<b>9</b>	<b>Logging</b>	<b>23</b>
<b>10</b>	<b>Convenience Functions</b>	<b>25</b>
10.1	Lookup . . . . .	25
10.1.1	Lookup Organization . . . . .	25
10.1.2	Lookup Node . . . . .	26
10.1.3	Lookup Site . . . . .	26
10.1.4	Lookup WAN . . . . .	26
10.1.5	Lookup Model . . . . .	26
10.2	Virtual Appliance Image Download . . . . .	26
10.2.1	download_image . . . . .	26
10.2.2	Other Binary Data . . . . .	27
10.3	Input functions . . . . .	27
10.3.1	Get Input . . . . .	27
10.3.2	Get Username . . . . .	27
10.3.3	Get Password . . . . .	27
10.4	sshtunnel . . . . .	27
<b>11</b>	<b>Examples</b>	<b>29</b>
11.1	create_site.py . . . . .	29
11.2	get_ports.py . . . . .	30
11.3	set_node_location.py . . . . .	31
11.4	ssh_to_appliance.py . . . . .	33
11.5	virtual_image_download.py . . . . .	35
<b>12</b>	<b>Indices and tables</b>	<b>37</b>





# CHAPTER 1

---

## License

---

### MIT License

Copyright (c) 2018-2020 Greg Mueller

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.





### 2.1 Getting Help with SteelConneciton:

For help with SteelConneciton (this Python package) open an issue on the project home page.  
<https://github.com/grelleum/SteelConnection>

If you decide to post an issue, please keep the following in mind:

- Errors are meaningless without the code that produced those errors.
- If your code does not produce the result you expect, let me know what results you expected.
- Please wrap your code in triple-backtics so that all indentation will be preserved.  
<https://guides.github.com/features/mastering-markdown/>

Here is how code and errors should be posted when opening an issue on GitHub:

```
'''  
for this in that:  
    print('triple backticks makes your code...')  
    print('...look the same as it does in your editor!')  
'''
```

### 2.2 SteelConnect CX API Reference:

The SteelConnect CX API is documented as two APIs:

Config API: Read and change configuration.

[https://support.riverbed.com/apis/scm\\_beta/scm-2.11.0/scm.config/index.html](https://support.riverbed.com/apis/scm_beta/scm-2.11.0/scm.config/index.html)

Reporting API: Get current status information.

[https://support.riverbed.com/apis/scm\\_beta/scm-2.11.0/scm.reporting/index.html](https://support.riverbed.com/apis/scm_beta/scm-2.11.0/scm.reporting/index.html)

### 3.1 Prerequisites

- Make sure the REST API is enabled on your SteelConnect CX realm before trying to access the REST API.
- Use pip to install steelconnection as shown above.

```
pip install steelconnection
```

### 3.2 Create an Object:

- Import steelconnection and create a new object.

```
import steelconnection  
sc = steelconnection.SConnect()
```

### 3.3 Quickstart

See the examples folder for sample scripts.

<https://github.com/grelleum/SteelConnection/tree/master/examples>



## CHAPTER 4

---

### Installation

---

#### 4.1 Install

```
pip install steelconnection
```

#### 4.2 Upgrade to latest version

```
pip install --upgrade steelconnection
```



In this tutorial, we are going to create a new site, configure static IP address on the uplinks and deploy a virtual gateway to that site.

### 5.1 Finding the realm and org name

When you manage an organization in the SteelConnect CX Manager via a web browser, the URL will look something like this:

`https://<REALM_FQDN>/admin/<ORG_SHORT_NAME>`

The <REALM\_FQDN> and <ORG\_SHORT\_NAME> will be unique to your setup and you will need these to follow the tutorial.

### 5.2 Create the SteelConnection object

Let's start by creating a SteelConnection object.

