

---

# **ZenossAPIClient Documentation**

***Release 0.1.2***

**Mark Troyer**

**Nov 15, 2017**



---

## Contents:

---

<b>1</b>	<b>zenossapi</b>	<b>3</b>
1.1	apiclient Class . . . . .	3
<b>2</b>	<b>zenossapi Routers</b>	<b>5</b>
2.1	routers Base Class . . . . .	5
2.2	Device Router . . . . .	5
2.3	Events Router . . . . .	17
2.4	Jobs Router . . . . .	19
2.5	Properties Router . . . . .	21
2.6	Template Router . . . . .	24
<b>3</b>	<b>Indices and tables</b>	<b>33</b>
	<b>Python Module Index</b>	<b>35</b>



`zenossapi` is a python module for interacting with the Zenoss API an an object-oriented way. The philosophy here is to use objects to work with everything in the Zenoss API, and to try to normalize the various calls to the different routers. Thus *get* methods will always return an object, *list* methods will return data. All methods to add or create start with *add*, all remove or delete start with *delete*. As much as possible the methods try to hide the idiosyncrasies of the JSON API, and to do the work for you, for example by letting you use a device name instead of having to provide the full device UID for every call.



## 1.1 apiclient Class

Zenoss API Client Class

**class** zenossapi.apiclient.**Client** (*host=None, user=None, password=None, ssl\_verify=None*)

Bases: object

Client class to access the Zenoss JSON API

**get\_router** (*router*)

Instantiates and returns a Zenoss router object

**Parameters** **router** (*str*) – The API router to use

**get\_router\_methods** (*router*)

List all available methods for an API router

**Parameters** **router** (*str*) – The router to get methods from

**Returns**

**Return type** list

**get\_routers** ()

Gets the list of available Zenoss API routers

**Returns**

**Return type** list

**exception** zenossapi.apiclient.**ZenossAPIClientAuthenticationError**

Bases: exceptions.Exception

**exception** zenossapi.apiclient.**ZenossAPIClientError**

Bases: exceptions.Exception





### 2.1 routers Base Class

**class** zenossapi.routers.ZenossRouter (*url, headers, ssl\_verify, endpoint, action*)

Bases: object

Base class for Zenoss router classes

### 2.2 Device Router

Zenoss device\_router

**class** zenossapi.routers.device.DeviceRouter (*url, headers, ssl\_verify*)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with the Zenoss device router

**get\_device\_class** (*device\_class*)

Get a device class

**Parameters** *device\_class* (*str*) – The name of the device class

**Returns**

**Return type** *ZenossDeviceClass*

**get\_tree** (*device\_class*)

Get the tree structure of a device class.

**Parameters** *device\_class* (*str*) – Device class to use as the top of the tree

**Returns**

**Return type** dict

**list\_collectors** ()

Get the list of collectors.

**Returns****Return type** list**list\_device\_classes()**

Get the list of all device classes.

**Returns****Return type** list**list\_groups()**

Get the list of all groups.

**Returns****Return type** list**list\_locations()**

Get the list of all locations.

**Returns****Return type** list**list\_systems()**

Get the list of all systems.

**Returns****Return type** list**class** zenossapi.routers.device.**ZenossComponent** (*url, headers, ssl\_verify, device\_data*)Bases: *zenossapi.routers.device.DeviceRouter*

Class for Zenoss component objects

**delete()**

Delete the component.

**Returns** Response message**Return type** str**lock** (*updates=False, deletion=False, send\_event=False*)

Lock the component for changes.

**Parameters**

- **updates** (*bool*) – Lock for updates
- **deletion** (*bool*) – Lock for deletion
- **send\_event** (*bool*) – Send an event when an action is blocked by locking

**Returns** Response message**Return type** str**lock\_for\_deletion** (*send\_event=False*)

Lock the component for updates.

**Parameters** **send\_event** (*bool*) – Send an event when deletion is blocked by locking**Returns** Response message**Return type** str

**lock\_for\_updates** (*send\_event=False*)

Lock the component for updates.

**Parameters** **send\_event** (*bool*) – Send an event when updates are blocked by locking

**Returns** Response message

**Return type** str

**set\_monitored** (*monitor=True*)

Sets the monitored state for the component.

**Parameters** **monitor** (*bool*) – True to monitor, False to stop monitoring

**Returns** Response message

**Return type** str

**class** zenossapi.routers.device.**ZenossDevice** (*url, headers, ssl\_verify, device\_data*)

Bases: *zenossapi.routers.device.DeviceRouter*

Class for Zenoss device objects

**add\_local\_template** (*template*)

Add a local template to the device.

**Parameters** **template** (*str*) – Name of the new local template

**bind\_or\_unbind\_template** (*path, template*)

Binds a template to the device if it's unbound, or unbinds it if it's bound.

**Parameters**

- **path** (*str*) – Template's path, as given in the display label
- **template** (*str*) – Name of the template to bind/unbind

**delete** (*action, del\_events=False, del\_perf=True*)

Remove a device from its organizer, or delete it from Zenoss altogether.

**Parameters**

- **action** (*str*) – 'remove' to remove the devices from their organizer, 'delete' to delete them from Zenoss
- **del\_events** (*bool*) – Remove all events for the devices
- **del\_perf** (*bool*) – Remove all perf data for the devices

**Returns**

**Return type** bool

**delete\_local\_template** (*template*)

Remove a local template from the device.

**Parameters** **template** (*str*) – Name of the template to remove

**delete\_property** (*zproperty*)

Delete the locally set value of a property for a device

**Parameters** **zproperty** (*str*) – ID of the property to delete.

**Returns**

**Return type** bool

**get\_active\_templates()**

Get ZenossTemplate objects for all active templates on a device.

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_bound\_templates()**

Get ZenossTemplate objects templates that are bound to the device.

**Returns**

**Return type** list(*ZenosTemplate*)

**get\_component(component)**

Get a component object.

**Parameters** **component** (*str*) – Name of the component, e.g. 'hw/cpus/0'

**Returns**

**Return type** *ZenossComponent*

**get\_components(meta\_type=None, start=0, limit=50, sort='name', dir='ASC', name=None)**

Get component objects for all components on the device. Supports Pagination.

