SoftLayer API Python Client Documentation

Release 2.3.1

SoftLayer Technologies, Inc.

November 20, 2013

Contents

This is the documentation to SoftLayer's Python API Bindings. These bindings use SoftLayer's XML-RPC interface in order to manage SoftLayer services.

CHAPTER 1

Installation

1.1 Using Pip

Install via pip:

```
$ pip install softlayer
```

Install from source gia pip (requires git):

```
$ pip install git+git://github.com/softlayer/softlayer-api-python-client.git
```

The most up to date version of this library can be found on the SoftLayer GitHub public repositories: https://github.com/softlayer. Please post to the SoftLayer forums https://forums.softlayer.com/ or open a support ticket in the SoftLayer customer portal if you have any questions regarding use of this library.

1.2 From Source

The project is developed on GitHub, at github.com/softlayer/softlayer-api-python-client.

You can clone the public repository:

```
$ git clone git://github.com/softlayer/softlayer-api-python-client.git
```

Or, Download the tarball:

```
$ curl -OL https://github.com/softlayer/softlayer-api-python-client/tarball/master
```

Or, download the zipball:

```
$ curl -OL https://github.com/softlayer/softlayer-api-python-client/zipball/master
```

Once you have a copy of the source, you can embed it in your Python package, or install it into your site-packages easily:

```
$ python setup.py install
```

CHAPTER 2

API Documentation

```
>>> import SoftLayer
>>> client = SoftLayer.Client(username="username", api_key="api_key")
>>> resp = client['Account'].getObject()
>>> resp['companyName']
'Your Company'
```

2.1 Developer Interface

This is the primary API client to make API calls. It deals with constructing and executing XML-RPC calls against the SoftLayer API.

2.1.1 Getting Started

You can pass in your username and api_key when creating a SoftLayer client instance. However, you can set these in the environmental variables 'SL_USERNAME' and 'SL_API_KEY'

Creating a client instance by passing in the username/api_key:

```
import SoftLayer
client = SoftLayer.Client(username='YOUR_USERNAME', api_key='YOUR_API_KEY')
```

Creating a client instance with environmental variables set:

```
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()
```

Below is an example of creating a client instance with more options. This will create a client with the private API endpoint (only accessable from the SoftLayer network), a timeout of 2 minutes, and with verbose mode on (prints out more than you ever wanted to know about the HTTP requests to stdout).

```
client = SoftLayer.Client(
        username='YOUR_USERNAME',
        api_key='YOUR_API_KEY'
```

```
        endpoint_url=SoftLayer.API_PRIVATE_ENDPOINT,
        timeout=240,
        verbose=True,
    )
```

2.1.2 Making API Calls

The SoftLayer API client for python leverages SoftLayer's XML-RPC API. It supports authentication, object masks, object filters, limits, offsets, and retrieving objects by id. The following section assumes you have a initialized client named 'client'.

The best way to test our setup is to call the getObject method on the SoftLayer_Account service.

```
client['Account'].getObject()
```

For a more complex example we'll retrieve a support ticket with id 123456 along with the ticket's updates, the user it's assigned to, the servers attached to it, and the datacenter those servers are in. To retrieve our extra information using an object mask.

Retreive a ticket using Object Masks.

```
ticket = client['Ticket'].getObject(
    id=123456, mask="mask[updates, assignedUser, attachedHardware.datacenter]")
```

Now add an update to the ticket with Ticket.addUpdate. This uses a parameter, which translate to positional arguments in the order that they appear in the API docs.

```
update = client['Ticket'].addUpdate({'entry' : 'Hello!'}, id=123456)
```

Let's get a listing of virtual guests using the domain example.com

```
client['Account'].getVirtualGuests(
    filter={'virtualGuests': {'domain': {'operation': 'example.com'}}})
```

This call gets tickets created between the beginning of March 1, 2013 and March 15, 2013.

```
client['Account'].getTickets(
    filter={
        'tickets': {
            'createDate': {
                'operation': 'betweenDate',
                'options': [
                    {'name': 'startDate', 'value': ['03/01/2013 0:0:0']},
                    {'name': 'endDate', 'value': ['03/15/2013 23:59:59']}
                ]
            }
        }
    }
)
```

SoftLayer's XML-RPC API also allows for pagination.

```
client['Account'].getVirtualGuests(limit=10, offset=0)  # Page 1
client['Account'].getVirtualGuests(limit=10, offset=10)  # Page 2
```

Here's how to create a new Cloud Compute Instance using SoftLayer_Virtual_Guest.createObject. Be warned, this call actually creates an hourly CCI so this does have billing implications.

```
client['Virtual_Guest'].createObject({
        'hostname': 'myhostname',
        'domain': 'example.com',
        'startCpus': 1,
        'maxMemory': 1024,
        'hourlyBillingFlag': 'true',
        'operatingSystemReferenceCode': 'UBUNTU_LATEST',
        'localDiskFlag': 'false'
    })
```

2.1.3 API Reference

class SoftLayer.**Client**(*service_name=None*, *id=None*, *username=None*, *api_key=None*, *endpoint_url=None*, *timeout=None*, *auth=None*)

A SoftLayer API client.

Parameters

- **service_name** – the name of the SoftLayer API service to query
- **id** (*integer*) – an optional object ID if you're instantiating a particular SoftLayer_API object. Setting an ID defines this client's initialization parameter.
- **username** – an optional API username if you wish to bypass the package's built-in username
- **api_key** – an optional API key if you wish to bypass the package's built in API key
- **endpoint_url** – the API endpoint base URL you wish to connect to. Set this to API_PRIVATE_ENDPOINT to connect via SoftLayer's private network.
- **timeout** (*integer*) – timeout for API requests
- **auth** – an object which responds to get_headers() to be inserted into the xml-rpc headers. Example: *BasicAuthentication*

Usage:

```
>>> import SoftLayer
>>> client = SoftLayer.Client(username="username", api_key="api_key")
>>> resp = client['Account'].getObject()
>>> resp['companyName']
'Your Company'
```

__getitem__(*name*)

Get a SoftLayer Service.

Parameters **name** – The name of the service. E.G. Account

Usage:

```
>>> client = SoftLayer.Client()
>>> client['Account']
<Service: Account>
```

authenticate_with_password(*username*, *password*, *security_question_id=None*, *security_question_answer=None*)

Performs Username/Password Authentication and gives back an auth handler to use to create a client that uses token-based auth.