- Use the `steelconnection.SConnect` constructor to create the object.
- We will assign this new object to the name `sc`.
- We will provide the name of our realm, as well as the username and password we use to login.

```
import steelconnection
sc = steelconnection.SConnect('myrealm.riverbed.cc', 'admin', 'LetM3in')
```

### 5.3 Find the ID for our Organization

We need to know the Org ID for our Organization. This is easy since we got the Org short name from the SteelConnect CX URL.

The `.lookup.org` method will return a dictionary representing the org object. That dictionary will include a key called `id` that holds the org id.

```
# Replace ORG_SHORT_NAME with your Org's short name.
org_id = sc.lookup.org('ORG_SHORT_NAME')['id']
```

## 5.4 Create a new site

Create a dictionary that represents the site we want to create. At a minimum, we must specify the name, longname, city, and country.

The country specified must be in the standardized two letter format. Here is one source for these codes:

<https://www.willmaster.com/blog/misc/country-name-abbreviation.php>

If this site will reside in a timezone that is different from the Organization timezone, then you will want to specify the timezone for this site. Timezones must be provided in the same format as the ‘TZ’ column in this list:

[https://en.wikipedia.org/wiki/List\\_of\\_tz\\_database\\_time\\_zones#List](https://en.wikipedia.org/wiki/List_of_tz_database_time_zones#List)

Once we have our new site dictionary created, we can send a POST request to the resource ‘/org/<ORG\_ID>/sites’.

```
# Dictionary containing the details for the site to be created:
new_site = {
    'name': 'NYC',
    'city': 'New York',
    'country': 'US',
    'longname': 'New York test lab.',
    'timezone': 'America/New_York',
}

# Created site
resource = '/org/{}/sites'.format(org_id)
site = sc.post(resource, data=new_site)
```

The post command will return the newly created site, which we have assigned to the name ‘site’.

## 5.5 Set uplink to static IP

Here we will set the uplink to use a static IP address. When you create a new site, it new uplink will be created for that site. The site object will include a list of uplinks for that site. Since our site only has one uplink, we can access the uplink ID using index zero.

```
# Get the uplink ID from the site object, index 0.
uplink_id = site['uplinks'][0]

# Get uplink object from SteelConnect CX Manager.
uplink = sc.get('uplink/' + uplink_id)
```



Next, we will change the uplink type from 'dhcp' to 'static', and configure an IP address and default gateway. The change we are making is to the local dictionary object, so we will need to upload the changes to the SteelConnect CX Manager.

```
# Set uplink to static and define IP addresses.
uplink['type'] = 'static'
uplink['static_ip_v4'] = '172.30.12.249/29'
uplink['static_gw_v4'] = '172.30.12.254'

# Upload modified object to the SCM.
result = sc.put('uplink/' + uplink_id, data=uplink)
```

## 5.6 Create virtual gateway

Let's create a virtual gateway in the new site we have created. The virtual gateway has the model name 'yogi' so we need to specify that model, as well as the site ID.

```
# Create dictionary with minimum required information.
new_node = { 'site': site['id'], 'model': 'yogi' }

# POST request to SteelConnect CX Manager.
node = sc.post('/org/' + org_id + '/node/virtual/register', data=new_node)
```

## 5.7 Assign Port to Zone

At this point, the virtual gateway should have its first network interface assigned to the site uplink. However, no interfaces will be assigned to our LAN zones, so we will do that now, before we generate and download the virtual gateway image.

When we created the new site earlier, a network and a zone were created and associated with this site. We want to configure the zone to the third network interface on our gateway (we are reserving the second interface for other purposes, like as a second uplink or HA control port).