**Parameters**

- **meta\_type** (*str*) – Meta type of components to list
- **start** (*int*) – Offset to start device list from, default 0
- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'name'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'ASC'
- **name** (*str*) – Regular expression pattern to filter on

**Returns**

**Return type** list(*ZenossComponent*)

**get\_custom\_properties(params=None)**

Get the cProperties for the device

**Parameters** **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossCustomProperty objects.
}
```

**Return type** dict(int, list(*ZenossCustomProperty*))

**get\_custom\_property(cproperty)**

Get a custom property for the device

**Parameters** **cproperty** (*str*) – ID of the property to get.

**Returns**

**Return type** *ZenossCustomProperty*

**get\_local\_templates()**

Get ZenossTemplate objects for all locally defined templates.

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_overridable\_templates()**

Get ZenossTemplate objects for templates that can be overridden.

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_properties(params=None)**

Get the configuration properties for the device

**Parameters** **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossProperty objects.
}
```

**Return type** dict(int, list(*ZenossProperty*))

**get\_property(zproperty)**

Get a configuration property

**Parameters** **zproperty** (*str*) – The id of the property to get

**Returns**

**Return type** *ZenossProperty*

**get\_unbound\_templates()**

Get ZenossTemplate objects for available templates that are not bound to the device.

**Returns**

**Return type** list(*ZenossTemplate*)

**list\_active\_templates()**

Get the list of templates active on a device, both bound and local.

**Returns**

```
{
    'name': Template name,
    'label': Display label for the template,
}
```

**Return type** list(dict(str, str))

**list\_bound\_templates()**

Get the list of templates bound to a device, does not include local templates.

**Returns**

```
{
    'name': Template name,
    'label': Display label for the template,
}
```

**Return type** list(dict(str, str))

**list\_components** (*meta\_type=None, start=0, limit=50, sort='name', dir='ASC', name=None*)

Get a list of all the components on a device. Supports pagination.

**Parameters**

- **meta\_type** (*str*) – Meta type of components to list
- **start** (*int*) – Offset to start device list from, default 0
- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'name'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'ASC'
- **name** (*str*) – Regular expression pattern to filter on

**Returns**

```
{
    'total': Total number of components found.
    'hash': Hash check to determine if components have changed
    'components': List of components found
}
```

**Return type** dict(int, str, list)

**list\_custom\_properties** ()

List the custom properties for the device

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_local\_properties** ()

List the locally defined configuration properties for the device

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_local\_templates** ()

Get the list of monitoring templates defined locally on a device.

**Returns**

**Return type** list

**list\_overridable\_templates** ()

Get the list of available templates on a device that can be overridden.

**Returns**

```
{
    'name': Template name,
    'label': Display label for the template,
}
```

**Return type** list(dict(str, str))

**list\_properties** (*params=None, sort=None, sort\_dir='ASC'*)

List the configuration properties for the device

#### Parameters

- **params** (*dict*) – Search parameters to filter the properties list on.
- **sort** (*str*) – Sort key for the properties list.
- **sort\_dir** (*str*) – Sort direction, either ASC or DESC

#### Returns

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_unbound\_templates** ()

Get the list of available templates that are not bound to the device.

#### Returns

```
{
    'name': Template name,
    'label': Display label for the template,
}
```

**Return type** list(dict(str, str))

**list\_user\_commands** ()

Get the list of user commands for a device.

#### Returns

```
{
    name: Name of the user command
    description: Command description
}
```

**Return type** dict(str, str)

**lock** (*updates=False, deletion=False, send\_event=False*)

Lock the device for changes.

#### Parameters

- **updates** (*bool*) – Lock for updates
- **deletion** (*bool*) – Lock for deletion
- **send\_event** (*bool*) – Send an event when an action is blocked by locking

**Returns** Response message

**Return type** str

**lock\_for\_deletion** (*send\_event=False*)

Lock the device for updates.

**Parameters** **send\_event** (*bool*) – Send an event when deletion is blocked by locking

**Returns** Response message

**Return type** str

**lock\_for\_updates** (*send\_event=False*)

Lock the device for updates.

**Parameters** **send\_event** (*bool*) – Send an event when updates are blocked by locking

**Returns** Response message

**Return type** str

**move** (*device\_class*)

Move the device to a different device class

**Parameters** **device\_class** (*str*) – Name of the device class to move the device into

**Returns** uuid of the Job Manager job for the move

**Return type** str

**reidentify** (*new\_id*)

Change the device's id in Zenoss. Note that changing the device id will cause the loss of all graph data for the device.

**Parameters** **new\_id** (*str*) – New ID for the device

**remodel** ()

Remodel the device.

**Returns** uuid of the Job Manager job for the remodel

**Return type** str

**reset\_bound\_templates** ()

Remove all bound templates from device.

**reset\_ip\_address** (*ip\_address=""*)

Reset the IP address of the device to ip\_address if specified or to the result of a DNS lookup if not.

**Parameters** **ip\_address** (*str*) – IP address to set device to

**Returns** Response message

**Return type** str

**set\_bound\_templates** (*templates*)

Set a list of templates as bound to a device.

**Parameters** **templates** (*list*) – List of template names

**set\_collector** (*collector*)

Set the collector for the device.

**Parameters** **collector** (*str*) – The collector to use for the device

**Returns** uuid of the Job Manager job for the change

**Return type** str



**set\_priority** (*priority*)

Set the priority for the device.

**Parameters** **priority** (*int*) – Numeric value for the desired priority

**Returns** Reponse message

**Return type** str

**set\_production\_state** (*production\_state*)

Set the production state for the device.

**Parameters** **production\_state** (*int*) – Numeric value for the desired production state.

**Returns** Response message

**Return type** str

**set\_property** (*zproperty, value=None*)

Set the value of a configuration property

**Parameters**

- **zproperty** (*str*) – The id of the property to set a value for
- **value** (*str*) – The value to set for the property

**Returns**

**Return type** bool

**class** zenossapi.routers.device.**ZenossDeviceClass** (*url, headers, ssl\_verify, device\_class\_data*)

Bases: [zenossapi.routers.device.DeviceRouter](#)

Class for Zenoss device class objects

**add\_device** (*device\_name, title="", ip\_address="", location=None, systems=None, groups=None, model=False, collector='localhost', production\_state=500, comments="", priority=3, snmp\_community="", snmp\_port=161, rack\_slot="", hw\_manufacturer="", hw\_product\_name="", os\_manufacturer="", os\_product\_name="", asset\_tag="", serial\_number="", windows\_user="", windows\_password="", zcommand\_user="", zcommand\_password="", configuration\_properties=None, custom\_properties=None*)

Add a new device to the device class.