Parameters

- **username** (*string*) – your SoftLayer username
- **password** (*string*) – your SoftLayer password
- **security_question_id** (*int*) – The security question id to answer
- **security_question_answer** (*string*) – The answer to the security question

call(*service*, *method*, **args*, ***kwargs*)
Make a SoftLayer API call

Parameters

- **service** – the name of the SoftLayer API service
- **method** – the method to call on the service
- ***args** – same optional arguments that `Service.call` takes
- ****kwargs** – same optional keyword arguments that `Service.call` takes
- **service** – the name of the SoftLayer API service

Usage:

```
>>> client = SoftLayer.Client()
>>> client['Account'].getVirtualGuests(mask="id", limit=10)
[...]
```

iter_call(*service*, *method*, *chunk=100*, *limit=None*, *offset=0*, **args*, ***kwargs*)
A generator that deals with paginating through results.

Parameters

- **service** – the name of the SoftLayer API service
- **method** – the method to call on the service
- **chunk** (*integer*) – result size for each API call
- ***args** – same optional arguments that `Service.call` takes
- ****kwargs** – same optional keyword arguments that `Service.call` takes

class SoftLayer.API.**Service**(*client*, *name*)

__call__(*name*, **args*, ***kwargs*)
Make a SoftLayer API call

Parameters

- **method** – the method to call on the service
- ***args** – (optional) arguments for the remote call
- **id** – (optional) id for the resource
- **mask** – (optional) object mask
- **filter** (*dict*) – (optional) filter dict
- **headers** (*dict*) – (optional) optional XML-RPC headers
- **raw_headers** (*dict*) – (optional) HTTP transport headers
- **limit** (*int*) – (optional) return at most this many results

- **offset** (*int*) – (optional) offset results by this many
- **iter** (*boolean*) – (optional) if True, returns a generator with the results

Usage:

```
>>> client['Account'].getVirtualGuests(mask="id", limit=10)
[...]
```

call(*name*, **args*, ***kwargs*)
Make a SoftLayer API call

Parameters

- **method** – the method to call on the service
- ***args** – (optional) arguments for the remote call
- **id** – (optional) id for the resource
- **mask** – (optional) object mask
- **filter** (*dict*) – (optional) filter dict
- **headers** (*dict*) – (optional) optional XML-RPC headers
- **raw_headers** (*dict*) – (optional) HTTP transport headers
- **limit** (*int*) – (optional) return at most this many results
- **offset** (*int*) – (optional) offset results by this many
- **iter** (*boolean*) – (optional) if True, returns a generator with the results

Usage:

```
>>> client['Account'].getVirtualGuests(mask="id", limit=10)
[...]
```

iter_call(*name*, **args*, ***kwargs*)
A generator that deals with paginating through results.

Parameters

- **method** – the method to call on the service
- **chunk** (*integer*) – result size for each API call
- ***args** – same optional arguments that `Service.call` takes
- ****kwargs** – same optional keyword arguments that `Service.call` takes

Usage:

```
>>> gen = client['Account'].getVirtualGuests(iter=True)
>>> for virtual_guest in gen:
...     virtual_guest['id']
...
1234
4321
```

SoftLayer.exceptions

Exceptions used throughout the library

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

exception SoftLayer.exceptions.**ApplicationError**(*faultCode*, *faultString*, **args*)
Application Error

exception SoftLayer.exceptions.**DNSZoneNotFound**

exception SoftLayer.exceptions.**InternalError**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**InvalidCharacter**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**InvalidMethodParameters**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**MethodNotFound**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**NotWellFormed**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**ParseError**(*faultCode*, *faultString*, **args*)
Parse Error

exception SoftLayer.exceptions.**RemoteSystemError**(*faultCode*, *faultString*, **args*)
System Error

exception SoftLayer.exceptions.**ServerError**(*faultCode*, *faultString*, **args*)
Server Error

exception SoftLayer.exceptions.**SoftLayerAPIError**(*faultCode*, *faultString*, **args*)
SoftLayerAPIError is an exception raised whenever an error is returned from the API.

Provides faultCode and faultString properties.

exception SoftLayer.exceptions.**SoftLayerError**
The base SoftLayer error.

exception SoftLayer.exceptions.**SpecViolation**(*faultCode*, *faultString*, **args*)

exception SoftLayer.exceptions.**TransportError**(*faultCode*, *faultString*, **args*)
Transport Error

exception SoftLayer.exceptions.**Unauthenticated**
Unauthenticated

exception SoftLayer.exceptions.**UnsupportedEncoding**(*faultCode*, *faultString*, **args*)

2.1.4 Backwards Compatibility

If you've been using the older Python client (<2.0), you'll be happy to know that the old API is still currently working. However, you should deprecate use of the old stuff. Below is an example of the old API converted to the new one.

SoftLayer.deprecated

This is where deprecated APIs go for their eternal slumber

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.deprecated.**DeprecatedClientMixin**(*id=None*, *username=None*, *api_key=None*, ***kwargs*)

This mixin is to be used in SoftLayer.Client so all of these methods should be available to the client but are all deprecated.

add_header(*name*, *value*)

Set a SoftLayer API call header.

Parameters

- **name** – the header name
- **value** – the header value

Deprecated since version 2.0.0.

add_raw_header(*name*, *value*)

Set HTTP headers for API calls.

Parameters

- **name** – the header name
- **value** – the header value

Deprecated since version 2.0.0.

remove_header(*name*)

Remove a SoftLayer API call header.

Parameters **name** – the header name

Deprecated since version 2.0.0.

set_authentication(*username*, *api_key*)

Set user and key to authenticate a SoftLayer API call.

Use this method if you wish to bypass the API_USER and API_KEY class constants and set custom authentication per API call.

See https://manage.softlayer.com/Administrative/apiKeychain for more information.

Parameters

- **username** – the username to authenticate with
- **api_key** – the user's API key

Deprecated since version 2.0.0.

set_init_parameter(*id*)

Set an initialization parameter header.

Initialization parameters instantiate a SoftLayer API service object to act upon during your API method call. For instance, if your account has a server with ID number 1234, then setting an initialization parameter of 1234 in the SoftLayer_Hardware_Server Service instructs the API to act on server record 1234 in your method calls.

See http://sldn.softlayer.com/article/Using-Initialization-Parameters-SoftLayer-API # NOQA for more information.

Parameters **id** – the ID of the SoftLayer API object to instantiate

Deprecated since version 2.0.0.

set_object_mask (*mask*)
Set an object mask to a SoftLayer API call.