The site has a 'networks' key that includes the networks at that site. We need to retrieve the network object in order to get the zone ID. The zone will be assigned to the network interface.

```
# Get network ID from site.
# Since there is only one network associated to this site,
# we take the first one (index zero).
net_id = site['networks'][0]

# Retrieve the network from SteelConnect CX Manager.
network = sc.get('/network/' + net_id)

# Get Zone ID from the network object.
zone_id = network['zone']

# Now we can assign this zone to the third network interface.
# First, we get the port ID from the node.
# Note that since indexes start at zero, the third port is at index '2'.
port_id = node['ports'][2]

# Retrieve the port from SteelConnect CX Manager.
```

(continues on next page)

(continued from previous page)

```
port = sc.get('/port/' + port_id)

# Set the 'segment' key to the zone ID.
port['segment'] = zone_id

# We can also disable tagging for this port.
# It should already be disabled, since that is the default state.
port['tagged'] = 0

# Upload port to the SteelConnect CX Manager.
result = sc.put('/port/' + port_id, data=port)
```

## 5.8 Download Virtual Gateway image

SteelConnection provides a convenience method to generate and download virtual gateway images.

```
# Here we specify the destination filename as 'vgw.zip1',
# and the type of hypervisor as 'kvm'.

sc.download_image(save_as='vgw.zip', build='kvm')
```

## 5.9 Fin

This completes the tutorial. I hope this gets you on your way to productive use of SteelConnection.

# CHAPTER 6

---

## Authentication

---

SteelConnect CX REST API version 1.0 uses 'Basic Auth', which requires a username and password are required for every request made. The steelconnection object can store the username and password for you, or you can use a .netrc file as detailed below.

Authentication credentials can be prompted for interactively if not supplied, or they can be supplied at the time of object creation to prevent the interactive method. Scripts that need to run unattended should supply credentials at the time of object creation.

## 6.1 Unattended Mode

### 6.1.1 Specifying authentication credentials

```
import steelconnection

sc = steelconnection.SConnect('REALM.riverbed.cc', 'username', 'password')
```

### 6.1.2 Using environment variables

It is best practice not to hard-code authentication credentials in your scripts. One option is to use operating system environment variables.

Here is an example of using environment variables to store authentication.

```
import os
import steelconnection

username = os.environ.get('SCONUSER')
password = os.environ.get('SCONPASSWD')

sc = steelconnection.SConnect('REALM.riverbed.cc', username, password)
```

### 6.1.3 Using a .netrc file

A .netrc file can be used to store credentials on Mac, Unix, and Linux machines. .netrc is a standard way of storing login credentials for many network based servers. It works like a hosts file. Each line in .netrc specifies a hostname, along with the username and password used to access that server. The .netrc file is stored in the root of your home directory.

When using a .netrc file, steelconnection will not have your password, rather the underlying requests library will be responsible for accessing the .netrc file.

Since .netrc access performs a lookup on the ‘machine’ field, you will still need to specify the realm you want to access, and that hostname will be passed to requests without credentials. Requests will perform the lookup in the .netrc file.

On Mac or Linux, you can the commands below to setup a .netrc file, replacing REALM, USERNAME, and PASSWORD with your actual values.

```
echo "machine REALM.riverbed.cc login USERNAME password PASSWORD" >> ~/.netrc
chmod 600 ~/.netrc
```

To prevent SteelConnection from prompting for authentication credentials, you must explicitly tell SteelConnection to use the .netrc file.

```
sc = steelconnection.SConnect('REALM.riverbed.cc', use_netrc=True)
```

## 6.2 Interactive login

If you do not specify a realm, username, or password, and a .netrc file is not configured, steelconnection will interactively prompt you for your the missing information. Steelconnection will validate the login by making a ‘status’ call against the REST API.

```
>>> import steelconnection
>>> sc = steelconnection.SConnect()
Enter SteelConnect CX Manager fully qualified domain name: REALM.riverbed.cc
Enter username: admin
Enter password:
>>>
```

### 6.2.1 Connection attempts

Three connection attempts are allowed by default. After the third attempt an AuthenticationError exception will be raised. You can change the number of allowed login attempts by adding the connections\_attempts=N parameter, when creating the steelconnection object. Replace N with an interger. Setting connections\_attempts=0 will prevent the interactive login from running. This is useful in testing and may have other applications.