**Parameters**

- **device\_name** (*str*) – Name of the new device, will be the device id
- **title** (*str*) – Optional title for the device, default is to match the device\_name
- **ip\_address** (*str*) – Ip address for the device, default is to derive this from DNS based on device\_name
- **location** (*str*) – Location for the device
- **systems** (*list[ str ]*) – List of systems for the device
- **groups** (*list[ str ]*) – List of groups for the device
- **model** (*bool*) – Set to True to model the device automatically after creation
- **collector** (*str*) – Collector to use for the device
- **production\_state** (*int*) – Numerical production state for the device, default is 500 (Pre-Production)
- **comments** (*str*) – Comments for the device

- **priority** (*int*) – Numerical priority for the device, default is 3 (Normal)
- **snmp\_community** (*str*) – SNMP community string for the device
- **snmp\_port** (*int*) – SNMP port for the device
- **rack\_slot** (*str*) – Rack slot description
- **hw\_manufacturer** (*str*) – Hardware manufacturer name, default is to derive by modeling
- **hw\_product\_name** (*str*) – Hardware product name, default is to derive by modeling
- **os\_manufacturer** (*str*) – Operating system developer, default is to derive by modeling
- **os\_product\_name** (*str*) – Operating system name, default is to derive by modeling
- **asset\_tag** (*str*) – Device’s inventory asset tag
- **serial\_number** (*str*) – Device’s serial number
- **windows\_user** (*str*) – Username for Windows device monitoring
- **windows\_password** (*str*) – Password for the windows\_user
- **zcommand\_user** (*str*) – Username for SSH-based monitoring user
- **zcommand\_password** (*str*) – Password for the zcommand\_user
- **configuration\_properties** (*dict*) – Key/value pairs for setting Configuration Properties for the device
- **custom\_properties** (*dict*) – Key/value pairs for setting Custom Properties for the device

**Returns** ID of the add device job

**Return type** str

**add\_subclass** (*name*, *description=""*, *connection\_info=None*)

Add a new subclass to the device class.

**Parameters**

- **name** (*str*) – Name of the new subclass
- **description** (*str*) – Description for the new subclass
- **connection\_info** (*list*) – zProperties that represent the credentials for access in the subclass

**delete\_property** (*zproperty*)

Delete the locally set value of a property for a device class

**Parameters** **zproperty** (*str*) – ID of the property to delete.

**Returns**

**Return type** bool

**get\_custom\_properties** (*params=None*)

Get the cProperties for the device class

**Parameters** **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossCustomProperty objects.
}
```

**Return type** dict(int, list(*ZenossCustomProperty*))

**get\_custom\_property** (*cproperty*)

Get a custom property for the device class

**Parameters** **cproperty** (*str*) – ID of the property to get.

**Returns**

**Return type** *ZenossCustomProperty*

**get\_device** (*device\_name*)

Get a device from the device class

**Parameters** **device\_name** (*str*) – The name of the device to get

**Returns**

**Return type** *ZenossDevice*

**get\_devices** (*params=None, start=0, limit=50, sort='name', dir='ASC'*)

Get the devices contained in a device class. Supports pagination.

**Parameters**

- **params** (*dict*) – Key/value filters for the search, options are name, ipAddress, device-Class, or productionState
- **start** (*int*) – Offset to start device list from, default 0
- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'name'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'ASC'

**Returns**

```
{
    'total': Total number of devices found
    'hash': Hashcheck to determine if any devices have changed,
    'devices': ZenossDevice objects,
}
```

**Return type** dict(int, str, list(*ZenossDevice*))

**get\_properties** (*params=None*)

Get the configuration properties for the device class

**Parameters** **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossProperty objects.
}
```

**Return type** dict(int, list(*ZenossProperty*))

**get\_property** (*zproperty*)

Get a configuration property

**Parameters** **zproperty** (*str*) – The id of the property to get

**Returns**

**Return type** *ZenossProperty*

**list\_custom\_properties** ()

List the custom properties for the device class

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_devices** (*params=None, keys=None, start=0, limit=50, sort='name', dir='ASC'*)

List the devices contained in a device class. Supports pagination.

**Parameters**

- **params** (*dict*) – Key/value filters for the search, options are name, ipAddress, device-Class, or productionState
- **keys** (*list*) – List of keys to return for the devices found
- **start** (*int*) – Offset to start device list from, default 0
- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'name'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'ASC'

**Returns**

**Return type** dict

**list\_local\_properties** ()

List the locally defined configuration properties for the device class

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_properties** (*params=None, sort=None, sort\_dir='ASC'*)

List the configuration properties for the device class

**Parameters**

- **params** (*dict*) – Search parameters to filter the properties list on.
- **sort** (*str*) – Sort key for the properties list.
- **sort\_dir** (*str*) – Sort direction, either ASC or DESC

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))**set\_property** (*zproperty*, *value=None*)

Set the value of a configuration property

**Parameters**

- **zproperty** (*str*) – The id of the property to set a value for
- **value** (*str*) – The value to set for the property

**Returns****Return type** bool

## 2.3 Events Router

Zenoss evconsole\_router

**class** zenossapi.routers.events.**EventsRouter** (*url*, *headers*, *ssl\_verify*)Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with Zenoss events.

**add\_event** (*summary*, *device*, *severity*, *component=None*, *event\_class\_key=None*,  
*event\_class='/Status'*)

Create a new Zenoss event.

**Parameters**

- **summary** (*str*) – Summary for the new event
- **device** (*str*) – Device ID for the new event
- **component** (*str*) – Component UID for the new event
- **severity** (*str*) – Severity to assign the new event, must be one of Critical, Error, Warning, Info, Debug, or Clear
- **event\_class\_key** (*str*) – The Event Class Key to assign to the event
- **event\_class** (*str*) – Event Class for the event

**Returns****Return type** *ZenossEvent***clear\_heartbeat** (*collector*, *daemon*)

Clear a heartbeat event for a specific daemon.

**Parameters**

- **collector** (*str*) – Collector the daemon is running in, e.g. slvcollector
- **daemon** (*str*) – Monitoring daemon to clear the heartbeat event for, e.g. zencommand

**clear\_heartbeats** ()

Clear all heartbeat events

**Returns** True on success

**Return type** bool

**get\_config()**

Get the event handling configuration.