Use an object mask to retrieve data related your API call's result. Object masks are skeleton objects, or strings that define nested relational properties to retrieve along with an object's local properties. See http://sldn.softlayer.com/article/Using-Object-Masks-SoftLayer-API for more information.

Parameters **mask** – the object mask you wish to define

Deprecated since version 2.0.0.

set_result_limit (*limit*, *offset=0*)
Set a result limit on a SoftLayer API call.

Many SoftLayer API methods return a group of results. These methods support a way to limit the number of results retrieved from the SoftLayer API in a way akin to an SQL LIMIT statement.

Parameters

- **limit** – the number of results to limit a SoftLayer API call to
- **offset** – An optional offset at which to begin a SoftLayer API call's returned result

Deprecated since version 2.0.0.

```
import SoftLayer.API
client = SoftLayer.API.Client('SoftLayer_Account', None, 'username', 'api_key')
client.set_object_mask({'ipAddresses' : None})
client.set_result_limit(10, offset=10)
client.getObject()
```

... changes to ...

```
import SoftLayer
client = SoftLayer.Client(username='username', api_key='api_key')
client['Account'].getObject(mask="mask[ipAddresses]", limit=10, offset=0)
```

Deprecated APIs

Below are examples of how the old usages to the new API.

Importing the module

```
# Old
import SoftLayer.API

# New
import SoftLayer
```

Creating a client instance

```
# Old
client = SoftLayer.API.Client('SoftLayer_Account', None, 'username', 'api_key')

# New
client = SoftLayer.Client(username='username', api_key='api_key')
service = client['Account']
```

Making an API call

```
# Old
client = SoftLayer.API.Client('SoftLayer_Account', None, 'username', 'api_key')
client.getObject()

# New
client = SoftLayer.Client(username='username', api_key='api_key')
client['Account'].getObject()

# Optionally
service = client['Account']
service.getObject()
```

Setting Object Mask

```
# Old
client.set_object_mask({'ipAddresses' : None})

# New
client['Account'].getObject(mask="mask[ipAddresses]")
```

Using Init Parameter

```
# Old
client.set_init_parameter(1234)

# New
client['Account'].getObject(id=1234)
```

Setting Result Limit and Offset

```
# Old
client.set_result_limit(10, offset=10)

# New
client['Account'].getObject(limit=10, offset=10)
```

Adding Additional Headers

```
# Old
# These headers are persisted accross API calls
client.add_header('header', 'value')

# New
# These headers are NOT persisted accross API calls
client['Account'].getObject(headers={'header': 'value'})
```

Removing Additional Headers

```
# Old
client.remove_header('header')

# New
client['Account'].getObject()
```

Adding Additional HTTP Headers

```
# Old
client.add_raw_header('header', 'value')
```

```
# New
client['Account'].getObject(raw_headers={'header': 'value'})
```

Changing Authentication Credentials

```
# Old
client.set_authentication('username', 'api_key')

# New
client.username = 'username'
client.api_key = 'api_key'
```

2.2 Managers

```
>>> from SoftLayer import CCIManager, Client
>>> client = Client(...)
>>> cci = CCIManager(client)
>>> cci.list_instances()
[...]
```

Managers mask out a lot of the complexities of using the API into classes that provide a simpler interface to various services. These are higher-level interfaces to the SoftLayer API.

2.2.1 SoftLayer.CCI

CCI Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.cci.**CCIManager**(*client*)
Manage CCIs

cancel_instance(*id*)
Cancel an instance immediately, deleting all its data.

Parameters **id** (*integer*) – the instance ID to cancel

change_port_speed(*id*, *public*, *speed*)

create_instance(***kwargs*)
see _generate_create_dict

edit(*id*, *userdata=None*, *hostname=None*, *domain=None*, *notes=None*)
Edit hostname, domain name, notes, and/or the user data of a CCI

Parameters set to None will be ignored and not attempted to be updated.

Parameters

- **id** (*integer*) – the instance ID to edit
- **userdata** (*string*) – user data on CCI to edit. If none exist it will be created
- **hostname** (*string*) – valid hostname
- **domain** (*string*) – valid domain namem

- **notes** (*string*) – notes about this particular CCI

get_create_options()

get_instance(*id*, ***kwargs*)
Get details about a CCI instance

Parameters **id** (*integer*) – the instance ID

list_instances(*hourly=True*, *monthly=True*, *tags=None*, *cpus=None*, *memory=None*, *hostname=None*, *domain=None*, *local_disk=None*, *datacenter=None*, *nic_speed=None*, *public_ip=None*, *private_ip=None*, ***kwargs*)
Retrieve a list of all CCIs on the account.

Parameters

- **hourly** (*boolean*) – include hourly instances
- **monthly** (*boolean*) – include monthly instances
- **tags** (*list*) – filter based on tags
- **cpus** (*integer*) – filter based on number of CPUS
- **memory** (*integer*) – filter based on amount of memory
- **hostname** (*string*) – filter based on hostname
- **domain** (*string*) – filter based on domain
- **local_disk** (*string*) – filter based on local_disk
- **datacenter** (*string*) – filter based on datacenter
- **nic_speed** (*integer*) – filter based on network speed (in MBPS)
- **public_ip** (*string*) – filter based on public ip address
- **private_ip** (*string*) – filter based on private ip address
- ****kwargs** (*dict*) – response-level arguments (limit, offset, etc.)

reload_instance(*id*, *post_uri=None*)
Perform an OS reload of an instance with its current configuration.

Parameters

- **id** (*integer*) – the instance ID to reload
- **post_url** (*string*) – The URI of the post-install script to run after reload

resolve_ids(*identifier*)
Takes a string and tries to resolve to a list of matching ids. What exactly 'identifier' can be depends on the resolvers

Parameters **identifier** (*string*) – identifying string

Returns list

resolvers = []

verify_create_instance(***kwargs*)
see _generate_create_dict

wait_for_transaction(*id*, *limit*, *delay=1*)

2.2.2 SoftLayer.DNS

DNS Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.dns.**DNSManager**(*client*)
Manage DNS zones.

create_record(*zone_id*, *record*, *type*, *data*, *ttl=60*)
Create a resource record on a domain.

Parameters

- **id** (*integer*) – the zone's ID
- **record** – the name of the record to add
- **type** – the type of record (A, AAAA, CNAME, MX, SRV, TXT, etc.)
- **data** – the record's value
- **ttl** (*integer*) – the TTL or time-to-live value (default: 60)

create_zone(*zone*, *serial=None*)
Create a zone for the specified zone.

Parameters

- **zone** – the zone name to create
- **serial** – serial value on the zone (default: strftime(%Y%m%d01))

delete_record(*record_id*)
Delete a resource record by its ID.