### 7.1 Realms and Organizations

There is a one to one relationship between a Realm and a SteelConnect CX Manager. The SteelConnect CX Manager acts as the controller for a the realm. A newly created realm would not have any organizations, otherwise a realm will have one or more organizations. Each organization within a realm acts an autonomous network system. In practice, most REST API operations are performed within a specific organization.

You normally access the SteelConnect CX Manager (SCM) using a web browser.

The URL you use includes the realm and organization that you are managing and takes the form:

`https://realm.riverbed.cc/admin/Organization`.

The Organization is case-sensistive and is also known as the organization short name, as opposed to the longname, which is more descriptive and can include spaces, and other characters.

### 7.2 Understanding the API

The Riverbed SteelConnect CX REST API allows HTTPS access to the SteelConnect CX Manager (SCM) via the use of GET, POST, PUT, and DELETE commands. SteelConneciton (this module) acts to simplify coding by providing an object that remembers your realm, version, and authentication and builds the HTTPS requests based on that information. A `requests.session` object is used to allow a single TCP connection to be re-used for all subsequent API requests.

### 7.3 Available Methods

SteelConneciton provides the `.get`, `.getstatus`, `.post`, `.put`, and `.delete` methods to simplify access to the API.

These methods will build the request to include api version, auth, etc, so you only need to specify the resource you are interested in.

- `get`: Used for retrieving information about a resource. Expect data to be returned.
- `getstatus`: Used for retrieving current status about a resource. Expect data to be returned.
- `post`: Create or deploy a new resource. Requires additional data in the payload and returns the newly created object.
- `put`: Use to edit or update some existing resource. Requires additional data in the payload.
- `delete`: Delete an existing resource.

## 7.4 A Tale of Two APIs

Riverbed divides the REST API into two APIs: \* Config: used to make configurations changes and get information about SteelConnect CX resources.

[https://support.riverbed.com/apis/scm\\_beta/scm-2.11.1/scm.config/index.html](https://support.riverbed.com/apis/scm_beta/scm-2.11.1/scm.config/index.html) \* Reporting: used to get current status information about a resource.

[https://support.riverbed.com/apis/scm\\_beta/scm-2.11.1/scm.reporting/index.html](https://support.riverbed.com/apis/scm_beta/scm-2.11.1/scm.reporting/index.html)

By nature, the Reporting API only requires the HTTP GET method, where-as the more commonly used Config API requires GET, POST, PUT and DELETE. SteelConnection combines the two APIs by implementing `.get`, `.post`, `.put`, and `.delete` methods to access to Config API and the `.getstatus` method to access the Reporting API.

For example: Calling `.get('/port/' + port)` would retrieve configuration settings on a port, where-as `.getstatus('/port/' + port)` would retrieve the actual link state, speed, duplex, etc. for that port.

## 7.5 Crafting your API calls

The Riverbed documentation describes the various REST API calls that can be made. These take the form: “*HTTP Method*” “*resource path*”.

Take the network section for example:

[https://support.riverbed.com/apis/scm\\_beta/scm-2.11.1/scm.config/index.html#!/network:](https://support.riverbed.com/apis/scm_beta/scm-2.11.1/scm.config/index.html#!/network:)

- `GET /networks` List networks.
- `GET /org/:orgid/networks` Get network for an org.
- `POST /org/:orgid/networks` Create network within an org.
- `DELETE /networks/:netid` Delete network.
- `GET /networks/:netid` Get network.
- `PUT /networks/:netid` Update a network.

Within the resource path, you may see a name preceded by a colon `:`. These are considered variables and must be replaced with an actual value. The `/networks/:netid` would require the `:netid` be replaced with the actual network ID for the network you are requesting.

SteelConnection methods mimic the HTTP Methods and accept the short form resource paths.

To update a network, the documentation lists `PUT /networks/:netid`. With the SteelConnection object, you would call the put method as `sc.put('/network/' + net_id)`. Note that the leading `/` in the resource is optional as the SteelConnection object will insert it if it is missing.

