SoftLayer API Python Client Documentation

Release 3.3.0

SoftLayer Technologies, Inc.

October 27, 2014

Contents

API Docs | Github | Issues | PyPI | Twitter | #softlayer on freenode

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
```

1.2 Debian/Ubuntu

For Debian "jessie" (currently testing) and Ubuntu 14.04, official system packages are available:

```
$ sudo apt-get install python-softlayer
```

1.3 From Source

The project is developed on GitHub, at https://github.com/softlayer/softlayer-python.

Install from source via pip (requires git):

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

You can clone the public repository:

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

Or, Download the tarball:

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

Or, download the zipball:

```
$ curl -OL https://github.com/softlayer/softlayer-python/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
```

For more information about working with the source, or contributing to the project, please see the *Contribution Guide*.

CHAPTER 2

Configuration File

The SoftLayer API bindings load your settings from a number of different locations.

- Input directly into SoftLayer.Client(...)
- Enviorment variables (SL_USERNAME, SL_API_KEY)
- Config file locations (~/.softlayer, /etc/softlayer.conf)
- 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 supply the *endpoint_url* as well. This file is created automatically by the *sl config setup* command detailed here: *Configuration Setup*.

Config Example

```
[softlayer]
username = username
api_key = oyVmeipYQCNrjVS4rF9bHWV7D75S6pa1fghFl384v7mwRCbHTfuJ8qRORIqoVnha
endpoint_url = https://api.softlayer.com/xmlrpc/v3/
timeout = 40
```

CHAPTER 3

API Documentation

This is the primary API client to make API calls. It deals with constructing and executing XML-RPC calls against the SoftLayer API. Below are some links that will help to use the SoftLayer API.

- SoftLayer API Documentation
- Source on Github

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

3.1 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) and a timeout of 4 minutes.

```
client = SoftLayer.Client(
        username='YOUR_USERNAME',
        api_key='YOUR_API_KEY'
        endpoint_url=SoftLayer.API_PRIVATE_ENDPOINT,
        timeout=240,
    )
```

3.2 Managers

For day to day operation, most users will find the managers to be the most convenient means for interacting with the API. 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.

```
>>> from SoftLayer import VSManager, Client
>>> client = Client(...)
>>> vs = VSManager(client)
>>> vs.list_instances()
[...]
```

Available managers:

Warning: This module is now deprecated. It has been replaced with the VSIManager.

3.2.1 SoftLayer.cci

CCIManager to provide backwards compatibility.

license MIT, see LICENSE for more details.

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

Wrapper for the VSManager class to provide backwards compatibility with the old CCIManager class.

cancel_instance(*instance_id*)

Cancel an instance immediately, deleting all its data.

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

```
# Cancel for instance ID 12345.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
mgr.cancel_instance(12345)
```

capture(*instance_id*, *name*, *additional_disks=False*, *notes=None*)

Capture one or all disks from a VS to a SoftLayer image.

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

Parameters

- **instance_id** (*integer*) – the instance ID to edit
- **name** (*string*) – name assigned to the image
- **additional_disks** (*string*) – set to true to include all additional attached storage devices
- **notes** (*string*) – notes about this particular image

change_port_speed(*instance_id*, *public*, *speed*)

Allows you to change the port speed of a virtual server's NICs.

Parameters

- **instance_id** (*int*) – The ID of the VS

- **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.

create_instance(***kwargs*)
Creates a new virtual server instance

Parameters

- **cpus** (*int*) – The number of virtual CPUs to include in the instance.
- **memory** (*int*) – The amount of RAM to order.
- **hourly** (*bool*) – Flag to indicate if this server should be billed hourly (default) or monthly.
- **hostname** (*string*) – The hostname to use for the new server.
- **domain** (*string*) – The domain to use for the new server.
- **local_disk** (*bool*) – Flag to indicate if this should be a local disk (default) or a SAN disk.
- **datacenter** (*string*) – The short name of the data center in which the VS should reside.
- **os_code** (*string*) – The operating system to use. Cannot be specified if image_id is specified.
- **image_id** (*int*) – The ID of the image to load onto the server. Cannot be specified if os_code is specified.
- **dedicated** (*bool*) – Flag to indicate if this should be housed on a dedicated or shared host (default). This will incur a fee on your account.
- **public_vlan** (*int*) – The ID of the public VLAN on which you want this VS placed.
- **private_vlan** (*int*) – The ID of the public VLAN on which you want this VS placed.
- **disks** (*list*) – A list of disk capacities for this server.
- **post_uri** (*string*) – The URI of the post-install script to run after reload
- **private** (*bool*) – If true, the VS will be provisioned only with access to the private network. Defaults to false
- **ssh_keys** (*list*) – The SSH keys to add to the root user
- **nic_speed** (*int*) – The port speed to set
- **tag** (*string*) – tags to set on the VS as a comma separated list

create_instances(*config_list*)
Creates multiple virtual server instances

This takes a list of dictionaries using the same arguments as create_instance().

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

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

Parameters

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

- **notes** (*string*) – notes about this particular VS
- **tag** (*string*) – tags to set on the VS as a comma separated list. Use the empty string to remove all tags.

get_create_options()
Retrieves the available options for creating a VS.

Returns A dictionary of creation options.

get_instance(*instance_id*, ***kwargs*)
Get details about a virtual server instance

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

Returns A dictionary containing a large amount of information about the specified instance.

```
# Print out the FQDN and IP address for instance ID 12345.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
vsi = mgr.get_instance(12345)
print vsi['fullyQualifiedDomainName'], vs['primaryIpAddress']
```

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 virtual servers 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 options (mask, limit, etc.)

Returns Returns a list of dictionaries representing the matching virtual servers

```
# Print out a list of all hourly instances in the DAL05 data center.
# env variables
# SL_USERNAME = YOUR_USERNAME
```

```
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
for vsi in mgr.list_instances(hourly=True, datacenter='dal05'):
    print vsi['fullyQualifiedDomainName'], vs['primaryIpAddress']
```

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

Parameters

- **instance_id** (*integer*) – the instance ID to reload
- **post_url** (*string*) – The URI of the post-install script to run after reload
- **ssh_keys** (*list*) – The SSH keys to add to the root user

```
# Reload instance ID 12345 then run a custom post-provision script.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

post_uri = 'https://somehost.com/bootstrap.sh'
mgr = SoftLayer.VSManager(client)
vsi = mgr.reload_instance(12345, post_uri=post_url)
```

rescue(*instance_id*)
Reboot a VSI into the Xen recsue kernel

Parameters **instance_id** (*integer*) – the instance ID to rescue

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

upgrade(*instance_id*, *cpus=None*, *memory=None*, *nic_speed=None*, *public=True*)
Upgrades a VS instance

Parameters

- **instance_id** (*int*) – Instance id of the VS to be upgraded
- **cpus** (*int*) – The number of virtual CPUs to upgrade to of a VS instance.
- **public** (*bool*) – CPU will be in Private/Public Node.
- **memory** (*int*) – RAM of the VS to be upgraded to.
- **nic_speed** (*int*) – The port speed to set