**Returns**

**Return type** list(dict)

**get\_event\_by\_evid(evid)**

Get an event by its event id

**Parameters** **evid** (*str*) – The event id

**Returns**

**Return type** *ZenossEvent*

**get\_open\_events(limit=10, start=0, sort='lastTime', sort\_dir='DESC')**

Get all open events (new or acknowledged state)

**Parameters**

- **limit** (*int*) – Maximum number of events to return
- **start** (*int*) – Minimum index of events to get
- **sort** (*str*) – Sort key for events list
- **sort\_dir** (*str*) – Sort direction, ASC or DESC

**Returns**

**Return type** list(*ZenossEvent*)

**get\_open\_production\_events(limit=10, start=0, sort='lastTime', sort\_dir='DESC')**

Get all open events (new or acknowledged state) for devices with a production state of Production

**Parameters**

- **limit** (*int*) – Maximum number of events to return
- **start** (*int*) – Minimum index of events to get
- **sort** (*str*) – Sort key for events list
- **sort\_dir** (*str*) – Sort direction, ASC or DESC

**Returns**

**Return type** list(*ZenossEvent*)

**list\_open\_events(limit=10, start=0, sort='lastTime', sort\_dir='DESC')**

Get a list of all open events (new or acknowledged state)

**Parameters**

- **limit** (*int*) – Maximum number of events to return
- **start** (*int*) – Minimum index of events to get
- **sort** (*str*) – Sort key for events list
- **sort\_dir** (*str*) – Sort direction, ASC or DESC

**Returns**

**Return type** dict

**list\_open\_production\_events** (*limit=10, start=0, sort='lastTime', sort\_dir='DESC'*)

Get a list of all open events (new or acknowledged state) for devices with a production state of Production

**Parameters**

- **limit** (*int*) – Maximum number of events to return
- **start** (*int*) – Minimum index of events to get
- **sort** (*str*) – Sort key for events list
- **sort\_dir** (*str*) – Sort direction, ASC or DESC

**Returns**

**Return type** dict

**update\_config** (*config\_values*)

Update the Zenoss event handling configuration.

**Parameters** **config\_values** (*dict*) – Key/value pairs of the config values to change.

**class** zenossapi.routers.events.**ZenossEvent** (*url, headers, ssl\_verify, event\_data*)

Bases: *zenossapi.routers.events.EventsRouter*

Class for Zenoss event objects

**ack** ()

Acknowledge the event.

**close** ()

Close the event.

**reopen** ()

Reopen (unacknowledge or unclosed) the event.

**update\_log** (*message*)

Add an entry to the event's log

**Parameters** **message** (*str*) – Log entry to add

## 2.4 Jobs Router

Zenoss jobs\_router

**class** zenossapi.routers.jobs.**JobsRouter** (*url, headers, ssl\_verify*)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with the Zenoss device router

**get\_job** (*job*)

Get a ZenossJob object by the job's uuid

**Parameters** **job** (*str*) – uuid of the job

**Returns**

**Return type** *ZenossJob*

**get\_jobs** (*start=0, limit=50, sort='scheduled', dir='ASC'*)

Get ZenossJob objects for Job Manager jobs. Supports pagination.

**Parameters**

- **start** (*int*) – Offset to start device list from, default 0

- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'scheduled'. Other sort keys are 'started', 'finished', 'status', 'type' and 'user'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'ASC'

**Returns****Return type** list(*ZenossJob*)**list\_jobs** (*start=0, limit=50, sort='scheduled', dir='DESC'*)

List all Job Manager jobs, supports pagination.

**Parameters**

- **start** (*int*) – Offset to start device list from, default 0
- **limit** (*int*) – The number of results to return, default 50
- **sort** (*str*) – Sort key for the list, default is 'scheduled'. Other sort keys are 'started', 'finished', 'status', 'type' and 'user'
- **dir** (*str*) – Sort order, either 'ASC' or 'DESC', default is 'DESC'

**Returns**

```
{
  'total': (int) Total number of jobs,
  'jobs': {
    'description': (str) Job description,
    'finished': (int) Time the job finished in timestamp format,
    'scheduled': (int) Time the job was scheduled in timestamp_
↵format,
    'started': (int) Time the job started in timestamp format,
    'status': (str) Status of the job,
    'type': (str) Job type,
    'uid': (str) JobManager UID - /zport/dmd/JobManager,
    'user': (str) User who scheduled the job,
    'uuid': (str) UUID of the job,
  }
}
```

**Return type** dict(int, dict(str, int, int, int, str, str, str, str, str))**class** zenossapi.routers.jobs.**ZenossJob** (*url, headers, ssl\_verify, job\_data*)Bases: *zenossapi.routers.jobs.JobsRouter*

Class for Zenoss job objects

**abort** ()

Abort the job.

**Returns****Return type** bool**delete** ()

Delete the job.

**Returns** Job ID**Return type** list



**get\_log()**

Get the log for the job.

**Returns**

```
{
    'logfile': Filesystem path of the log file,
    'maxLimit': True or False,
    'content': Log file lines
}
```

**Return type** dict(str, bool, list)

## 2.5 Properties Router

Zenoss properties\_router

**class** zenossapi.routers.properties.**PropertiesRouter** (*url, headers, ssl\_verify*)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with Zenoss properties.

**delete\_property** (*uid, zproperty*)

Delete a ZenProperty.

**Parameters**

- **uid** (*str*) – UID to delete the property from
- **zproperty** (*str*) – ID of the property to delete.

**Returns**

**Return type** bool

**get\_custom\_properties** (*uid, params=None*)

Get ZenossCustomProperties objects for the cProperties of a uid context.

**Parameters**

- **uid** (*str*) – UID of the object to get properties for.
- **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossCustomProperty objects.
}
```

**Return type** dict(int, list(*ZenossCustomProperty*))

**get\_custom\_property** (*uid, cproperty*)

Get a single ZenossCustomProperty

**Parameters**

- **uid** (*str*) – UID to get the property of.
- **cproperty** (*str*) – ID of the property to get.

**Returns**

Return type *ZenossCustomProperty*

**get\_local\_properties** (*uid*)

Get ZenossProperty objects for the local properties of a specified uid.

**Parameters** *uid* (*str*) – UID to get local properties for.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossProperty objects.
}
```

Return type dict(int, list(*ZenossProperty*))

**get\_properties** (*uid*, *params=None*)

Get ZenossProperties objects for the properties of a uid context.

**Parameters**

- **uid** (*str*) – UID of the object to get properties for.
- **params** (*dict*) – Search parameters for filter the properties on.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of ZenossProperty objects.
}
```

Return type dict(int, list(*ZenossProperty*))

**get\_property** (*uid*, *zproperty*)

Get a single ZenossProperty

**Parameters**

- **uid** (*str*) – UID to get the property of.
- **zproperty** (*str*) – ID of the property to get.

**Returns**

Return type *ZenossProperty*

**list\_custom\_properties** (*uid*, *params=None*, *sort=None*, *sort\_dir='ASC'*)

Get a list of cProperties for the uid context.

**Parameters**

- **uid** (*str*) – UID of the object to list properties for.
- **params** (*dict*) – Search parameters to filter the properties list on.
- **sort** (*str*) – Sort key for the properties list.
- **sort\_dir** (*str*) – Sort direction, either ASC or DESC

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_local\_properties** (*uid*)

Get a list of properties set locally to the specified UID.

**Parameters** *uid* (*str*) – UID to get local properties for.

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**list\_properties** (*uid*, *params=None*, *sort=None*, *sort\_dir='ASC'*)

Get a list of ZenProperties for the uid context.

**Parameters**

- **uid** (*str*) – UID of the object to list properties for.
- **params** (*dict*) – Search parameters to filter the properties list on.
- **sort** (*str*) – Sort key for the properties list.
- **sort\_dir** (*str*) – Sort direction, either ASC or DESC

**Returns**