### 7.5.1 Model Schema (Data Payload):

Post (create) and Put (update) requests require additional data in the form of a payload. This gets sent to the server in the form of JSON data, however the SteelConnection object will accept either JSON data or a native Python dictionary (`isinstance(data, dict)`). The Riverbed documentation will specify the format of the data as a “Model Schema”. Not everything listed in the model schema is required. Generally, you can determine the minimum required data by checking the equivalent function in SteelConnect CX Manager web GUI.

## 7.6 Retrieving Data

The SteelConnection methods leverage the popular requests package. Methods calls always return a native Python dictionary, or a list of dictionaries, depending on the API call. The `requests.response` object will be stored as an attribute of the object (`sc.response`) so the latest response is always easily accessible. By providing the full `requests.response` object you are free to check status and see all headers.

For example, the ‘get orgs’ request should always provide a list of orgs within the realm, so we can directly assign the result as a native Python list.

```
list_of_all_orgs = sc.get('orgs')
```

Here are the rules to determine what gets returned by an API request:

- If `response.json()` is True and the ‘items’ key exists, then return a python list of `response.json()['items']`.
- If `response.json()` is True and the ‘items’ key *does not* exist, then return a python dictionary.
- If `response.json()` is False, return an empty python dictionary.





---

## Errors and Exceptions

---

The **Zen of Python** states:

*Errors should never pass silently.*

*Unless explicitly silenced.*

With this in mind, steelconnection assumes all REST API calls should complete without error. Successful requests will return with an HTTP 200-level response. Any other response is considered a failed request and will cause steelconnection to raise either a `RuntimeError`, or a custom exception that inherits from `RuntimeError`. Exception handling can be used to catch the exception:

```
try:
    sc.put(f'node/{node_id}', data={'location': 'LAB'})
except RuntimeError as e:
    your_code_to_handle_exception(e)
```

### 8.1 Specific Exceptions:

Exception	HTTP code	Reason
<code>AuthenticationError</code>	401	Incorrect username and password.
<code>APINotEnabled</code>	502	Rest API is not enabled on Realm.
<code>BadRequest</code>	400	Tried creating a resource that already exists.
<code>InvalidResource</code>	404	Path or resource not found.
<code>ResourceGone</code>	402	Resource no longer available.

### 8.2 Alternate Error Behavior

If you prefer to have your script exit with a simple error message and no traceback, which can be confusing to users who are not programmers, you can set `on_error='exit'` when you create your `SConnect` object.

```
sc = SConnect('REALM.riverbed.cc', on_error='exit')
```

If you prefer to handle errors manually and do not want steelconnection to generate exceptions based on HTTP response code, you can set `on_error=None` when you create your `SConnect` object. The steelconnection object will evaluate as `True` after a successful request and `False` otherwise. This reflects the status of the object attribute `SConnect.response.ok`.

```
sc = SConnect('REALM.riverbed.cc', on_error=None)
```

## CHAPTER 9

---

### Logging

---

Real-time logging can be enabled by placing the following code near the top of your script.

```
import logging

logging.basicConfig(level=logging.DEBUG)
logger = logging.getLogger(__name__)
```

This will provide details on what has been sent to and received from the SteelConnect CX manager.

The code above will display logging to standard output (the screen). Here is alternative code that can be used to log to a file:

```
import logging

logging.basicConfig(
    level=logging.DEBUG,
    format="%(asctime)s [%(name)s.%(levelname)s]: %(message)s",
    filename='steelog.txt',
)
logger = logging.getLogger(__name__)
```

The inclusion of the *filename* parameter, sends it to a file, while the *format* parameter adds a timestamp to each logged message.



---

## Convenience Functions

---

Convenience methods and functions are available to accomplish common tasks.

### 10.1 Lookup

Lookup methods provide simplified ways of finding objects.