```
# Upgrade instance 12345 to 4 CPUs and 4 GB of memory
import SoftLayer
client = SoftLayer.Client(config="~/.softlayer")
```

```
mgr = SoftLayer.VSManager(client)
mgr.upgrade(12345, cpus=4, memory=4)
```

verify_create_instance(***kwargs*)
Verifies an instance creation command without actually placing an order. See `create_instance()` for a list of available options.

wait_for_ready(*instance_id*, *limit*, *delay=1*, *pending=False*)
Determine if a VS is ready and available. In some cases though, that can mean that no transactions are running. The default arguments imply a VS is operational and ready for use by having network connectivity and remote access is available. Setting `pending=True` will ensure future API calls against this instance will not error due to pending transactions such as OS Reloads and cancellations.

Parameters

- **instance_id** (*int*) – The instance ID with the pending transaction
- **limit** (*int*) – The maximum amount of time to wait.
- **delay** (*int*) – The number of seconds to sleep before checks. Defaults to 1.
- **pending** (*bool*) – Wait for pending transactions not related to provisioning or reloads such as monitoring.

wait_for_transaction(*instance_id*, *limit*, *delay=1*)
Waits on a VS transaction for the specified amount of time. is really just a wrapper for wait_for_ready(pending=True). Provided for backwards compatibility.

Parameters

- **instance_id** (*int*) – The instance ID with the pending transaction
- **limit** (*int*) – The maximum amount of time to wait.
- **delay** (*int*) – The number of seconds to sleep before checks. Defaults to 1.

3.2.2 SoftLayer.cdn

CDN Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.cdn.**CDNManager**(*client*)
Manage CDN accounts

add_origin(*account_id*, *media_type*, *origin_url*, *cname=None*, *secure=False*)
Adds an original pull mapping to an origin-pull based CDN account with the options provided.

Parameters

- **account_id** (*int*) – the numeric ID associated with the CDN account.
- **media_type** (*string*) – the media type/protocol associated with this origin pull mapping; valid values are HTTP, FLASH, and WM.
- **origin_url** (*string*) – the base URL from which content should be pulled.
- **cname** (*string*) – an optional CNAME that should be associated with this origin pull rule; only the hostname should be included (i.e., no 'http://', directories, etc.).
- **secure** (*boolean*) – specifies whether this is an SSL origin pull rule, if SSL is enabled on your account (defaults to false).

get_account (*account_id*, ***kwargs*)
Retrieves a CDN account with the specified account ID.

Parameters

- **int** (*account_id*) – the numeric ID associated with the CDN account.
- ****kwargs** (*dict*) – additional arguments to include in the object mask.

get_origins (*account_id*, ***kwargs*)
Retrieves a list of origin pull mappings for a specified CDN account.

Parameters

- **int** (*account_id*) – the numeric ID associated with the CDN account.
- ****kwargs** (*dict*) – additional arguments to include in the object mask.

list_accounts ()
Lists CDN accounts for the active user.

load_content (*account_id*, *urls*)
Prefetches one or more URLs to the CDN edge nodes.

Parameters

- **account_id** (*int*) – the CDN account ID into which content should be preloaded.
- **urls** – a string or a list of strings representing the CDN URLs that should be pre-loaded.

Returns true if all load requests were successfully submitted; otherwise, returns the first error encountered.

purge_content (*account_id*, *urls*)
Purges one or more URLs from the CDN edge nodes.

Parameters

- **account_id** (*int*) – the CDN account ID from which content should be purged.
- **urls** – a string or a list of strings representing the CDN URLs that should be purged.

Returns true if all purge requests were successfully submitted; otherwise, returns the first error encountered.

remove_origin (*account_id*, *origin_id*)
Removes an origin pull mapping with the given origin pull ID under the provided CDN account ID.

Parameters

- **account_id** (*int*) – the CDN account ID from which the mapping should be deleted.
- **origin_id** (*int*) – the origin pull mapping ID to delete.

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

3.2.3 SoftLayer.dns

DNS Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.dns.**DNSManager**(*client*)
DNSManager initialization.

Parameters **client** (*SoftLayer.API.Client*) – the client instance

create_record(*zone_id*, *record*, *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
- **record_type** – the type of record (A, AAAA, CNAME, MX, 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(*zone_id*)
Delete a zone by its ID.

Parameters **zone_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*, *record_type=None*)
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
- **record_type** – optionally, the type of record:

Returns A list of dictionaries representing the matching records within the specified zone.

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

Parameters **zone** – the zone name

Returns A dictionary containing a large amount of information about the specified zone.

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

Parameters ****kwargs** (*dict*) – response-level options (mask, limit, etc.)

Returns A list of dictionaries representing the matching zones.

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

3.2.4 SoftLayer.firewall

Firewall Manager/helpers

license MIT, see LICENSE for more details.

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

Parameters **client** (*SoftLayer.API.Client*) – the API client instance

add_standard_firewall(*server_id*, *is_cci=True*)
Creates a firewall for the specified CCI/Server

Parameters

- **cci_id** (*int*) – The ID of the CCI to create the firewall for
- **is_cci** (*bool*) – If false, will create the firewall for a server, otherwise for a CCI

Returns A dictionary containing the standard CCI firewall order

add_vlan_firewall(*vlan_id*, *ha_enabled=False*)
Creates a firewall for the specified vlan

Parameters

- **vlan_id** (*int*) – The ID of the vlan to create the firewall for
- **ha_enabled** (*bool*) – If True, Ha firewall will be created

Returns A dictionary containing the VLAN firewall order

cancel_firewall (*firewall_id*, *dedicated=False*)
Cancels the specified firewall.

Parameters

- **firewall_id** (*int*) – Firewall ID to be cancelled.
- **dedicated** (*bool*) – If true, the firewall instance is dedicated, otherwise, the firewall instance is shared.

edit_dedicated_fwl_rules (*firewall_id*, *rules*)
Edit the rules for dedicated firewall

Parameters

- **firewall_id** (*integer*) – the instance ID of the dedicated firewall
- **rules** (*dict*) – the rules to be pushed on the firewall

edit_standard_fwl_rules (*firewall_id*, *rules*)
Edit the rules for standard firewall

Parameters

- **firewall_id** (*integer*) – the instance ID of the standard firewall
- **rules** (*dict*) – the rules to be pushed on the firewall

get_dedicated_fwl_rules (*firewall_id*)
Get the rules of a dedicated firewall

Parameters **firewall_id** (*integer*) – the instance ID of the dedicated firewall

Returns A list of the rules.

get_dedicated_package (*ha_enabled=False*)
Retrieves the dedicated firewall package.

Parameters **ha_enabled** (*bool*) – True if HA is to be enabled on the firewall False for No HA

Returns A dictionary containing the dedicated CCI firewall package

get_firewalls ()
Returns a list of all firewalls on the account.

Returns A list of firewalls on the current account.

get_standard_fwl_rules (*firewall_id*)
Get the rules of a standard firewall

Parameters **firewall_id** (*integer*) – the instance ID of the standard firewall

Returns A list of the rules.

get_standard_package (*server_id*, *is_cci=True*)
Retrieves the standard firewall package for the CCI.

Parameters

- **server_id** (*int*) – The ID of the server to create the firewall for
- **is_cci** (*bool*) – True if the id provided is for a CCI, False for a server

Returns A dictionary containing the standard CCI firewall package

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

`SoftLayer.managers.firewall.`**`has_firewall`**(*vlan*)
Helper to determine whether or not a VLAN has a firewall.

Parameters **vlan** (*dict*) – A dictionary representing a VLAN

Returns True if the VLAN has a firewall, false if it doesn't.

3.2.5 SoftLayer.hardware

Hardware Manager/helpers

license MIT, see LICENSE for more details.

class `SoftLayer.managers.hardware.`**`HardwareManager`**(*client*, *ordering_manager=None*)
Manages hardware devices.

Parameters

- **client** (*SoftLayer.API.Client*) – an API client instance
- **ordering_manager** (*SoftLayer.managers.OrderingManager*) – an optional manager to handle ordering. If none is provided, one will be auto initialized.

`cancel_hardware`(*hardware_id*, *reason='unneeded'*, *comment=''*, *immediate=False*)
Cancels the specified dedicated server.

Parameters

- **hardware_id** (*int*) – The ID of the hardware to be cancelled.
- **reason** (*string*) – The reason code for the cancellation. This should come from `get_cancellation_reasons()`.
- **comment** (*string*) – An optional comment to include with the cancellation.

`cancel_metal`(*hardware_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`(*hardware_id*, *public*, *speed*)
Allows you to change the port speed of a server's NICs.

Parameters

- **hardware_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(*hardware_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

- **hardware_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 name
- **notes** (*string*) – notes about this particular hardware

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

Returns A list of tuples of available dedicated server packages in the form (id, name, description)

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

Returns A dictionary of creation options. The categories to order are contained within the 'categories' key. See `_parse_package_data()` for detailed information.

Note: 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_bare_metal_package_id()
Return the bare metal package id

get_cancellation_reasons()
Returns a dictionary of valid cancellation reasons that can be used when cancelling a dedicated server via `cancel_hardware()`.

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

Parameters **package_id** (*int*) – The package ID to retrieve the creation options for. This should come from `get_available_dedicated_server_packages()`.

Returns A dictionary of creation options. The categories to order are contained within the 'categories' key. See `_parse_package_data()` for detailed information.

Note: The information for ordering dedicated servers comes from multiple API calls. In order to make the process simpler, 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(*hardware_id*, ***kwargs*)
Get details about a hardware device

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

Returns A dictionary containing a large amount of information about the specified server.

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 (servers and bare metal computing instances).

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 options (mask, limit, etc.)

Returns Returns a list of dictionaries representing the matching hardware. This list will contain both dedicated servers and bare metal computing instances

place_order(***kwargs*)

Places an order for a piece of hardware.

Translates a list of arguments into a dictionary necessary for creating a server.

> **Warning:** All items here must be price IDs, NOT quantities!

Parameters

- **server** (*int*) – The identification string for the server to order. This will either be the CPU/Memory combination ID for bare metal instances or the CPU model for dedicated servers.
- **hostname** (*string*) – The hostname to use for the new server.
- **domain** (*string*) – The domain to use for the new server.
- **hourly** (*bool*) – Flag to indicate if this server should be billed hourly (default) or monthly. Only applies to bare metal instances.
- **location** (*string*) – The location string (data center) for the server
- **os** (*int*) – The operating system to use
- **disks** (*array*) – An array of disks for the server. Disks will be added in the order specified.
- **port_speed** (*int*) – The port speed for the server.
- **bare_metal** (*bool*) – Flag to indicate if this is a bare metal server or a dedicated server (default).
- **ram** (*int*) – The amount of RAM to order. Only applies to dedicated servers.
- **package_id** (*int*) – The package_id to use for the server. This should either be a chassis ID for dedicated servers or the bare metal instance package ID, which can be obtained by calling get_bare_metal_package_id
- **disk_controller** (*int*) – The disk controller to use.

- **ssh_keys** (*list*) – The SSH keys to add to the root user
- **public_vlan** (*int*) – The ID of the public VLAN on which you want this server placed.
- **private_vlan** (*int*) – The ID of the public VLAN on which you want this server placed.
- **post_uri** (*string*) – The URI of the post-install script to run after reload

Warning: Due to how the ordering structure currently works, all ordering takes place using price IDs rather than quantities. See the following sample for an example of using HardwareManager functions for ordering a basic server.