```
{
    'total': Total count of properties returned.
    'properties': List of properties found.
}
```

**Return type** dict(int, list(dict))

**set\_property\_value** (*uid*, *zproperty*, *value=None*)

Sets (or updates) the local value of a property

**Parameters** *value* – The new value for the property, type varies by property.

**Returns**

**Return type** bool

**class** zenossapi.routers.properties.**ZenossCustomProperty** (*url*, *headers*, *ssl\_verify*,  
*property\_data*)

Bases: *zenossapi.routers.properties.PropertiesRouter*

Class for Zenoss CustomProperties

**delete** ()

Delete the local instance of a property.

**Returns**

**Return type** bool

**set\_value** (*path=None*, *value=None*)

Sets (or updates) the local value of a custom property

**Parameters**

- **path** (*str*) – UID of the node to set the property for.

- **value** (*str*) – The new value for the property, type varies by property.

**Returns****Return type** bool

**class** zenossapi.routers.properties.**ZenossProperty** (*url*, *headers*, *ssl\_verify*, *property\_data*)

Bases: *zenossapi.routers.properties.PropertiesRouter*

Class for ZenProperties

**delete** ()

Delete the local instance of a property.

**Returns****Return type** bool

**set\_value** (*path=None*, *value=None*)

Sets (or updates) the local value of a property

**Parameters**

- **path** (*str*) – UID of the node to set the property for.
- **value** (*str*) – The new value for the property, type varies by property.

**Returns****Return type** bool

## 2.6 Template Router

Zenoss template\_router

**class** zenossapi.routers.template.**TemplateRouter** (*url*, *headers*, *ssl\_verify*)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with the Zenoss template router

**add\_data\_point\_to\_graph** (*datapoint*, *graph*, *include\_thresholds=False*)

Adds a data point to a graph.

**Parameters**

- **datapoint** (*str*) – Uid of the data point to add
- **graph** (*str*) – Uid of the graph to add the data point to
- **include\_thresholds** (*bool*) – Set to True to include the related thresholds for the data point

**Returns****Return type** dict

**add\_local\_template** (*zenoss\_object*, *name*)

Adds a local template to an object.

**Parameters**

- **zenoss\_object** (*str*) – Uid of the object to add the local template to
- **name** – Unique name for the new local template

**add\_template** (*target*, *name*)

Adds a template to a device class.

**Parameters**

- **target** (*str*) – The uid of the target device class
- **name** (*str*) – Unique name of the template to add

**Returns**

**Return type** *ZenossTemplate*

**delete\_local\_template** (*zenoss\_object*, *name*)

Builds the request data for deleting a local template to an object.

**Parameters**

- **object** (*str*) – Uid of the object to remove the local template from
- **name** – Unique name of the new local template

**delete\_template** (*device\_class*, *template*)

Removes a template.

**Parameters**

- **device\_class** (*str*) – Name of the device class where the template is defined
- **template** (*str*) – Name of the template to remove

**Returns**

**Return type** dict

**get\_all\_templates** ()

Returns all defined templates.

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_data\_source\_types** ()

Gets the list of available data source types.

**Returns**

**Return type** list

**get\_device\_class\_templates** (*device\_class*)

Gets the defined templates for a device class

**Parameters** **device\_class** (*str*) – Device class to get templates for

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_object\_templates** (*zenoss\_object*)

Gets the templates bound to a specific object (monitored resource or component)

**Parameters** **zenoss\_object** (*str*) – The uid of the object, e.g. `Devices/Server/Zuora/Aspose/devices/10.aspose.prod.slv.zuora`

**Returns**

**Return type** list(*ZenossTemplate*)

**get\_template** (*device\_class*, *template*)

Get a Zenoss template

**Parameters**

- **device\_class** (*str*) – Name of the device class where the template is defined
- **template** (*str*) – Name of the template to get

**Returns**

**Return type** *ZenossTemplate*

**get\_threshold\_types** ()

Gets the list of available threshold types.

**Returns**

**Return type** list

**list\_all\_templates** ()

Returns all defined templates as a list of tuples containing the template UID and description.

**Returns**

**Return type** list(*ZenossTemplate*)

**list\_device\_class\_templates** (*device\_class*)

Returns the defined templates for a device class as a list of tuples containing the template UID and description.

**Parameters** **device\_class** (*str*) – Device class to list templates for

**Returns**

**Return type** list(str)

**set\_properties** (*properties*)

Sets properties of an object.

**Parameters** **properties** (*dict*) – Properties and values to set

**class** zenossapi.routers.template.**ZenossDataPoint** (*url*, *headers*, *ssl\_verify*, *dp\_data*)

Bases: *zenossapi.routers.template.TemplateRouter*

Class for Zenoss data points

**add\_to\_graph** (*graph*, *include\_thresholds=False*)

Adds a data point to a graph.

**Parameters**

- **graph** (*str*) – Name of the graph to add the data point to
- **include\_thresholds** (*bool*) – Set to True to include the related thresholds for the data point

**Returns**

**Return type** dict

**delete** ()

Deletes a data point from a template.

**Returns**

**Return type** dict

**make\_counter()**

Sets the RRD Type of the data point to COUNTER

**Returns**

**Return type** bool

**make\_gauge()**

Sets the RRD Type of the data point to GAUGE

**Returns**

**Return type** bool

**set\_threshold(threshold, threshold\_type)**

Adds a threshold for the data point

**Parameters**

- **threshold** (*str*) – Name of the threshold to add
- **threshold\_type** (*str*) – Type of the new threshold, must be one of the types returned by `get_threshold_types()`