The SteelConnect CX Manager stores resources in a database with a unique identifier (id). Many API calls require that you know the id number of the resource you are interested in, which you might not know off hand. SteelConnection provides a collection of `lookup` functions to look up the resources based on known values. These functions return the actual resource.

These are the available lookup functions:

```
<object>.lookup.org(org_short_name)
<object>.lookup.node(serial)
<object>.lookup.site(site_name, org['id'])
<object>.lookup.wan(wan_name, org['id'])
<object>.lookup.model(model)
```

These functions are accessed directly from the object you created and are specific to the SteelConnect CX API.

#### 10.1.1 Lookup Organization

Many REST API calls require that you know the org id of your organization. You can provide the organization short name to the function and it will return the org object, which includes the 'id' as a field.

```
>>> org = sc.lookup.org('Spacely')
>>> org['id']
'org-Spacely-0a0b1cbadb33f34'
>>>
```

### 10.1.2 Lookup Node

Similarly, the `lookup.node` method exists to provide the node object when you supply the commonly known appliance serial number.

```
>>> node = sc.lookup.node('XN00012345ABCDEF')
>>> node['id']
'node-56f1968e222ab789'
>>>
```

### 10.1.3 Lookup Site

The site id can be found in a similar way, but since the same site name could exist in multiple organizations, the `org_id` is also required.

```
>>> site = sc.lookup.site('Skypad', orgid='org-Spacely-0a501e7f27b2c03e')
>>> site['id']
'site-Skypad-884b9071141e4bc0'
>>>
```

### 10.1.4 Lookup WAN

The site id can be found in a similar way, but since the same site name could exist in multiple organizations, the `org_id` is also required.

```
>>> wan = sc.lookup.site('MPLS', orgid='org-Spacely-0a501e7f27b2c03e')
>>> wan['id']
'wan-MPLS-f26c9eb4f80a868b'
>>>
```

### 10.1.5 Lookup Model

The `lookup.model()` method is simply a translation service to map model code names to standard model names. It can also be used to make the opposite translations:

```
>>> sc.lookup.model('panda')
'SDI-130'
>>> sc.lookup.model('SDI-1030')
'grizzly'
>>>
```

## 10.2 Virtual Appliance Image Download

### 10.2.1 download\_image

There is a convenience method `.download_image` that can be used to download a virtual appliance image file. This method will optionally request the 'build' of a virtual appliance image, when you set `build=` a vm type, such as `build=kvm` or `build=ova`. Then it will check the availability of the image file every one second until the file is found. Next it will download the file to the location specified by the `save_as=` parameter. `download_image`

will print status messages while checking the status and downloading the file. To disable status messages, include the `quiet=True` parameter. Here are some examples:

```
# Build kvm image and specify the downloads folder and filename.
sc.download_image(node['id'], save_as='Downloads/scon_vgw.zip', build='kvm')

# Build a hyperv image and download to the current directory using the default file_
↪name.
sc.download_image(node['id'], build='hyperv')

# Download an existing image to /images/ directory and suppress status updates.
sc.download_image(node['id'], save_as='/images/scon_vgw.zip', quiet=True)
```

## 10.2.2 Other Binary Data

In the event another API call returns binary data, You can access it directly through the object's `.response.content` attribute, or by calling the `.savefile(filename)` method, which will save the binary data to a file.

## 10.3 Input functions

These functions are accessed directly from the imported module and can be used independently of the SteelConnect CX API.

### 10.3.1 Get Input

`get_input(prompt)` function works with both Python 2 and Python 3 to get user input.

### 10.3.2 Get Username

`get_username(prompt)` function works with both Python 2 and Python 3 to get username.

### 10.3.3 Get Password

`get_password(prompt)` function works with both Python 2 and Python 3 to get user input. Uses `getpass` to provide discretion. Requires user to type password twice for verification.