```
# client is assumed to be an initialized SoftLayer.API.Client object
mgr = HardwareManager(client)

# Package ID 32 corresponds to the 'Quad Processor, Quad Core Intel'
# package. This information can be obtained from the
# :func:'get_available_dedicated_server_packages' function.
options = mgr.get_dedicated_server_create_options(32)

# Review the contents of options to find the information that
# applies to your order. For the sake of this example, we assume
# that your selections are a series of item IDs for each category
# organized into a key-value dictionary.

# This contains selections for all required categories
selections = {
    'server': 542, # Quad Processor Quad Core Intel 7310 - 1.60GHz
    'pri_ip_addresses': 15, # 1 IP Address
    'notification': 51, # Email and Ticket
    'ram': 280, # 16 GB FB-DIMM Registered 533/667
    'bandwidth': 173, # 5000 GB Bandwidth
    'lockbox': 45, # 1 GB Lockbox
    'monitoring': 49, # Host Ping
    'disk0': 14, # 500GB SATA II (for the first disk)
    'response': 52, # Automated Notification
    'port_speed': 187, # 100 Mbps Public & Private Networks
    'power_supply': 469, # Redundant Power Supplies
    'disk_controller': 487, # Non-RAID
    'vulnerability_scanner': 307, # Nessus
    'vpn_management': 309, # Unlimited SSL VPN Users
    'remote_management': 504, # Reboot / KVM over IP
    'os': 4166, # Ubuntu Linux 12.04 LTS Precise Pangolin (64 bit)
}

args = {
    'location': 'FIRST_AVAILABLE', # Pick the first available DC
    'packageId': 32, # From above
    'disks': [],
}

for cat, item_id in selections:
    for item in options['categories'][cat]['items'].items():
        if item['id'] == item_id:
            if 'disk' not in cat or 'disk_controller' == cat:
                args[cat] = item['price_id']
            else:
                args['disks'].append(item['price_id'])
```