Parameters **id** (*integer*) – the record's ID

delete_zone(*id*)
Delete a zone by its ID.

Parameters **id** (*integer*) – the zone ID to delete

dump_zone(*zone_id*)
Retrieve a zone dump in BIND format.

Parameters **id** (*integer*) – The zone ID to dump

edit_record(*record*)
Update an existing record with the options provided. The provided dict must include an 'id' key and value corresponding to the record that should be updated.

Parameters **record** (*dict*) – the record to update

edit_zone(*zone*)
Update an existing zone with the options provided. The provided dict must include an 'id' key and value corresponding to the zone that should be updated.

Parameters **zone** (*dict*) – the zone to update

get_records(*zone_id*, *ttl=None*, *data=None*, *host=None*, *type=None*, ***kwargs*)
List, and optionally filter, records within a zone.

Parameters

- **zone** – the zone name in which to search.
- **ttl** (*int*) – optionally, time in seconds:
- **data** – optionally, the records data
- **host** – optionally, record's host
- **type** – optionally, the type of record:

Returns list

get_zone(*zone_id*, *records=True*)
Get a zone and its records.

Parameters **zone** – the zone name

list_zones(***kwargs*)
Retrieve a list of all DNS zones.

Parameters ****kwargs** (*dict*) – response-level arguments (limit, offset, etc.)

resolve_ids(*identifier*)
Takes a string and tries to resolve to a list of matching ids. What exactly 'identifier' can be depends on the resolvers

Parameters **identifier** (*string*) – identifying string

Returns list

resolvers = []

2.2.3 SoftLayer.firewall

Firewall Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.firewall.**FirewallManager**(*client*)

get_firewalls()

SoftLayer.managers.firewall.**has_firewall**(*vlan*)

2.2.4 SoftLayer.hardware

Hardware Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.hardware.**HardwareManager**(*client*)
Manages hardware devices.

cancel_hardware(*id*, *reason='unneeded'*, *comment=''*)
Cancels the specified dedicated server.

Parameters

- **id** (*int*) – The ID of the hardware to be cancelled.
- **immediate** (*bool*) – If true, the hardware will be cancelled immediately. Otherwise, it will be scheduled to cancel on the anniversary date.
- **reason** (*string*) – The reason code for the cancellation.

cancel_metal(*id*, *immediate=False*)
Cancels the specified bare metal instance.

Parameters

- **id** (*int*) – The ID of the bare metal instance to be cancelled.
- **immediate** (*bool*) – If true, the bare metal instance will be cancelled immediately. Otherwise, it will be scheduled to cancel on the anniversary date.

change_port_speed(*id*, *public*, *speed*)
Allows you to change the port speed of a server's NICs.

Parameters

- **id** (*int*) – The ID of the server
- **public** (*bool*) – Flag to indicate which interface to change. True (default) means the public interface. False indicates the private interface.
- **speed** (*int*) – The port speed to set.

edit(*id*, *userdata=None*, *hostname=None*, *domain=None*, *notes=None*)
Edit hostname, domain name, notes, and/or the user data of the hardware

Parameters set to None will be ignored and not attempted to be updated.

Parameters

- **id** (*integer*) – the instance ID to edit
- **userdata** (*string*) – user data on the hardware to edit. If none exist it will be created
- **hostname** (*string*) – valid hostname
- **domain** (*string*) – valid domain namem
- **notes** (*string*) – notes about this particular hardware

get_available_dedicated_server_packages()
Retrieves a list of packages that are available for ordering dedicated servers.

Note - This currently returns a hard coded list until the API is updated to allow filtering on packages to just those for ordering servers.

get_bare_metal_create_options()
Retrieves the available options for creating a bare metal server.

The information for ordering bare metal instances comes from multiple API calls. In order to make the process easier, this function will make those calls and reformat the results into a dictionary that's easier to manage. It's recommended that you cache these results with a reasonable lifetime for performance reasons.

get_cancellation_reasons()

get_dedicated_server_create_options(*package_id*)
Retrieves the available options for creating a dedicated server in a specific chassis (based on package ID).

The information for ordering dedicated servers comes from multiple API calls. In order to make the process easier, this function will make those calls and reformat the results into a dictionary that's easier to manage. It's recommended that you cache these results with a reasonable lifetime for performance reasons.

get_hardware(*id*, ***kwargs*)
Get details about a hardware device

Parameters **id** (*integer*) – the hardware ID

list_hardware(*tags=None*, *cpus=None*, *memory=None*, *hostname=None*, *domain=None*, *datacenter=None*, *nic_speed=None*, *public_ip=None*, *private_ip=None*, ***kwargs*)
List all hardware.

Parameters

- **tags** (*list*) – filter based on tags
- **cpus** (*integer*) – filter based on number of CPUS
- **memory** (*integer*) – filter based on amount of memory in gigabytes
- **hostname** (*string*) – filter based on hostname
- **domain** (*string*) – filter based on domain
- **datacenter** (*string*) – filter based on datacenter
- **nic_speed** (*integer*) – filter based on network speed (in MBPS)
- **public_ip** (*string*) – filter based on public ip address
- **private_ip** (*string*) – filter based on private ip address
- ****kwargs** (*dict*) – response-level arguments (limit, offset, etc.)

place_order(***kwargs*)
Places an order for a piece of hardware. See _generate_create_dict for a list of available options.

reload(*id*, *post_uri=None*)
Perform an OS reload of a server with its current configuration.

Parameters

- **id** (*integer*) – the instance ID to reload
- **post_url** (*string*) – The URI of the post-install script to run after reload

resolve_ids(*identifier*)
Takes a string and tries to resolve to a list of matching ids. What exactly 'identifier' can be depends on the resolvers

Parameters **identifier** (*string*) – identifying string

Returns list

resolvers = []

verify_order(***kwargs*)
Verifies an order for a piece of hardware without actually placing it. See _generate_create_dict for a list of available options.