**Returns**

**Return type** *ZenossThreshold*

**class** zenossapi.routers.template.**ZenossDataSource** (*url, headers, ssl\_verify, ds\_data*)

Bases: *zenossapi.routers.template.TemplateRouter*

Class for Zenoss template data sources

**add\_data\_point(datapoint)**

Adds a data point to a data source.

**Parameters** **datapoint** (*str*) – Name of the new data point

**Returns**

**Return type** *ZenossDataPoint*

**delete()**

Deletes a data source from a template.

**Returns**

**Return type** dict

**delete\_data\_point(datapoint)**

Deletes a data point from a template.

**Parameters** **datapoint** (*str*) – Name of the data point to remove

**Returns**

**Return type** dict

**get\_data\_point(datapoint)**

Get a particular data point.

**Parameters** **datapoint** (*str*) – Name of the data point to get details for

**Returns**

**Return type** *ZenossDataPoint*

**get\_data\_points()**

Get all the data points for a datasource.

**Returns**

**Return type** list(*ZenossDataPoint*)

**list\_data\_points()**

Returns all the data points for a datasource as a list.

**Returns**

**Return type** list(str)

**class** zenossapi.routers.template.**ZenossGraph**(url, headers, ssl\_verify, graph\_data)

Bases: *zenossapi.routers.template.TemplateRouter*

Class for Zenoss graphs

**add\_graph\_threshold**(threshold)

Adds a threshold to a graph.

**Parameters** **threshold** (str) – Uid of the threshold to add

**Returns**

**Return type** dict

**add\_point**(datasource, datapoint, include\_thresholds=False)

Adds a data point to a graph.

**Parameters**

- **datasource** (str) – Name of the data source holding the data point
- **datapoint** (str) – Name of the data point to add
- **include\_thresholds** (bool) – Set to True to include the related thresholds for the data point

**Returns**

**Return type** dict

**delete\_point**(datapoint)

Deletes a data point from a graph.

**Parameters** **datapoint** (str) – Name of the data point to remove

**Returns**

**Return type** dict

**get\_points()**

Gets the data points of a graph.

**Returns**

**Return type** list(*ZenossDataPoint*)

**list\_points()**

Returns the data points of a graph as a list.

**Returns**

**Return type** list(str)



**set\_graph\_properties** (*properties*)

Set the properties for a graph.

**Parameters** **properties** (*dict*) – Properties and values to set

**Returns**

**Return type** dict

**set\_point\_sequence** (*datapoints*)

Sets the order of data points in a graph.

**Parameters** **datapoints** (*list*) – List of data point names in the desired order

**Returns**

**Return type** dict

**set\_zero\_baseline** ()

Set the minimum value of a graph display to zero. By default Zenoss graph scale is dynamic, meaning the display can be skewed because the minimum value isn't fixed.

**class** zenossapi.routers.template.**ZenossTemplate** (*url*, *headers*, *ssl\_verify*, *template\_data*)

Bases: *zenossapi.routers.template.TemplateRouter*

Class for Zenoss Template objects

**add\_data\_source** (*datasource*, *type*)

Adds a data source to a template.

**Parameters**

- **datasource** (*str*) – Name of the new data source
- **type** (*str*) – Type of the new data source, must be one of the types returned by `get_data_source_types()`

**Returns**

**Return type** *ZenossDataSource*

**add\_graph** (*graph*)

Add a new graph to a template.

**Parameters** **graph** (*str*) – Name for the new graph

**Returns**

**Return type** *ZenossGraph*

**add\_threshold** (*threshold*, *threshold\_type*, *datapoints*)

Adds a threshold to a template.

**Parameters**

- **threshold** (*str*) – Name of the new threshold
- **threshold\_type** (*str*) – Type of the new threshold, must be one of the types returned by `get_threshold_types()`
- **datapoints** (*list*) – List of datapoints to select for the threshold

**Returns**

**Return type** *ZenossThreshold*

**copy** (*target*)

Copy a template to another device or device class.

**Parameters** **target** (*str*) – Uid of the device or device class to copy to

**Returns**

**Return type** *ZenossTemplate*

**delete** ()

Removes a template.

**Returns**

**Return type** dict

**delete\_data\_source** (*datasource*)

Deletes a data source from a template.

**Parameters** **datasource** (*str*) – Name the data source to remove

**Returns**

**Return type** dict

**delete\_threshold** (*threshold*)

Deletes a threshold.

**Parameters** **threshold** (*str*) – Name of the threshold to remove

**Returns**

**Return type** dict

**get\_data\_points** ()

Get all the data points in a template.

**Returns**

**Return type** list(*ZenossDataPoint*)

**get\_data\_source** (*datasource*)

Get a particular data source.

**Parameters** **datasource** (*str*) – Name of the data source to get

**Returns**

**Return type** *ZenossDataSource*

**get\_data\_sources** ()

Gets data sources configured for a template.

**Returns**

**Return type** list(*ZenossDataSource*)

**get\_graph** (*graph*)

Get a particular graph.

**Parameters** **graph** (*str*) – Name of the graph to get the definition of

**Returns**

**Return type** *ZenossGraph*

**get\_graphs** ()

Get the graphs defined for a template.

**Returns****Return type** list(*ZenossGraph*)**get\_threshold** (*threshold*)

Get a particular threshold.

**Parameters** **threshold** (*str*) – Name of the threshold to get details on**Returns****Return type** *ZenossThreshold***get\_thresholds** ()

Gets the thresholds of a template.

**Returns****Return type** list(*ZenossThresholds*)**list\_data\_points** ()

Returns all the data points in a template as a list.

**Returns****Return type** list(*str*)**list\_data\_sources** ()

Returns data sources configured for a template as a list.

**Returns****Return type** list(*str*)**list\_graphs** ()

Returns the graphs defined for a template as a list.

**Returns****Return type** list(*str*)**list\_thresholds** ()

Returns the thresholds of a template as a list.

**Returns****Return type** list(*str*)