```
    # You can call :func:'verify_order' here to test the order instead
    # of actually placing it if you prefer.
    result = mgr.place_order(**args)
```

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

Parameters

- **hardware_id** (*integer*) – the instance ID to reload
- **post_url** (*string*) – The URI of the post-install script to run after reload
- **ssh_keys** (*list*) – The SSH keys to add to the root user

rescue(*hardware_id*)
Reboot a server into the a recsue kernel

Parameters **instance_id** (*integer*) – the server ID to rescue

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

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

SoftLayer.managers.hardware.**get_default_value**(*package_options*, *category*, *hourly=False*)
Returns the default price ID for the specified category.

This determination is made by parsing the items in the package_options argument and finding the first item that has zero specified for every fee field.

Note: If the category has multiple items with no fee, this will return the first it finds and then short circuit. This may not match the default value presented on the SoftLayer ordering portal. Additionally, this method will return None if there are no free items in the category.

Returns Returns the price ID of the first free item it finds or None if there are no free items.

3.2.6 SoftLayer.image

Image Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.image.**ImageManager**(*client*)
Manages server images

Parameters **client** (*SoftLayer.API.Client*) – an API client instance

delete_image(*image_id*)
deletes the specified image.

Parameters **image** (*int*) – The ID of the image.

edit (*image_id*, *name=None*, *note=None*, *tag=None*)
Edit image related details :param int image: The ID of the image :param string name: Name of the Image. :param string note: Note of the image. :param string tag: Tags of the image to be updated to.

get_image (*image_id*, ***kwargs*)
Get details about an image

Parameters

- **image** (*int*) – The ID of the image.
- ****kwargs** (*dict*) – response-level options (mask, limit, etc.)

list_private_images (*guid=None*, *name=None*, ***kwargs*)
List all private images.

Parameters

- **guid** (*string*) – filter based on GUID
- **name** (*string*) – filter based on name
- ****kwargs** (*dict*) – response-level options (mask, limit, etc.)

list_public_images (*guid=None*, *name=None*, ***kwargs*)
List all public images.

Parameters

- **guid** (*string*) – filter based on GUID
- **name** (*string*) – filter based on name
- ****kwargs** (*dict*) – response-level options (mask, limit, 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

3.2.7 SoftLayer.iscsi

ISCSI Manager/helpers

class SoftLayer.managers.iscsi.**ISCSIManager** (*client*)
Manages iSCSI storages

cancel_iscsi (*volume_id*, *reason='unNeeded'*, *immediate=False*)
Cancels the given iSCSI volume

Parameters **volume_id** (*integer*) – the volume ID

create_iscsi (*size=None*, *location=None*)
Places an order for iSCSI volume :param integer size: size of iSCSI volume to create :param string location: datacenter to use to create volume in

create_snapshot (*volume_id*, *notes='No longer needed'*)
Orders a snapshot for given volume

Parameters **volume_id** (*integer*) – the volume ID

create_snapshot_space(*volume_id*, *capacity*)
Orders a snapshot space for given volume

Parameters

- **volume_id** (*integer*) – the volume ID
- **capacity** (*integer*) – capacity in ~GB

delete_snapshot(*snapshot_id*)
Deletes the given snapshot

Params integer snapshot_id: the snapshot ID

get_iscsi(*volume_id*, ***kwargs*)
Get details about a iSCSI storage

Parameters **volume_id** (*integer*) – the volume ID

Returns A dictionary containing a large amount of information about the specified storage.

list_iscsi()
List iSCSI volume

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

restore_from_snapshot(*volume_id*, *snapshot_id*)
Restore the volume to snapshot's contents :params: imteger volume_id: the volume ID :params: integer snapshot_id: the snapshot ID

3.2.8 SoftLayer.load_balancer

Load Balancer Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.load_balancer.**LoadBalancerManager**(*client*)
Manages load balancers.

Parameters **client** (*SoftLayer.API.Client*) – the API client instance

add_local_lb(*price_item_id*, *datacenter*)
Creates a local load balancer in the specified data center

Parameters

- **price_item_id** (*int*) – The price item ID for the load balancer
- **datacenter** (*string*) – The datacenter to create the loadbalancer in

Returns A dictionary containing the product order

add_service(*loadbal_id*, *service_group_id*, *ip_address_id*, *port=80*, *enabled=True*, *hc_type=21*, *weight=1*)
Adds a new service to the service group

Parameters

- **loadbal_id** (*int*) – The id of the loadbal where the service resides

- **service_group_id** (*int*) – The group to add the service to
- **ip_address id** (*int*) – The ip address ID of the service
- **port** (*int*) – the port of the service
- **enabled** (*bool*) – Enable or disable the service
- **hc_type** (*int*) – The health check type
- **weight** (*int*) – the weight to give to the service

add_service_group(*lb_id*, *allocation=100*, *port=80*, *routing_type=2*, *routing_method=10*)
Adds a new service group to the load balancer

Parameters

- **loadbal_id** (*int*) – The id of the loadbal where the service resides
- **allocation** (*int*) – percent of connections to allocate toward the group
- **port** (*int*) – the port of the service group
- **routing_type** (*int*) – the routing type to set on the service group
- **routing_method** (*int*) – The routing method to set on the group

cancel_lb(*loadbal_id*)
Cancels the specified load balancer.

Parameters **loadbal_id** (*int*) – Load Balancer ID to be cancelled.

delete_service(*service_id*)
Deletes a service from the loadbal_id

Parameters **service_id** (*int*) – The id of the service to delete

delete_service_group(*group_id*)
Deletes a service group from the loadbal_id

Parameters **group_id** (*int*) – The id of the service group to delete

edit_service(*loadbal_id*, *service_id*, *ip_address_id=None*, *port=None*, *enabled=None*, *hc_type=None*, *weight=None*)
Edits an existing service properties

Parameters

- **loadbal_id** (*int*) – The id of the loadbal where the service resides
- **service_id** (*int*) – The id of the service to edit
- **ip_address** (*string*) – The ip address of the service
- **port** (*int*) – the port of the service
- **enabled** (*bool*) – enable or disable the search
- **hc_type** (*int*) – The health check type
- **weight** (*int*) – the weight to give to the service

edit_service_group(*loadbal_id*, *group_id*, *allocation=None*, *port=None*, *routing_type=None*, *routing_method=None*)
Edit an existing service group

Parameters

- **loadbal_id** (*int*) – The id of the loadbal where the service resides

- **group_id** (*int*) – The id of the service group
- **allocation** (*int*) – the % of connections to allocate to the group
- **port** (*int*) – the port of the service group
- **routing_type** (*int*) – the routing type to set on the service group
- **routing_method** (*int*) – The routing method to set on the group

get_hc_types()
Retrieves the health check type values

Returns A dictionary containing the health check types

get_ip_address(*ip_address=None*)
Retrieves the IP address object given the ip address itself

Returns A dictionary containing the IP address properties

get_lb_pkgs()
Retrieves the local load balancer packages.

Returns A dictionary containing the load balancer packages

get_local_lb(*loadbal_id*, ***kwargs*)
Returns a specified local load balancer given the id.

Parameters **loadbal_id** (*int*) – The id of the load balancer to retrieve

Returns A dictionary containing the details of the load balancer

get_local_lbs()
Returns a list of all local load balancers on the account.

Returns A list of all local load balancers on the current account.

get_routing_methods()
Retrieves the load balancer routing methods.

Returns A dictionary containing the load balancer routing methods

get_routing_types()
Retrieves the load balancer routing types.

Returns A dictionary containing the load balancer routing types

reset_service_group(*loadbal_id*, *group_id*)
Resets all the connections on the service group

Parameters

- **loadbal_id** (*int*) – The id of the loadbal
- **group_id** (*int*) – The id of the service group to reset

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

toggle_service_status(*service_id*)
Toggles the service status

Parameters **service_id** (*int*) – The id of the service to delete

3.2.9 SoftLayer.messaging

Manager for the SoftLayer Message Queue service

license MIT, see LICENSE for more details.

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

Parameters

- **account_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*, *subscription_type*, ***kwargs*)
Create Subscription

Parameters

- **topic_name** – Topic Name
- **subscription_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*)

Pop a single message from a queue. If no messages are returned this returns None

Parameters **queue_name** – Queue Name

pop_messages(*queue_name*, *count=1*)
Pop messages 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(*account_id*, *datacenter=None*, *network=None*)
Get connection to Message Queue Service

Parameters

- **account_id** – Message Queue Account id
- **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 options (mask, limit, etc.)

ping(*datacenter=None*, *network=None*)
Ping a message queue endpoint

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(*resp*, **_)
Handle errors

3.2.10 SoftLayer.metadata

Metadata Manager/helpers

license MIT, 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:

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

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

Parameters

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

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',**

3.2.11 SoftLayer.network

Network Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.network.**NetworkManager**(*client*)
Manage Networks

add_global_ip(*version=4*, *test_order=False*)
Adds a global IP address to the account.

Parameters

- **version** (*int*) – Specifies whether this is IPv4 or IPv6
- **test_order** (*bool*) – If true, this will only verify the order.

add_subnet(*subnet_type*, *quantity=None*, *vlan_id=None*, *version=4*, *test_order=False*)
Orders a new subnet

Parameters

- **subnet_type** (*str*) – Type of subnet to add: private, public, global
- **quantity** (*int*) – Number of IPs in the subnet
- **vlan_id** (*int*) – VLAN id for the subnet to be placed into
- **version** (*int*) – 4 for IPv4, 6 for IPv6
- **test_order** (*bool*) – If true, this will only verify the order.

assign_global_ip(*global_ip_id*, *target*)
Assigns a global IP address to a specified target.

Parameters

- **global_ip_id** (*int*) – The ID of the global IP being assigned
- **target** (*string*) – The IP address to assign

cancel_global_ip(*global_ip_id*)
Cancels the specified global IP address.

Parameters **id** (*int*) – The ID of the global IP to be cancelled.

cancel_subnet(*subnet_id*)
Cancels the specified subnet.

Parameters **subnet_id** (*int*) – The ID of the subnet to be cancelled.

edit_rwhois(*abuse_email=None*, *address1=None*, *address2=None*, *city=None*, *company_name=None*, *country=None*, *first_name=None*, *last_name=None*, *postal_code=None*, *private_residence=None*, *state=None*)
Edit rwhois record

get_rwhois()
Returns the RWhois information about the current account.

Returns A dictionary containing the account's RWhois information.

get_subnet(*subnet_id*, ***kwargs*)
Returns information about a single subnet.

Parameters **id** (*string*) – Either the ID for the subnet or its network identifier

Returns A dictionary of information about the subnet

get_vlan(*vlan_id*)
Returns information about a single VLAN.

Parameters **id** (*int*) – The unique identifier for the VLAN

Returns A dictionary containing a large amount of information about the specified VLAN.

ip_lookup(*ip_address*)
Looks up an IP address and returns network information about it.

Parameters **ip_address** (*string*) – An IP address. Can be IPv4 or IPv6

Returns A dictionary of information about the IP

list_global_ips(*version=None*, *identifier=None*, ***kwargs*)
Returns a list of all global IP address records on the account.

Parameters

- **version** (*int*) – Only returns IPs of this version (4 or 6)
- **identifier** (*string*) – If specified, the list will only contain the global ips matching this network identifier.

list_subnets(*identifier=None*, *datacenter=None*, *version=0*, *subnet_type=None*, ***kwargs*)
Display a list of all subnets on the account.

This provides a quick overview of all subnets including information about data center residence and the number of devices attached.

Parameters

- **identifier** (*string*) – If specified, the list will only contain the subnet matching this network identifier.
- **datacenter** (*string*) – If specified, the list will only contain subnets in the specified data center.
- **version** (*int*) – Only returns subnets of this version (4 or 6).
- **subnet_type** (*string*) – If specified, it will only returns subnets of this type.
- ****kwargs** (*dict*) – response-level options (mask, limit, etc.)

list_vlans(*datacenter=None*, *vlan_number=None*, *name=None*, ***kwargs*)
Display a list of all VLANs on the account.

This provides a quick overview of all VLANs including information about data center residence and the number of devices attached.

Parameters

- **datacenter** (*string*) – If specified, the list will only contain VLANs in the specified data center.
- **vlan_number** (*int*) – If specified, the list will only contain the VLAN matching this VLAN number.
- **name** (*int*) – If specified, the list will only contain the VLAN matching this VLAN name.
- ****kwargs** (*dict*) – response-level options (mask, limit, etc.)

resolve_global_ip_ids(*identifier*)
Resolve global ip ids

resolve_subnet_ids(*identifier*)
Resolve subnet ids

resolve_vlan_ids(*identifier*)
Resolve VLAN ids

summary_by_datacenter()
Provides a dictionary with a summary of all network information on the account, grouped by data center.

The resultant dictionary is primarily useful for statistical purposes. It contains count information rather than raw data. If you want raw information, see the `list_vlans()` method instead.

Returns A dictionary keyed by data center with the data containing a series of counts for hardware, subnets, CCIs, and other objects residing within that data center.

unassign_global_ip(*global_ip_id*)
Unassigns a global IP address from a target.

Parameters **id** (*int*) – The ID of the global IP being unassigned

3.2.12 SoftLayer.sshkey

SSH Key Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.sshkey.**SshKeyManager**(*client*)
Manages account SSH keys.

Parameters **client** (*SoftLayer.API.Client*) – an API client instance

add_key(*key*, *label*, *notes=None*)
Adds a new SSH key to the account.

Parameters

- **key** (*string*) – The SSH key to add
- **label** (*string*) – The label for the key

Returns A dictionary of the new key's information.

delete_key(*key_id*)
Permanently deletes an SSH key from the account.

Parameters **key_id** (*int*) – The ID of the key to delete

edit_key(*key_id*, *label=None*, *notes=None*)
Edits information about an SSH key.

Parameters

- **key_id** (*int*) – The ID of the key to edit
- **label** (*string*) – The new label for the key
- **notes** (*string*) – Notes to set or change on the key

Returns A Boolean indicating success or failure

get_key(*key_id*)
Returns full information about a single SSH key.

Parameters **key_id** (*int*) – The ID of the key to retrieve

Returns A dictionary of information about the key

list_keys(*label=None*)
Lists all SSH keys on the account.

Parameters **label** (*string*) – Filter list based on SSH key label

Returns A list of dictionaries with information about each key

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

3.2.13 SoftLayer.ssl

SSL Manager/helpers

license MIT, see LICENSE for more details.

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

Parameters **client** (*SoftLayer.API.Client*) – an API client instance

add_certificate(*certificate*)
Creates a new certificate.

Parameters **certificate** (*dict*) – A dictionary representing the parts of the certificate. See SLDN for more information.

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(*cert_id*)
Gets a certificate with the ID specified.

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

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

Parameters **method** (*string*) – The type of certificates to list. Options are 'all', 'expired', and 'valid'.

Returns A list of dictionaries representing the requested SSL certs.

remove_certificate(*cert_id*)
Removes a certificate.

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

3.2.14 SoftLayer.ticket

Ticket Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.ticket.**TicketManager**(*client*)
Manages account Tickets

Parameters **client** (*SoftLayer.API.Client*) – an API client instance

create_ticket(*title=None*, *body=None*, *subject=None*)
Create a new ticket

Parameters

- **title** (*string*) – title for the new ticket
- **body** (*string*) – body for the new ticket
- **subject** (*integer*) – id of the subject to be assigned to the ticket

get_ticket(*ticket_id*)
Get details about a ticket

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

Returns A dictionary containing a large amount of information about the specified ticket.

list_subjects()
List all tickets

list_tickets(*open_status=True*, *closed_status=True*)
List all tickets

Parameters

- **open_status** (*boolean*) – include open tickets
- **closed_status** (*boolean*) – include closed tickets

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

update_ticket(*ticket_id=None*, *body=None*)
Update a ticket

Parameters

- **ticket_id** (*integer*) – the id of the ticket to update
- **body** (*string*) – entry to update in the ticket

3.2.15 SoftLayer.vs

VS Manager/helpers

license MIT, see LICENSE for more details.

class SoftLayer.managers.vs.**VSManager**(*client*, *ordering_manager=None*)
Manages Virtual Servers

Parameters

- **client** (*SoftLayer.API.Client*) – an API client instance
- **ordering_manager** (*SoftLayer.managers.OrderingManager*) – an optional manager to handle ordering. If none is provided, one will be auto initialized.

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

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