SoftLayer.managers.hardware.**get_default_value**(*package_options*, *category*)

2.2.5 SoftLayer.messaging

class SoftLayer.managers.messaging.**MessagingConnection**(*id*, *endpoint=None*)
Message Queue Service Connection

Parameters

- **id** – Message Queue Account id
- **endpoint** – Endpoint URL

authenticate(*username*, *api_key*, *auth_token=None*)
Make request. Generally not called directly

Parameters

- **username** – SoftLayer username
- **api_key** – SoftLayer API Key
- **auth_token** – (optional) Starting auth token

create_queue(*queue_name*, ***kwargs*)
Create Queue

Parameters

- **queue_name** – Queue Name
- ****kwargs** (*dict*) – queue options

create_subscription(*topic_name*, *type*, ***kwargs*)
Create Subscription

Parameters

- **topic_name** – Topic Name
- **type** – type ('queue' or 'http')
- ****kwargs** (*dict*) – Subscription options

create_topic(*topic_name*, ***kwargs*)
Create Topic

Parameters

- **topic_name** – Topic Name
- ****kwargs** (*dict*) – Topic options

delete_message(*queue_name*, *message_id*)
Delete a message

Parameters

- **queue_name** – Queue Name
- **message_id** – Message id

delete_queue(*queue_name*, *force=False*)
Delete Queue

Parameters

- **queue_name** – Queue Name
- **force** – (optional) Force queue to be deleted even if there are pending messages

delete_subscription(*topic_name*, *subscription_id*)
Delete a subscription

Parameters

- **topic_name** – Topic Name
- **subscription_id** – Subscription id

delete_topic(*topic_name*, *force=False*)
Delete Topic

Parameters

- **topic_name** – Topic Name
- **force** – (optional) Force topic to be deleted even if there are attached subscribers

get_queue(*queue_name*)
Get queue details

Parameters **queue_name** – Queue Name

get_queues(*tags=None*)
Get listing of queues

Parameters **tags** (*list*) – (optional) list of tags to filter by

get_subscriptions(*topic_name*)
Listing of subscriptions on a topic

Parameters **topic_name** – Topic Name

get_topic(*topic_name*)
Get topic details

Parameters **topic_name** – Topic Name

get_topics(*tags=None*)
Get listing of topics

Parameters **tags** (*list*) – (optional) list of tags to filter by

modify_queue(*queue_name*, ***kwargs*)
Modify Queue

Parameters

- **queue_name** – Queue Name
- ****kwargs** (*dict*) – queue options

modify_topic(*topic_name*, ***kwargs*)
Modify Topic

Parameters

- **topic_name** – Topic Name
- ****kwargs** (*dict*) – Topic options

pop_message(*queue_name*, *count=1*)
Pop message from a queue

Parameters

- **queue_name** – Queue Name

- **count** – (optional) number of messages to retrieve

push_queue_message(*queue_name*, *body*, ***kwargs*)
Create Queue Message

Parameters

- **queue_name** – Queue Name
- **body** – Message body
- ****kwargs** (*dict*) – Message options

push_topic_message(*topic_name*, *body*, ***kwargs*)
Create Topic Message

Parameters

- **topic_name** – Topic Name
- **body** – Message body
- ****kwargs** (*dict*) – Topic message options

stats(*period='hour'*)
Get account stats

Parameters **period** – 'hour', 'day', 'week', 'month'

class SoftLayer.managers.messaging.**MessagingManager**(*client*)
Manage SoftLayer Message Queue

get_connection(*id*, *username*, *api_key*, *datacenter=None*, *network=None*)
Get connection to Message Queue Service

Parameters

- **id** – Message Queue Account id
- **username** – SoftLayer username
- **api_key** – SoftLayer API key
- **datacenter** – Datacenter code
- **network** – network ('public' or 'private')

get_endpoint(*datacenter=None*, *network=None*)
Get a message queue endpoint based on datacenter/network type

Parameters

- **datacenter** – datacenter code
- **network** – network ('public' or 'private')

get_endpoints()
Get all known message queue endpoints

list_accounts(***kwargs*)
List message queue accounts

Parameters ****kwargs** (*dict*) – response-level arguments (limit, offset, etc.)

ping(*datacenter=None*, *network=None*)

class SoftLayer.managers.messaging.**QueueAuth**(*endpoint*, *username*, *api_key*, *auth_token=None*)
Message Queue authentication for requests

Parameters

- **endpoint** – endpoint URL
- **username** – SoftLayer username
- **api_key** – SoftLayer API Key
- **auth_token** – (optional) Starting auth token

auth()
Do Authentication

handle_error(*r*, ***kwargs*)
Handle errors

2.2.6 SoftLayer.metadata

Metadata Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.metadata.**MetadataManager**(*client=None*, *timeout=5*)
Provides an interface for the metadata service. This provides metadata about the resourse it is called from. See *METADATA_ATTRIBUTES* for full list of attributes.

Usage:

```
>>> from SoftLayer.metadata import MetadataManager
>>> meta = MetadataManager(client)
>>> meta.get('datacenter')
'dal05'
>>> meta.get('fqdn')
'test.example.com'
```

attribs = {'datacenter': {'call': 'Datacenter'}, 'domain': {'call': 'Domain'}, 'backend_mac': {'call': 'BackendMacAdd

get(*name*, *param=None*)
Retreive a metadata attribute

Parameters

- **name** – name of the attribute to retrieve. See *attribs*
- **param** – Required parameter for some attributes

make_request(*path*)

private_network(***kwargs*)
Returns details about the private network

Parameters

- **router** (*boolean*) – True to return router details
- **vlans** (*boolean*) – True to return vlan details
- **vlan_ids** (*boolean*) – True to return vlan_ids

public_network(***kwargs*)
Returns details about the public network

Parameters

- **router** (*boolean*) – True to return router details
- **vlans** (*boolean*) – True to return vlan details
- **vlan_ids** (*boolean*) – True to return vlan_ids

metadata.**METADATA_ATTRIBUTES = ['datacenter', 'domain', 'backend_mac', 'primary_ip', 'primary_backend_ip', 'tags',**

2.2.7 SoftLayer.SSL

SSL Manager/helpers

copyright

3. 2013, SoftLayer Technologies, Inc. All rights reserved.

license BSD, see LICENSE for more details.

class SoftLayer.managers.ssl.**SSLManager**(*client*)
Manages SSL certificates.

add_certificate(*certificate*)
Creates a new certificate.

Parameters **certificate** – # TODO: is this a dict?

edit_certificate(*certificate*)
Updates a certificate with the included options. The provided dict must include an 'id' key and value corresponding to the certificate ID that should be updated.

Parameters **certificate** (*dict*) – the certificate to update.

get_certificate(*id*)
Gets a certificate with the ID specified.

Parameters **id** (*integer*) – the certificate ID to retrieve

list_certs(*method='all'*)
List all certificates.

Parameters **method** – # TODO: explain this param

remove_certificate(*id*)
Removes a certificate.