## 10.4 sshunnel

The `sshunnel` method provides an easy way to start, stop, or restart a reverse SSH tunnel.

```
<object>.sshunnel(node_id, timeout=15, restart=False, stop=False)
```

Examples:

```
# Start ssh tunnel. Wait as long as 15 seconds for tunnel to establish.
result = <object>.sshtunnel(node_id)

# Start ssh tunnel. Increase timeout to 30 seconds for tunnel to establish.
result = <object>.sshtunnel(node_id, timeout=30)

# Stop an existing ssh tunnel.
result = <object>.sshtunnel(node_id, stop=True)

# Stop an existing ssh tunnel and re-establish tunnel.
result = <object>.sshtunnel(node_id, restart=True)
```

Returns a dictionary object with the state of the tunnel, or an empty dictionary if stop=True.



Examples scripts to get you started.

## 11.1 create\_site.py

```
# coding: utf-8

#
#
#
#
#
#
#
#
#
# SteelConnection
# Simplify access to the Riverbed SteelConnect CX REST API.
#
# https://pypi.org/project/steelconnection
# https://github.com/grelleum/SteelConnection

from __future__ import print_function
from pprint import pprint

import steelconnection

# Change the below values to match the realm and org
# as seen in the URL for your SteelConnect CX Manager.
# for example: https://realm.riverbed.cc/admin/TestLab
scm_name = "realm.riverbed.cc"
org_name = "TestLab"

# Details for the site to be created:
```

(continues on next page)

(continued from previous page)

```
new_site = {
    "name": "NYC",
    "city": "New York",
    "country": "US",
    "longname": "New York test lab.",
    "timezone": "America/New_York",
}

def main():
    # Initialize the steelconnection object.
    sc = steelconnection.SConnect(scm_name)

    # Get the org ID for your organization.
    org = sc.lookup.org(org_name)
    print("Org name: {}, Org id: {}".format(org["longname"], org["id"]))

    # API resource for posting.
    resource = "/org/{}/sites".format(org["id"])

    # Make the post request.
    result = sc.post(resource, data=new_site)

    # Display response.
    print("Response:", sc.response.status_code, sc.response.reason)
    pprint(result)

if __name__ == "__main__":
    main()
```

## 11.2 get\_ports.py

```
# coding: utf-8

#
#
#
#
#
#
#
#
#
# SteelConnection
# Simplify access to the Riverbed SteelConnect CX REST API.
#
# https://pypi.org/project/steelconnection
# https://github.com/grelleum/SteelConnection

"""List port information for a SteelConnect CX appliance."""

from __future__ import print_function

import steelconnection
```

(continues on next page)



(continued from previous page)

```

for those nodes where the location is unset.

Works with both Python2 and Python3.

USAGE:
    set_node_location.py REALM.riverbed.cc organization
    set_node_location.py REALM.riverbed.cc organization -u $USER -p $PASSWD
"""

from __future__ import print_function
import argparse
import sys
import steelconnection

def main(argv):
    """Update nodes."""
    args = arguments(argv)

    realm, organization = args.realm, args.organization
    if organization.endswith(".cc") and not realm.endswith(".cc"):
        realm, organization = organization, realm

    sc = steelconnection.SConnect(realm, username=args.username, password=args.
↪password)

    # Find the target organization.
    org = sc.lookup.org(organization)
    print("\nOrg:", organization, "\tID:", org["id"])

    # Get list of all sites in target organization.
    sites = sc.get("org/{}/sites".format(org["id"]))
    print(status("site", sites, "in '{}'.format(organization)))

    # Create a map of site id to site name.
    site_names = {site["id"]: site["name"] for site in sites}

    # Get list of all nodes in target organization.
    nodes = sc.get("org/{}/nodes".format(org["id"]))
    print(status("node", nodes, "in '{}'.format(organization)))

    # Reduce list of nodes to those assigned to a site.
    nodes = [node for node in nodes if node["site"]]
    print(status("node", nodes, "assigned to a site"))

    # Reduce list of nodes to those not already assigned a location.
    nodes = [node for node in nodes if not node["location"]]
    print(status("node", nodes, "with no specified location"))

    # Update location for the remaining nodes.
    return update_nodes(nodes, sc, organization, org["id"], site_names)

def update_nodes(nodes, sc, organization, org_id, site_names):
    """Loop through nodes and push location to SCM where applicable."""
    for node in nodes:

```

(continues on next page)

(continued from previous page)

```

print("=" * 75)
print("Node:", node["id"], node["serial"], node["model"])
print("org:", node["org"], organization)
print("site:", node["site"])
print("location:", node["location"])

site_id = node["site"]
site_name = site_names[site_id]
print("\nSetting location to '{}'.format(site_name))
node["location"] = site_name
result = sc.put("node/" + node["id"], data=node)
print("updated location:", result["location"])
print("Response:", sc.response.status_code, sc.response.reason, "\n")
print()

def status(category, values, suffix=""):
    """Return status in human-readable format."""
    size = len(values)
    pluralization = "" if size == 1 else "s"
    return "* Found {} {}{} {}".format(size, category, pluralization, suffix)

def arguments(argv):
    """Get command line arguments."""
    description = (
        "Update SteelConnect CX nodes within a specified Org "
        "by copying the site name to the location field "
        "for those nodes where the location is unset."
    )
    parser = argparse.ArgumentParser(description=description)
    parser.add_argument(
        "realm", type=str, help="Domain name of SteelConnect CX Manager"
    )
    parser.add_argument("organization", type=str, help="Name of target organization")
    parser.add_argument(
        "-u", "--username", help="Username for SteelConnect CX Manager (optional)"
    )
    parser.add_argument(
        "-p", "--password", help="Password for SteelConnect CX Manager (optional)"
    )
    return parser.parse_args()

if __name__ == "__main__":
    result = main(sys.argv[1:])

```

## 11.4 ssh\_to\_appliance.py

```

#!/usr/bin/env python3

# coding: utf-8

#

```

(continues on next page)

(continued from previous page)

[illegible]

(continues on next page)

```
client.set_missing_host_key_policy(paramiko.AutoAddPolicy())

# Make SSH connection to appliance.
client.connect(hostname=hostname, username="root", sock=sock)

# Execute the commands
for command in commands:
    stdin, stdout, stderr = client.exec_command(command)
    output = stdout.read().decode()
    print(f"# Output of '{command}'")
    print(output)

# close the connection:
client.close()

if __name__ == "__main__":
    main()
```

```
# coding: utf-8

#
#      _____
#     /  _  / /  _  _  /  /
#    ____\  \ /  _  _  /  /
#   /  _  / \  \ /  _  _  /  /  _  _
#  /  _  /  _  \ /  _  _  /  _  _  \
# \  _  /  _  / /  _  _  /  _  _  \
#
# SteelConnection
# Simplify access to the Riverbed SteelConnect CX REST API.
#
# https://pypi.org/project/steelconnection
# https://github.com/grelleum/SteelConnection

from __future__ import print_function

import steelconnection
import os

def main():
    sc = steelconnection.SConnect()
    sc.get("status")

    # steelconnection.get_input is compatible with both Python 2 and 3.
    serial = steelconnection.get_input("Enter appliance serial number: ")
    node = sc.lookup.node(serial)

    hypervisor = steelconnection.get_input("Enter the hypervisor type: ")
    filename = "scon_vgw_{ }_{ }.zip".format(serial, hypervisor)
```

(continued from previous page)

```
# Put filename into the HOME/Downloads folder.
home = os.path.expanduser("~")
filepath = os.path.join(home, "Downloads", filename)

success = sc.download_image(node["id"], save_as=filepath, build=hypervisor)
print(success)

if __name__ == "__main__":
    main()
```



## CHAPTER 12

---

### Indices and tables

---

- `genindex`
- `modindex`