```
# Cancel for instance ID 12345.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
mgr.cancel_instance(12345)
```

capture(*instance_id*, *name*, *additional_disks=False*, *notes=None*)
Capture one or all disks from a VS to a SoftLayer image.

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

Parameters

- **instance_id** (*integer*) – the instance ID to edit
- **name** (*string*) – name assigned to the image
- **additional_disks** (*string*) – set to true to include all additional attached storage devices
- **notes** (*string*) – notes about this particular image

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

Parameters

- **instance_id** (*int*) – The ID of the VS
- **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.

create_instance(***kwargs*)
Creates a new virtual server instance

Parameters

- **cpus** (*int*) – The number of virtual CPUs to include in the instance.
- **memory** (*int*) – The amount of RAM to order.
- **hourly** (*bool*) – Flag to indicate if this server should be billed hourly (default) or monthly.
- **hostname** (*string*) – The hostname to use for the new server.
- **domain** (*string*) – The domain to use for the new server.
- **local_disk** (*bool*) – Flag to indicate if this should be a local disk (default) or a SAN disk.
- **datacenter** (*string*) – The short name of the data center in which the VS should reside.
- **os_code** (*string*) – The operating system to use. Cannot be specified if image_id is specified.
- **image_id** (*int*) – The ID of the image to load onto the server. Cannot be specified if os_code is specified.
- **dedicated** (*bool*) – Flag to indicate if this should be housed on a dedicated or shared host (default). This will incur a fee on your account.
- **public_vlan** (*int*) – The ID of the public VLAN on which you want this VS placed.
- **private_vlan** (*int*) – The ID of the public VLAN on which you want this VS placed.
- **disks** (*list*) – A list of disk capacities for this server.
- **post_uri** (*string*) – The URI of the post-install script to run after reload
- **private** (*bool*) – If true, the VS will be provisioned only with access to the private network. Defaults to false
- **ssh_keys** (*list*) – The SSH keys to add to the root user
- **nic_speed** (*int*) – The port speed to set
- **tag** (*string*) – tags to set on the VS as a comma separated list

create_instances(*config_list*)
Creates multiple virtual server instances

This takes a list of dictionaries using the same arguments as create_instance().

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

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

Parameters

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

- **tag** (*string*) – tags to set on the VS as a comma separated list. Use the empty string to remove all tags.

get_create_options()
Retrieves the available options for creating a VS.

Returns A dictionary of creation options.

get_instance(*instance_id*, ***kwargs*)
Get details about a virtual server instance

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

Returns A dictionary containing a large amount of information about the specified instance.