```
class zenossapi.routers.template.ZenossThreshold (url, headers, ssl_verify, threshold_data)
```

Bases: *zenossapi.routers.template.TemplateRouter*

Class for Zenoss thresholds

**delete** ()

Deletes a threshold.

**Returns****Return type** dict**set\_max** (*maxval*)

Sets the threshold value for a MinMaxThreshold checking the max value of a data point.

**Parameters** **maxval** (*str*) – Maximum value for the data point before alerting**Returns****Return type** bool

**set\_min** (*minval*)

Sets the threshold value for a MinMaxThreshold checking the minimum value of a data point.

**Parameters** **minval** (*str*) – Minimum value for the data point before alerting

**Returns**

**Return type** bool

## CHAPTER 3

---

### Indices and tables

---

- `genindex`
- `modindex`
- `search`



### Z

- `zenossapi.apiclient`, 3
- `zenossapi.routers`, 5
  - `zenossapi.routers.device`, 5
  - `zenossapi.routers.events`, 17
  - `zenossapi.routers.jobs`, 19
  - `zenossapi.routers.properties`, 21
  - `zenossapi.routers.template`, 24





## A

abort() (zenossapi.routers.jobs.ZenossJob method), 20  
 ack() (zenossapi.routers.events.ZenossEvent method), 19  
 add\_data\_point() (zenossapi.routers.template.ZenossDataSource method), 27  
 add\_data\_point\_to\_graph() (zenossapi.routers.template.TemplateRouter method), 24  
 add\_data\_source() (zenossapi.routers.template.ZenossTemplate method), 29  
 add\_device() (zenossapi.routers.device.ZenossDeviceClass method), 13  
 add\_event() (zenossapi.routers.events.EventsRouter method), 17  
 add\_graph() (zenossapi.routers.template.ZenossTemplate method), 29  
 add\_graph\_threshold() (zenossapi.routers.template.ZenossGraph method), 28  
 add\_local\_template() (zenossapi.routers.device.ZenossDevice method), 7  
 add\_local\_template() (zenossapi.routers.template.TemplateRouter method), 24  
 add\_point() (zenossapi.routers.template.ZenossGraph method), 28  
 add\_subclass() (zenossapi.routers.device.ZenossDeviceClass method), 14  
 add\_template() (zenossapi.routers.template.TemplateRouter method), 24  
 add\_threshold() (zenossapi.routers.template.ZenossTemplate method), 29  
 add\_to\_graph() (zenoss-

api.routers.template.ZenossDataPoint method), 26

## B

bind\_or\_unbind\_template() (zenossapi.routers.device.ZenossDevice method), 7

## C

clear\_heartbeat() (zenossapi.routers.events.EventsRouter method), 17  
 clear\_heartbeats() (zenossapi.routers.events.EventsRouter method), 17  
 Client (class in zenossapi.apiclient), 3  
 close() (zenossapi.routers.events.ZenossEvent method), 19  
 copy() (zenossapi.routers.template.ZenossTemplate method), 29

## D

delete() (zenossapi.routers.device.ZenossComponent method), 6  
 delete() (zenossapi.routers.device.ZenossDevice method), 7  
 delete() (zenossapi.routers.jobs.ZenossJob method), 20  
 delete() (zenossapi.routers.properties.ZenossCustomProperty method), 23  
 delete() (zenossapi.routers.properties.ZenossProperty method), 24  
 delete() (zenossapi.routers.template.ZenossDataPoint method), 26  
 delete() (zenossapi.routers.template.ZenossDataSource method), 27  
 delete() (zenossapi.routers.template.ZenossTemplate method), 30  
 delete() (zenossapi.routers.template.ZenossThreshold method), 31

delete_data_point()	(zenoss- api.routers.template.ZenossDataSource method), 27	8	get_custom_properties()	(zenoss- api.routers.device.ZenossDeviceClass method), 14
delete_data_source()	(zenoss- api.routers.template.ZenossTemplate method), 30	14	get_custom_properties()	(zenoss- api.routers.properties.PropertiesRouter method), 21
delete_local_template()	(zenoss- api.routers.device.ZenossDevice method), 7	7	get_custom_property()	(zenoss- api.routers.device.ZenossDevice method), 8
delete_local_template()	(zenoss- api.routers.template.TemplateRouter method), 25	25	get_custom_property()	(zenoss- api.routers.device.ZenossDeviceClass method), 15
delete_point()	(zenossapi.routers.template.ZenossGraph method), 28	28	get_custom_property()	(zenoss- api.routers.properties.PropertiesRouter method), 21
delete_property()	(zenoss- api.routers.device.ZenossDevice method), 7	7	get_data_point()	(zenoss- api.routers.template.ZenossDataSource method), 27
delete_property()	(zenoss- api.routers.device.ZenossDeviceClass method), 14	14	get_data_points()	(zenoss- api.routers.template.ZenossDataSource method), 27
delete_property()	(zenoss- api.routers.properties.PropertiesRouter method), 21	21	get_data_points()	(zenoss- api.routers.template.ZenossTemplate method), 30
delete_template()	(zenoss- api.routers.template.TemplateRouter method), 25	25	get_data_source()	(zenoss- api.routers.template.ZenossTemplate method), 30
delete_threshold()	(zenoss- api.routers.template.ZenossTemplate method), 30	30	get_data_source_types()	(zenoss- api.routers.template.TemplateRouter method), 25
DeviceRouter (class in zenossapi.routers.device), 5		5	get_data_sources()	(zenoss- api.routers.template.ZenossTemplate method), 30
<b>E</b>			get_device()	(zenossapi.routers.device.ZenossDeviceClass method), 15
EventsRouter (class in zenossapi.routers.events), 17		17	get_device_class()	(zenoss- api.routers.device.DeviceRouter method), 5
<b>G</b>			get_device_class_templates()	(zenoss- api.routers.template.TemplateRouter method), 25
get_active_templates()	(zenoss- api.routers.device.ZenossDevice method), 7	7	get_devices()	(zenossapi.routers.device.ZenossDeviceClass method), 15
get_all_templates()	(zenoss- api.routers.template.TemplateRouter method), 25	25	get_event_by_evid()	(zenoss- api.routers.events.EventsRouter method), 18
get_bound_templates()	(zenoss- api.routers.device.ZenossDevice method), 8	8	get_graph()	(zenossapi.routers.template.ZenossTemplate method), 30
get_component()	(zenoss- api.routers.device.ZenossDevice method), 8	8	get_graphs()	(zenossapi.routers.template.ZenossTemplate method), 30
get_components()	(zenoss- api.routers.device.ZenossDevice method), 8	8	get_job()	(zenossapi.routers.jobs.JobsRouter method), 19
get_config()	(zenossapi.routers.events.EventsRouter method), 18	18	get_jobs()	(zenossapi.routers.jobs.JobsRouter method), 19
get_custom_properties()	(zenoss- api.routers.device.ZenossDevice method),			

get_local_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 22	get_unbound_templates()	(zenoss-api.routers.device.ZenossDevice method), 9
get_local_templates()	(zenoss-api.routers.device.ZenossDevice method), 8	<b>J</b>	
get_log()	(zenossapi.routers.jobs.ZenossJob method), 20	<b>L</b>	
get_object_templates()	(zenoss-api.routers.template.TemplateRouter method), 25	list_active_templates()	(zenoss-api.routers.device.ZenossDevice method), 9
get_open_events()	(zenoss-api.routers.events.EventsRouter method), 18	list_all_templates()	(zenoss-api.routers.template.TemplateRouter method), 26
get_open_production_events()	(zenoss-api.routers.events.EventsRouter method), 18	list_bound_templates()	(zenoss-api.routers.device.ZenossDevice method), 9