Parameters **id** (*integer*) – a certificate ID to remove

CHAPTER 3

Command-Line Interface

```
$ sl cci list
:.........:...........:..................:.......:........:................:..............:.......
:    id   : datacenter :       host       : cores : memory :   primary_ip   :  backend_ip  : active
:.........:...........:..................:.......:........:................:..............:.......
: 1234567 :   dal05    :  test.example.com  :   4   :   4G   :    12.34.56    :   65.43.21   :
:.........:...........:..................:.......:........:................:..............:.......
```

3.1 Command-line Interface

The SoftLayer command line interface is available via the *sl* command available in your *PATH*. The *sl* command is a reference implementation of SoftLayer API bindings for python and how to efficiently make API calls.

3.1.1 Configuration Setup

To check the configuration, you can use *sl config show*.

```
$ sl config setup
Username: username
API Key: oyVmeipYQCNrjVS4rF9bHWV7D75S6pa1fghFl384v7mwRCbHTfuJ8qRORIqoVnha
Endpoint URL [https://api.softlayer.com/xmlrpc/v3/]:
Are you sure you want to write settings to "/path/to/home/.softlayer"? [y/N]: y
```

To check the configuration, you can use *sl config show*.

```
$ sl config show
:.............:..............................................................:
:         Name : Value                                                      :
:.............:..............................................................:
:     Username : username                                                   :
:      API Key : oyVmeipYQCNrjVS4rF9bHWV7D75S6pa1fghFl384v7mwRCbHTfuJ8qRORIqoVnha :
: Endpoint URL : https://api.softlayer.com/xmlrpc/v3/                       :
:.............:..............................................................:
```

3.1.2 Configuration File Details

The CLI loads your settings from a number of different locations.

- Enviorment variables (SL_USERNAME, SL_API_KEY)
- Config file (~/.softlayer)
- Or argument (-C/path/to/config or –config=/path/to/config)

The configuration file is INI based and requires the *softlayer* section to be present. The only required fields are *username* and *api_key*. You can optionally also/exclusively supply the *endpoint_url* as well.

Full config

```
[softlayer]
username = username
api_key = oyVmeipYQCNrjVS4rF9bHWV7D75S6pa1fghFl384v7mwRCbHTfuJ8qRORIqoVnha
endpoint_url = https://api.softlayer.com/xmlrpc/v3/
```

exclusive url

```
[softlayer]
endpoint_url = https://api.softlayer.com/xmlrpc/v3/
```

3.1.3 Usage Examples

To discover the available commands, simply type *sl*.

```
$ sl
usage: sl <command> [<args>...]
       sl help <command>
       sl [-h | --help]

SoftLayer Command-line Client

The available commands are:
  firewall  Firewall rule and security management
  image     Manages compute and flex images
  ssl       Manages SSL
  cci       Manage, delete, order compute instances
  dns       Manage DNS
  config    View and edit configuration for this tool
  metadata  Get details about this machine. Also available with 'my' and 'meta'
  nas       View NAS details
  iscsi     View iSCSI details

See 'sl help <command>' for more information on a specific command.

To use most commands your SoftLayer username and api_key need to be configured.
The easiest way to do that is to use: 'sl config setup'
```

As you can see, there are a number of commands. To look at the list of subcommands for Cloud Compute Instances, type *sl <command>*. For example:

```
$ sl cci
usage: sl cci [<command>] [<args>...] [options]

Manage, delete, order compute instances
```

```
The available commands are:
  network         Manage network settings
  create          Order and create a CCI
                    (see 'sl cci create-options' for choices)
  manage          Manage active CCI
  list            List CCI's on the account
  detail          Output details about a CCI
  dns             DNS related actions to a CCI
  cancel          Cancel a running CCI
  create-options  Output available available options when creating a CCI
  reload          Reload the OS on a CCI based on its current configuration

Standard Options:
  -h --help  Show this screen
```

Finally, we can make an actual call. Let's list out the CCIs on our account using *sl cci list*.

```
$ sl cci list
:.........:...........:.................:......:.......:.............:............:.....
:    id   : datacenter :       host        : cores : memory :   primary_ip   :  backend_ip  : active
:.........:...........:.................:......:.......:.............:............:.....
: 1234567 :   dal05    :  test.example.com  :   4   :   4G   :    12.34.56    :   65.43.21   :
:.........:...........:.................:......:.......:.............:............:.....
```

Most commands will take in additional options/arguments. To see all available actions, use *–help*.

```
$ sl cci list --help
usage: sl cci list [--hourly | --monthly] [--sortby=SORT_COLUMN] [--tags=TAGS]
                   [options]

List CCIs

Examples:
    sl cci list --datacenter=dal05
    sl cci list --network=100 --cpu=2
    sl cci list --memory='>= 2048'
    sl cci list --tags=production,db

Options:
  --sortby=ARG  Column to sort by. options: id, datacenter, host,
                Cores, memory, primary_ip, backend_ip

Filters:
  --hourly                Show hourly instances
  --monthly               Show monthly instances
  -H --hostname=HOST      Host portion of the FQDN. example: server
  -D --domain=DOMAIN      Domain portion of the FQDN example: example.com
  -c --cpu=CPU            Number of CPU cores
  -m --memory=MEMORY      Memory in mebibytes (n * 1024)
  -d DC, --datacenter=DC  datacenter shortname (sng01, dal05, ...)
  -n MBPS, --network=MBPS Network port speed in Mbps
  --tags=ARG              Only show instances that have one of these tags.
                          Comma-separated. (production,db)

For more on filters see 'sl help filters'

Standard Options:
  --format=ARG          Output format. [Options: table, raw] [Default: table]
```

```
  -C FILE --config=FILE  Config file location. [Default: ~/.softlayer]
  -h --help              Show this screen
```

3.2 Working with Cloud Compute Instances

Using the SoftLayer portal for ordering Cloud Compute Instances is fine but for a number of reasons it's sometimes to use the command-line. For this, you can use the SoftLayer command-line client to make administrative tasks quicker and easier. This page gives an intro to working with SoftLayer Cloud Compute Instances using the SoftLayer command-line client.

Note: The following assumes that the client is already *configured with valid SoftLayer credentials*.