```
# Print out the FQDN and IP address for instance ID 12345.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
vsi = mgr.get_instance(12345)
print vsi['fullyQualifiedDomainName'], vs['primaryIpAddress']
```

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 virtual servers 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 options (mask, limit, etc.)

Returns Returns a list of dictionaries representing the matching virtual servers

```
# Print out a list of all hourly instances in the DAL05 data center.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
```

```
client = SoftLayer.Client()

mgr = SoftLayer.VSManager(client)
for vsi in mgr.list_instances(hourly=True, datacenter='dal05'):
    print vsi['fullyQualifiedDomainName'], vs['primaryIpAddress']
```

reload_instance(*instance_id*, *post_uri=None*, *ssh_keys=None*)

Perform an OS reload of an instance with its current configuration.

Parameters

- **instance_id** (*integer*) – the instance ID to reload
- **post_url** (*string*) – The URI of the post-install script to run after reload
- **ssh_keys** (*list*) – The SSH keys to add to the root user

```
# Reload instance ID 12345 then run a custom post-provision script.
# env variables
# SL_USERNAME = YOUR_USERNAME
# SL_API_KEY = YOUR_API_KEY
import SoftLayer
client = SoftLayer.Client()

post_uri = 'https://somehost.com/bootstrap.sh'
mgr = SoftLayer.VSManager(client)
vsi = mgr.reload_instance(12345, post_uri=post_url)
```

rescue(*instance_id*)

Reboot a VSI into the Xen recsue kernel

Parameters **instance_id** (*integer*) – the instance ID to rescue

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

upgrade(*instance_id*, *cpus=None*, *memory=None*, *nic_speed=None*, *public=True*)

Upgrades a VS instance

Parameters

- **instance_id** (*int*) – Instance id of the VS to be upgraded
- **cpus** (*int*) – The number of virtual CPUs to upgrade to of a VS instance.
- **public** (*bool*) – CPU will be in Private/Public Node.
- **memory** (*int*) – RAM of the VS to be upgraded to.
- **nic_speed** (*int*) – The port speed to set