get_overridable_templates()	(zenoss-api.routers.device.ZenossDevice method), 9	list_collectors()	(zenossapi.routers.device.DeviceRouter method), 5
get_points()	(zenossapi.routers.template.ZenossGraph method), 28	list_components()	(zenoss-api.routers.device.ZenossDevice method), 10
get_properties()	(zenossapi.routers.device.ZenossDevice method), 9	list_custom_properties()	(zenoss-api.routers.device.ZenossDevice method), 10
get_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 15	list_custom_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 16
get_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 22	list_custom_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 22
get_property()	(zenossapi.routers.device.ZenossDevice method), 9	list_data_points()	(zenoss-api.routers.template.ZenossDataSource method), 28
get_property()	(zenoss-api.routers.device.ZenossDeviceClass method), 15	list_data_points()	(zenoss-api.routers.template.ZenossTemplate method), 31
get_property()	(zenoss-api.routers.properties.PropertiesRouter method), 22	list_data_sources()	(zenoss-api.routers.template.ZenossTemplate method), 31
get_router()	(zenossapi.apiclient.Client method), 3	list_device_class_templates()	(zenoss-api.routers.template.TemplateRouter method), 26
get_router_methods()	(zenossapi.apiclient.Client method), 3	list_device_classes()	(zenoss-api.routers.device.DeviceRouter method), 6
get_routers()	(zenossapi.apiclient.Client method), 3	list_devices()	(zenossapi.routers.device.ZenossDeviceClass method), 16
get_template()	(zenoss-api.routers.template.TemplateRouter method), 25	list_graphs()	(zenossapi.routers.template.ZenossTemplate method), 31
get_threshold()	(zenoss-api.routers.template.ZenossTemplate method), 31	list_groups()	(zenossapi.routers.device.DeviceRouter method), 6
get_threshold_types()	(zenoss-api.routers.template.TemplateRouter method), 26		
get_thresholds()	(zenoss-api.routers.template.ZenossTemplate method), 31		
get_tree()	(zenossapi.routers.device.DeviceRouter		

list_jobs()	(zenossapi.routers.jobs.JobsRouter method), 20	api.routers.device.ZenossDevice method), 12
list_local_properties()	(zenoss-api.routers.device.ZenossDevice method), 10	lock_for_updates() (zenoss-api.routers.device.ZenossComponent method), 6
list_local_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 16	lock_for_updates() (zenoss-api.routers.device.ZenossDevice method), 12
list_local_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 23	<b>M</b>
list_local_templates()	(zenoss-api.routers.device.ZenossDevice method), 10	make_counter() (zenoss-api.routers.template.ZenossDataPoint method), 26
list_locations()	(zenossapi.routers.device.DeviceRouter method), 6	make_gauge() (zenossapi.routers.template.ZenossDataPoint method), 27
list_open_events()	(zenoss-api.routers.events.EventsRouter method), 18	move() (zenossapi.routers.device.ZenossDevice method), 12
list_open_production_events()	(zenoss-api.routers.events.EventsRouter method), 18	<b>P</b>
list_overridable_templates()	(zenoss-api.routers.device.ZenossDevice method), 10	PropertiesRouter (class in zenossapi.routers.properties), 21
list_points()	(zenossapi.routers.template.ZenossGraph method), 28	<b>R</b>
list_properties()	(zenossapi.routers.device.ZenossDevice method), 11	reidentify() (zenossapi.routers.device.ZenossDevice method), 12
list_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 16	remodel() (zenossapi.routers.device.ZenossDevice method), 12
list_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 23	reopen() (zenossapi.routers.events.ZenossEvent method), 19
list_systems()	(zenossapi.routers.device.DeviceRouter method), 6	reset_bound_templates() (zenoss-api.routers.device.ZenossDevice method), 12
list_thresholds()	(zenoss-api.routers.template.ZenossTemplate method), 31	reset_ip_address() (zenoss-api.routers.device.ZenossDevice method), 12
list_unbound_templates()	(zenoss-api.routers.device.ZenossDevice method), 11	<b>S</b>
list_user_commands()	(zenoss-api.routers.device.ZenossDevice method), 11	set_bound_templates() (zenoss-api.routers.device.ZenossDevice method), 12
lock()	(zenossapi.routers.device.ZenossComponent method), 6	set_collector() (zenossapi.routers.device.ZenossDevice method), 12
lock()	(zenossapi.routers.device.ZenossDevice method), 11	set_graph_properties() (zenoss-api.routers.template.ZenossGraph method), 28
lock_for_deletion()	(zenoss-api.routers.device.ZenossComponent method), 6	set_max() (zenossapi.routers.template.ZenossThreshold method), 31
lock_for_deletion()	(zenoss-	set_min() (zenossapi.routers.template.ZenossThreshold method), 32
		set_monitored() (zenoss-api.routers.device.ZenossComponent method), 7
		set_point_sequence() (zenoss-api.routers.template.ZenossGraph method), 29

[set\\_priority\(\)](#) (zenossapi.routers.device.ZenossDevice method), [12](#)  
[set\\_production\\_state\(\)](#) (zenossapi.routers.device.ZenossDevice method), [13](#)  
[set\\_properties\(\)](#) (zenossapi.routers.template.TemplateRouter method), [26](#)  
[set\\_property\(\)](#) (zenossapi.routers.device.ZenossDevice method), [13](#)  
[set\\_property\(\)](#) (zenossapi.routers.device.ZenossDeviceClass method), [17](#)  
[set\\_property\\_value\(\)](#) (zenossapi.routers.properties.PropertiesRouter method), [23](#)  
[set\\_threshold\(\)](#) (zenossapi.routers.template.ZenossDataPoint method), [27](#)  
[set\\_value\(\)](#) (zenossapi.routers.properties.ZenossCustomProperty method), [23](#)  
[set\\_value\(\)](#) (zenossapi.routers.properties.ZenossProperty method), [24](#)  
[set\\_zero\\_baseline\(\)](#) (zenossapi.routers.template.ZenossGraph method), [29](#)

## T

[TemplateRouter](#) (class in zenossapi.routers.template), [24](#)

## U

[update\\_config\(\)](#) (zenossapi.routers.events.EventsRouter method), [19](#)  
[update\\_log\(\)](#) (zenossapi.routers.events.ZenossEvent method), [19](#)

## Z

[zenossapi.apiclient](#) (module), [3](#)  
[zenossapi.routers](#) (module), [5](#)  
[zenossapi.routers.device](#) (module), [5](#)  
[zenossapi.routers.events](#) (module), [17](#)  
[zenossapi.routers.jobs](#) (module), [19](#)  
[zenossapi.routers.properties](#) (module), [21](#)  
[zenossapi.routers.template](#) (module), [24](#)  
[ZenossAPIClientAuthenticationError](#), [3](#)  
[ZenossAPIClientError](#), [3](#)  
[ZenossComponent](#) (class in zenossapi.routers.device), [6](#)  
[ZenossCustomProperty](#) (class in zenossapi.routers.properties), [23](#)  
[ZenossDataPoint](#) (class in zenossapi.routers.template), [26](#)  
[ZenossDataSource](#) (class in zenossapi.routers.template), [27](#)  
[ZenossDevice](#) (class in zenossapi.routers.device), [7](#)  
[ZenossDeviceClass](#) (class in zenossapi.routers.device), [13](#)  
[ZenossEvent](#) (class in zenossapi.routers.events), [19](#)