First, let's list the current Cloud Compute Instances with *sl cci list*.

```
$ sl cci list
:....:...........:.....:......:........:...........:...........:.................:
: id : datacenter : host : cores : memory : primary_ip : backend_ip : active_transaction :
:....:...........:.....:......:........:...........:...........:.................:
:....:...........:.....:......:........:...........:...........:.................:
```

We don't have any Cloud Compute Instances! Let's fix that. Before we can create a CCI, we need to know what options are available to me: RAM, CPU, operating systems, disk sizes, disk types, datacenters. Luckily, there's a simple command to do that, *sl cci create-options*.

```
$ sl cci create-options
:................:.....................................................................
:            Name : Value
:................:.....................................................................
:      datacenter : ams01,dal01,dal05,sea01,sjc01,sng01,wdc01
:  cpus (private) : 1,2,4,8
: cpus (standard) : 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16
:          memory : 1024,2048,3072,4096,5120,6144,7168,8192,9216,10240,11264,12288,13312,14336,15360,
:     os (CENTOS) : CENTOS_5_32
:                 : CENTOS_5_64
:                 : CENTOS_6_32
:                 : CENTOS_6_64
: os (CLOUDLINUX) : CLOUDLINUX_5_32
:                 : CLOUDLINUX_5_64
:                 : CLOUDLINUX_6_32
:                 : CLOUDLINUX_6_64
:     os (DEBIAN) : DEBIAN_5_32
:                 : DEBIAN_5_64
:                 : DEBIAN_6_32
:                 : DEBIAN_6_64
:                 : DEBIAN_7_32
:                 : DEBIAN_7_64
:     os (REDHAT) : REDHAT_5_64
:                 : REDHAT_6_32
:                 : REDHAT_6_64
:     os (UBUNTU) : UBUNTU_10_32
:                 : UBUNTU_10_64
:                 : UBUNTU_12_32
:                 : UBUNTU_12_64
:                 : UBUNTU_8_32
:                 : UBUNTU_8_64
```

```
:   os (VYATTACE) : VYATTACE_6.5_64
:        os (WIN) : WIN_2003-DC-SP2-1_32
:                 : WIN_2003-DC-SP2-1_64
:                 : WIN_2003-ENT-SP2-5_32
:                 : WIN_2003-ENT-SP2-5_64
:                 : WIN_2003-STD-SP2-5_32
:                 : WIN_2003-STD-SP2-5_64
:                 : WIN_2008-DC-R2_64
:                 : WIN_2008-DC-SP2_32
:                 : WIN_2008-DC-SP2_64
:                 : WIN_2008-ENT-R2_64
:                 : WIN_2008-ENT-SP2_32
:                 : WIN_2008-ENT-SP2_64
:                 : WIN_2008-STD-R2-SP1_64
:                 : WIN_2008-STD-R2_64
:                 : WIN_2008-STD-SP2_32
:                 : WIN_2008-STD-SP2_64
:                 : WIN_2012-DC_64
:                 : WIN_2012-STD_64
:                 : WIN_7-ENT_32
:                 : WIN_7-PRO_32
:                 : WIN_8-ENT_64
:   local disk(0) : 25,100
:   local disk(2) : 25,100,150,200,300
:     san disk(0) : 25,100
:     san disk(2) : 10,20,25,30,40,50,75,100,125,150,175,200,250,300,350,400,500,750,1000,1500,2000
:     san disk(3) : 10,20,25,30,40,50,75,100,125,150,175,200,250,300,350,400,500,750,1000,1500,2000
:     san disk(4) : 10,20,25,30,40,50,75,100,125,150,175,200,250,300,350,400,500,750,1000,1500,2000
:     san disk(5) : 10,20,25,30,40,50,75,100,125,150,175,200,250,300,350,400,500,750,1000,1500,2000
:             nic : 10,100,1000
:.................:.....................................................................................
```

Here's the command to create a 2-core, 1G memory, Ubuntu 12.04 hourly instance in the San Jose datacenter using the command *sl cci create*.

```
$ sl cci create --host=example --domain=softlayer.com -c 2 -m 1024 -o UBUNTU_12_64 --hourly --datacer
This action will incur charges on your account. Continue? [y/N]: y
:.........:.....................................:
:    name : value                               :
:.........:.....................................:
:      id : 1234567                             :
: created : 2013-06-13T08:29:44-06:00           :
:    guid : 6e013cde-a863-46ee-8s9a-f806dba97c89 :
:.........:.....................................:
```

With the last command, the Cloud Compute Instance has begun being created. It should instantly appear in your listing now.

```
$ sl cci list
:.........:...........:......................:......:........:................:..............:....
:    id   : datacenter :          host         : cores : memory :   primary_ip   :  backend_ip  : act
:.........:...........:......................:......:........:................:..............:....
: 1234567 :   sjc01    : example.softlayer.com :   2   :   1G   : 108.168.200.11 : 10.54.80.200 :
:.........:...........:......................:......:........:................:..............:....
```

Cool. You may ask "It's creating... but how do I know when it's done?". Well, here's how:

```
$ sl cci ready 'example' --wait=600
READY
```

When the previous command returns, I know that the Cloud Compute Instance has finished the provisioning process and is ready to use. This is *very* useful for chaining commands together. Now that you have your Cloud Compute Instance, let's get access to it. To do that, use the *sl cci detail* command. From the example below, you can see that the username is 'root' and password is 'ABCDEFGH'.

Warning: Be careful when using the *–passwords* flag. This will print the password to the Cloud Compute Instance onto the screen. Make sure no one is looking over your shoulder. It's also advisable to change your root password soon after creating your Cloud Compute Instance.

```
$ sl cci detail example --passwords
:..............:............................:
:         Name : Value                      :
:..............:............................:
:           id : 1234567                    :
:     hostname : example.softlayer.com      :
:       status : Active                     :
:        state : Running                    :
:   datacenter : sjc01                      :
:        cores : 2                          :
:       memory : 1G                         :
:    public_ip : 108.168.200.11             :
:   private_ip : 10.54.80.200               :
:           os : Ubuntu                     :
: private_only : False                      :
:  private_cpu : False                      :
:      created : 2013-06-13T08:29:44-06:00  :
:     modified : 2013-06-13T08:31:57-06:00  :
:        users : root ABCDEFGH              :
:..............:............................:
```

There are many other commands to help manage Cloud Compute Instances. To see them all, use *sl help cci*.

```
$ sl help cci
usage: sl cci [<command>] [<args>...] [options]

Manage, delete, order compute instances

The available commands are:
  network         Manage network settings
  create          Order and create a CCI
                    (see 'sl cci create-options' for choices)
  manage          Manage active CCI
  list            List CCI's on the account
  detail          Output details about a CCI
  dns             DNS related actions to a CCI
  cancel          Cancel a running CCI
  create-options  Output available available options when creating a CCI
  reload          Reload the OS on a CCI based on its current configuration
  ready           Check if a CCI has finished provisioning

For several commands, <identifier> will be asked for. This can be the id,
hostname or the ip address for a CCI.

Standard Options:
  -h --help  Show this screen
```