```
# Upgrade instance 12345 to 4 CPUs and 4 GB of memory
import SoftLayer
client = SoftLayer.Client(config="~/.softlayer")

mgr = SoftLayer.VSManager(client)
mgr.upgrade(12345, cpus=4, memory=4)
```

verify_create_instance(***kwargs*)
Verifies an instance creation command without actually placing an order. See `create_instance()` for a list of available options.

wait_for_ready(*instance_id*, *limit*, *delay=1*, *pending=False*)
Determine if a VS is ready and available. In some cases though, that can mean that no transactions are running. The default arguments imply a VS is operational and ready for use by having network connectivity and remote access is available. Setting `pending=True` will ensure future API calls against this instance will not error due to pending transactions such as OS Reloads and cancellations.

Parameters

- **instance_id** (*int*) – The instance ID with the pending transaction
- **limit** (*int*) – The maximum amount of time to wait.
- **delay** (*int*) – The number of seconds to sleep before checks. Defaults to 1.
- **pending** (*bool*) – Wait for pending transactions not related to provisioning or reloads such as monitoring.

wait_for_transaction(*instance_id*, *limit*, *delay=1*)
Waits on a VS transaction for the specified amount of time. is really just a wrapper for wait_for_ready(pending=True). Provided for backwards compatibility.

Parameters

- **instance_id** (*int*) – The instance ID with the pending transaction
- **limit** (*int*) – The maximum amount of time to wait.
- **delay** (*int*) – The number of seconds to sleep before checks. Defaults to 1.

If you need more power or functionality than the managers provide, you can make direct API calls as well.

3.3 Making API Calls

For full control over your account and services, you can directly call the SoftLayer API. 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="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 virtual server 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'
    })
```

3.4 API Reference

class SoftLayer.**Client**(*username=None*, *api_key=None*, *endpoint_url=None*, *timeout=None*, *auth=None*, *config_file=None*, *proxy=None*, *user_agent=None*)

A SoftLayer API client.

Parameters

- **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.
- **proxy** – proxy to be used to make API calls
- **timeout** (*integer*) – timeout for API requests
- **auth** – an object which responds to get_headers() to be inserted into the xml-rpc headers. Example: *BasicAuthentication*
- **config_file** – A path to a configuration file used to load settings

- **user_agent** – an optional User Agent to report when making API calls if you wish to bypass the packages built in User Agent string

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:

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

authenticate_with_password(*username*, *password*, *security_question_id=None*, *security_question_answer=None*)
Performs Username/Password Authentication

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:

```
>>> import SoftLayer
>>> 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*)

A SoftLayer Service.

Parameters

- **client** – A SoftLayer.API.Client instance
- **str** (*name*) – The service 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
- **compress** (*boolean*) – (optional) Enable/Disable HTTP compression
- **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
- **verify** (*bool*) – verify SSL cert
- **cert** – client certificate path

Usage:

```
>>> import SoftLayer
>>> client = SoftLayer.Client()
>>> 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
- **compress** (*boolean*) – (optional) Enable/Disable HTTP compression
- **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
- **verify** (*bool*) – verify SSL cert
- **cert** – client certificate path

Usage:

```
>>> import SoftLayer
>>> client = SoftLayer.Client()
>>> 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:

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

3.4.1 SoftLayer.exceptions

Exceptions used throughout the library

license MIT, see LICENSE for more details.

exception `SoftLayer.exceptions.`**ApplicationError**(*fault_code*, *fault_string*, **args*)

Application Error.

exception SoftLayer.exceptions.**InternalError**(*fault_code*, *fault_string*, **args*)
Internal Server Error.

exception SoftLayer.exceptions.**InvalidCharacter**(*fault_code*, *fault_string*, **args*)
There was an invalid character.

exception SoftLayer.exceptions.**InvalidMethodParameters**(*fault_code*, *fault_string*, **args*)
Invalid method paramters.

exception SoftLayer.exceptions.**MethodNotFound**(*fault_code*, *fault_string*, **args*)
Method name not found.

exception SoftLayer.exceptions.**NotWellFormed**(*fault_code*, *fault_string*, **args*)
Request was not well formed.

exception SoftLayer.exceptions.**ParseError**(*fault_code*, *fault_string*, **args*)
Parse Error.

exception SoftLayer.exceptions.**RemoteSystemError**(*fault_code*, *fault_string*, **args*)
System Error.

exception SoftLayer.exceptions.**ServerError**(*fault_code*, *fault_string*, **args*)
Server Error.

exception SoftLayer.exceptions.**SoftLayerAPIError**(*fault_code*, *fault_string*, **args*)
SoftLayerAPIError is an exception raised during API errors.

Provides faultCode and faultString properties.

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

exception SoftLayer.exceptions.**SpecViolation**(*fault_code*, *fault_string*, **args*)
There was a spec violation.

exception SoftLayer.exceptions.**TransportError**(*fault_code*, *fault_string*, **args*)
Transport Error.

exception SoftLayer.exceptions.**Unauthenticated**
Unauthenticated.

exception SoftLayer.exceptions.**UnsupportedEncoding**(*fault_code*, *fault_string*, **args*)
Encoding not supported.

3.5 Backwards Compatibility

As of 3.0, the old API methods and parameters no longer work. Below are examples of converting the old API to the new one.

Get the IP address for an account

```
# Old
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()

# New
import SoftLayer
```

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

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="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'
```

CHAPTER 4

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. See the *Usage Examples* section to see how to discover all of the functionality not fully documented here.

4.1 Working with Virtual Servers

Using the SoftLayer portal for ordering virtual servers 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 virtual servers 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 virtual servers with *sl vs list*.

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

We don't have any virtual servers! Let's fix that. Before we can create a VS, 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 vs create-options*.

```
$ sl vs 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 vs create*.

```
$ sl vs create --host=example --domain=softlayer.com -c 2 -m 1024 -o UBUNTU_12_64 --hourly --datacent
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 virtual server has begun being created. It should instantly appear in your listing now.

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

When the previous command returns, I know that the virtual server has finished the provisioning process and is ready to use. This is *very* useful for chaining commands together. Now that you have your virtual server, let's get access to it. To do that, use the *sl vs 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 virtual server 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 virtual server.

```
$ sl vs 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 virtual servers. To see them all, use *sl help vs*.

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

Manage, delete, order compute instances

The available commands are:
  cancel          Cancel a running virtual server
  capture         Create an image the disk(s) of a virtual server
  create          Order and create a virtual server
                    (see sl vs create-options for choices)
```

```
  create-options  Output available available options when creating a VS
  detail          Output details about a virtual server
  dns             DNS related actions to a virtual server
  edit            Edit details of a virtual server
  list            List virtual servers on the account
  nic-edit        Edit NIC settings
  pause           Pauses an active virtual server
  power-off       Powers off a running virtual server
  power-on        Boots up a virtual server
  ready           Check if a virtual server has finished provisioning
  reboot          Reboots a running virtual server
  reload          Reload the OS on a VS based on its current configuration
  resume          Resumes a paused virtual server
  upgrade         Upgrades parameters of a virtual server

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

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

4.2 Configuration Setup

To update the configuration, you can use *sl config setup*.

```
$ sl config setup
Username []: username
API Key or Password []:
Endpoint (public|private|custom): public
:..............:..................................................................:
:         Name : Value                                                            :
:..............:..................................................................:
:     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/                            :
:..............:..................................................................:
```

To see more about the config file format, see *Configuration File*.

4.3 Usage Examples

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

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

SoftLayer Command-line Client

Compute:
  image     Manages compute and flex images
  metadata  Get details about this machine. Also available with 'my' and 'meta'
  server    Bare metal servers
  sshkey    Manage SSH keys on your account
  vs        Virtual Servers (formerly CCIs)

Networking:
  cdn        Content Delivery Network service management
  dns        Domain Name System
  firewall   Firewall rule and security management
  globalip   Global IP address management
  messaging  Message Queue Service
  rwhois     RWhoIs operations
  ssl        Manages SSL
  subnet     Subnet ordering and management
  vlan       Manage VLANs on your account

Storage:
  iscsi     View iSCSI details
  nas       View NAS details

General:
  config    View and edit configuration for this tool
  ticket    Manage account tickets
  summary   Display an overall summary of your account
  help      Show help

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

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 vs
usage: sl vs [<command>] [<args>...] [options]

Manage, delete, order compute instances

The available commands are:
  cancel          Cancel a running virtual server
  capture         Create an image the disk(s) of a virtual server
  create          Order and create a virtual server
                    (see sl vs create-options for choices)
  create-options  Output available available options when creating a VS
  detail          Output details about a virtual server
  dns             DNS related actions to a virtual server
  edit            Edit details of a virtual server
```

```
  list            List virtual servers on the account
  nic-edit        Edit NIC settings
  pause           Pauses an active virtual server
  power-off       Powers off a running virtual server
  power-on        Boots up a virtual server
  ready           Check if a virtual server has finished provisioning
  reboot          Reboots a running virtual server
  reload          Reload the OS on a VS based on its current configuration
  resume          Resumes a paused virtual server
  upgrade         Upgrades parameters of a virtual server
```

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

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

List virtual servers

Examples:
    sl vs list --datacenter=dal05
    sl vs list --network=100 --cpu=2
    sl vs list --memory='>= 2048'
    sl vs 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
```

CHAPTER 5

Contributing

5.1 Contribution Guide

This page explains how to get started contributing code to the SoftLayer API Python Bindings project.

5.1.1 Code Organization

- **docs** - Where The source to this documentation lives.
- **SoftLayer** - All the source lives under here.
  - **API** - Primary API client.
  - **CLI** - Code for the command-line interface.
    - **modules** - CLI Modules.
  - **managers** - API Managers. Abstractions to help use the API.

5.1.2 Setting Up A Dev Environment

Before working with the SoftLayer Python API client source, we strongly recommend that you know how to use Python's virtualization environment, virtualenv. Virtualenv allows you to create Python setups that are individually tailored to particular develpment efforts. Each environment can have its own set of libraries and even its own Python interpreter. This keeps them isolated, reducing the possibility of library conflicts between different projects.

After you have virtualenv, you should set up a virtual environment and activate it whenever you are working on softlayer-python. The commands needed to setup an environment and activate it might look something like this:

```
virtualenv --no-site-packages softlayer_env

source softlayer_env/bin/activate
```

Please refer to the virtualenv documentation for more information about creating, and working with virtual environments.

Once you have an appropriate environment, you will then download the SoftLayer API Python Bindings source code by following the *installation instructions*. Change into softlayer-python source directory and run the following to install the pre-requisites needed to run the development tests:

```
pip install -r tools/test-requirements.txt
```

5.1.3 Testing

The project has a mix of functional and unit tests. Before submitting changes to be integrated into the project, you should validate your code using tox. Simply issue the tox command from the root of the source tree:

```
tox
```

In addition to testing different versions of Python, tox checks for common mistakes in the code using Flake8. You should eliminate the simple errors reported by Flake8 before submitting your code.

The project's configuration instructs tox to test against many different versions of Python. A tox test will use as many of those as it can find on your local computer. Rather than installing all those versions, we recommend that you point the Travis integration tool at your github fork. Travis will run the test against the full suite of Python versions every time you push new code.

Running the tests in multiple environments, using tox, is very time consuming. If you wish to quickly run the tests in your own environment, you may do so using nose. The command to do that is:

```
python setup.py nosetests
```

5.1.4 Documentation

The project is documented in reStructuredText and built using Sphinx. If you have fabric installed, you simply need to run the following to build the docs:

```
fab make_html
```

The documentation will be built in *docs/_build/html*. If you don't have fabric, use the following commands.

```
cd docs
make html
```

The primary docs are built at Read the Docs.

5.1.5 Style

This project follows **PEP 8** and most of the style suggestions that pyflakes recommends. Run Flake8 regularly.

5.1.6 Contributing

Contributing to the Python API bindings follows the fork-pull-request model on github. The project uses Github's issues and pull requests to manage source control, bug fixes and new feature development regarding the API bindings and the CLI.

5.1.7 Developer Resources

Note: Full example module available *here*

5.2 Command-Line Interface Developer Guide

A CLI interface is broken into 4 major parts:

- module
- docblock
- action
- docblock
- docblock

5.2.1 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 vs* is *SoftLayer/CLI/modules/vs.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:
  parse   Parsing args example
  pretty  Formatted print example
  print   Print example
"""
```

There are some tenants for styling the doc blocks

- These are parsed with docopt so conform to the spec.
- Two spaces before commands in a command list and options in an option list.
- Align the descriptions two spaces after the loggest command/option
- If a description has to take up more than one line, indent two spaces past the current indention for all additional lines.
- Alphabetize all commands/option listings. For options, use the long name to judge ordering.

5.2.2 Action

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

```
class CLIRunnable(object):
    options = []  # set by subclass
    action = None  # set by subclass

    def __init__(self, client=None, env=None):
        self.client = client
        self.env = env

    def execute(self, args):
        pass
```

The required interfaces are:

- The docblock (__doc__) for docopt
- action class attribute
- def execute(self, args):

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

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

Print example
"""

    action = 'print'

    def execute(self, 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
```

5.2.3 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'

    execute(self, 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.

5.2.4 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'

    def execute(self, 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']
```

5.2.5 Accessing the API

API access is available via an attribute of the CLIRunnable instance called. In execute(), for example, you can refer to *self.client* to access an instanciated instance of *SoftLayer.API.Client*. Please refer to [using the api](API-Usage) for further details on howto use the *Client* object.

5.2.6 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

    def execute(self, 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
```

5.2.7 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 6

External Links

S

Symbols

A

C

D

E

F

G

H

I

L

M

N

P

Q

R

S

T

U

V

W