Note: Full example module available *here*

3.3 Command-Line Interface Developer Guide

A CLI interface is broken into 4 major parts:

- module
- docblock
- action
- docblock
- docblock

3.3.1 Defining a module

A module is a python module residing in *SoftLayer/CLI/modules/<module>.py*. The filename represented here is what is directly exposed after the *sl* command. I.e. *sl cci* is *SoftLayer/CLI/modules/cci.py*. The module's docblock is used as the argument parser and usage the end user will see. *SoftLayer.CLI.helpers* contain all the helper functions and classes used for creating a CLI interface. This is a typical setup and how it maps:

```
"""
usage: sl example [<command>] [<args>...] [options]

Example implementation of a CLI module

Available commands are:
  print    print example
  pretty   formatted print example
  parse  parsing args example
"""

from SoftLayer.CLI import (
    CLIRunnable, Table, no_going_back, confirm)

$ sl example
usage: sl example [<command>] [<args>...] [options]

Example implementation of a CLI module

Available commands are:
  print    print example
  pretty   formatted print example
  parse  parsing args example

Standard Options:
  -h --help  Show this screen
```

3.3.2 Action

Actions are implemented using classes in the module that subclass *CLIRunnable*. The actual class name is irrelevant for the implementation details as it isn't used anywhere. The docblock is used as the arguement parser as well. Unlike the modules docblock, additional, common, arguments are added to the end as well; i.e. *–config* and *–format*.

```
class CLIRunnable(object):
    action = None

    @staticmethod
    def add_additional_args(parser):
        pass

    @staticmethod
    def execute(client, args):
        pass
```

The required interfaces are:

- The docblock (__doc__) for docopt
- action
- def execute(client, args)
  - Don't forget the @staticmethod annotation!
  - you can also use @classmethod and use execute(cls, client, args) if you plan on dispatching instead of executing a simple task.

A minimal implementation for *sl example print* would look like this:

```
class ExampleAction(CLIRunnable):
    """
usage: sl example print [options]

Print example
"""

    action = 'print'

    @staticmethod
    def execute(client, args):
        print "EXAMPLE!"
```

Which in turn, works like this:

```
$ sl example print
EXAMPLE!
$ sl example print -h
usage: sl example print [options]

Print example

Standard Options:
  --format=ARG           Output format. [Options: table, raw] [Default: table]
  -C FILE --config=FILE  Config file location. [Default: ~/.softlayer]
  -h --help              Show this screen
```

3.3.3 Output

The *execute()* method is expected to return either *None* or an instance of *SoftLayer.CLI.helpers.Table*. When *None* is returned, it assumes all output is handled inside of *execute*. *SoftLayer.CLI.modules.dns.DumpZone* is a great example of when handling your own output is ideal as the data is already coming back preformatted from the API. 99% of the time though, data will be raw and unformatted. As an example, we create *sl example pretty* as such:

```
class ExamplePretty(CLIRunnable):
    """
usage: sl example pretty [options]

Pretty output example
"""

    action = 'pretty'

    @staticmethod
    def execute(client, args):
        # create a table with two columns: col1, col2
        t = Table(['col1', 'col2'])

        # align the data facing each other
        # valid values are r, c, l for right, center, left
        # note, these are suggestions based on the format chosen by the user
        t.align['col1'] = 'r'
        t.align['col2'] = 'l'

        # add rows
        t.add_row(['test', 'test'])
        t.add_row(['test2', 'test2'])

        return t
```

Which gives us

```
$ sl example pretty
:.......:.......:
:  col1 : col2  :
:.......:.......:
:  test : test  :
: test2 : test2 :
:.......:.......:

$ sl example pretty --format raw
 test   test
 test2  test2
```

Formatting of the data represented in the table is actually controlled upstream from the CLIRunnable's making supporting more data formats in the future easier.

3.3.4 Adding arguments

Refer to docopt for more complete documentation

```
class ExampleArgs(CLIRunnable):
    """
usage: sl example parse [--test] [--this=THIS|--that=THAT]
                        (--one|--two) [options]

Argument parsing example

Options:
  --test  Print different output
"""
```

```
    action = 'parse'

    @staticmethod
    def execute(client, args):
        if args.get('--test'):
            print "Just testing, move along..."
        else:
            print "This is fo'realz!"

        if args['--one']:
            print 1
        elif args['--two']:
            print 2

        if args.get('--this'):
            print "I gots", args['--this']

        if args.get('--that'):
            print "you dont have", args['--that']
```

3.3.5 Accessing the API

API access is available via the first argument of *execute* which will be an initialized copy of *SoftLayer.API.Client*. Please refer to [using the api](API-Usage) for further details on howto use the *Client* object.

3.3.6 Confirmations

All confirmations should be easily bypassed by checking for *args['–really']*. To inject *–really* add *options = ['confirm']* to the class definition, typically just below *action*. This ensures that *–really* is consistent throughout the CLI.

```
class ExampleArgs(CLIRunnable):
    """
usage: sl example parse [--test] [--this=THIS|--that=THAT]
                        (--one|--two) [options]

Argument parsing example

Options:
  --test  Print different output
"""

    action = 'parse'
    options = ['confirm']  # confirm adds the '-y|--really' options and help

    @staticmethod
    def execute(client, args):
        pass
```

There are two primary confirmation prompts that both leverage *SoftLayer.CLI.valid_response*:

- *SoftLayer.CLI.helpers.confirm*
- *SoftLayer.CLI.helpers.no_going_back*

no_going_back accepts a single confirmation parameter that is generally unique to that action. This is similar to typing in the hostname of a machine you are canceling or some other string that isn't reactionary such as "yes", "just do it".

Some good examples would be the ID of the object, a phrase "I know what I am doing" or anything of the like. It returns True, False, or None. The prompt string is pre-defined.

confirm is a lot more flexible in that you can set the prompt string, allowing default values, and such. But it's limited to 'yes' or 'no' values. Returns True, False, or None.

```
confirmation = args.get('--really') or no_going_back('YES')

if confirmation:
    pass
```

3.3.7 Aborting execution

When a confirmation fails, you will need to bail out of *execute()*. Raise a *SoftLayer.CLI.helpers.CLIAbort* with the message for the user as the first parameter. This will prevent any further execution and properly return the right error code.

```
if not confirmation:
   raise CLIAbort("Aborting. Failed confirmation")
```

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

S