
ZenossAPIClient Documentation

Release 0.1.2

Mark Troyer

Apr 10, 2020

Contents:

1	zenossapi	3
1.1	apiclient Class	3
2	zenossapi Routers	5
2.1	routers Base Class	5
2.2	Device Router	5
2.3	DeviceManagement Router	17
2.4	Events Router	22
2.5	Jobs Router	24
2.6	Monitor Router	26
2.7	Properties Router	27
2.8	Template Router	31
3	Indices and tables	39
	Python Module Index	41
	Index	43

`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')

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'

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', keys=None, 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'
- **keys** (*list*) – Keys to include in the returned data
- **name** (*str*) – Regular expression pattern to filter on, requires the keys parameter

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 DeviceManagement Router

Zenoss devicemanagement_router

class zenossapi.routers.devicemanagement.**DeviceManagementRouter** (*url*, *headers*,
ssl_verify)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with the Zenoss devicemanagement router

add_admin (*uid*, *name*, *role=None*)

Add an admin user to a device or device class.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **name** (*str*) – The name of the user to add
- **role** (*str*) – The role to associate with the user for this device or device class

Returns**Return type** *ZenossDeviceManagementAdmin*

add_maintenance_window (*uid*, *name*, *start*, *duration*, *enabled=False*, *start_state=300*, *repeat='Never'*, *occurrence='1st'*, *days='Sunday'*)

Add a new maintenance window for device or device class.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **start** (*str*) – Window start time in UNIX epoch timestamp format, e.g. “1511290393”
- **duration** (*str*) – Duration of the window in HH:MM:SS format
- **start_state** (*int*) – Production state for the maintenance window, default is 300 (Maintenance)

- **repeat** (*str*) – Maintenance window repeat interval, default is ‘Never’. Other valid choices are: ‘Daily’, ‘Every Weekday’, ‘Weekly’, ‘Monthly: day of month’, ‘Monthly: day of week’
- **occurrence** (*str*) – For ‘Monthly: day of week’ repeats, options are ‘1st’, ‘2nd’, ‘3rd’, ‘4th’, ‘5th’, ‘Last’
- **days** (*str*) – For ‘Monthly: day of week’ repeats, options are ‘Monday’, ‘Tuesday’, ‘Wednesday’, ‘Thursday’, ‘Friday’, ‘Saturday’, ‘Sunday’

Returns

Return type *ZenossMaintenanceWindow*

add_user_command (*uid, name, description, command, password*)

Add a new user command to a device or device class.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **name** (*str*) – Name for the new command
- **description** (*str*) – Description of the new command
- **command** (*str*) – Command line of the new command, can include TALES expressions
- **password** (*str*) – Password of the user adding the command.

get_admin_by_id (*uid, admin_id*)

Get and admin user for a device or device class by id.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **admin_id** (*str*) – The ID of the admin user

Returns

Return type *ZenossDeviceManagementAdmin*

get_admin_by_name (*uid, name*)

Get an admin user for a device or device class by name.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **name** (*str*) – The name of the admin user

Returns

Return type *ZenossDeviceManagementAdmin*

get_admins (*uid*)

Get ZenossDeviceManagementAdmin objects for the configured admin users for a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type *list(ZenossDeviceManagementAdmin)*

get_admins_by_role (*uid, role*)

Get ZenossDeviceManagementAdmin objects for the configured admin users of a device or device class by role.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **role** (*str*) – The role to filter on

Returns

Return type `list(ZenossDeviceManagementAdmin)`

get_maintenance_window (*uid, name*)

Get a maintenance window object for the named window.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **name** (*str*) – Name of the maintenance window

Returns

Return type `ZenossMaintenanceWindow`

get_maintenance_windows (*uid*)

Returns a list of ZenossMaintenanceWindow objects for the maintenance windows configured for a device or device class

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type `list(ZenossMaintenanceWindow)`

get_user_command_by_id (*uid, command_id*)

Get a configured user command by its id

Parameters

- **uid** (*str*) – The UID of the device or device class
- **command_id** (*str*) – The ID of the user command

get_user_command_by_name (*uid, command_name*)

Get a configured user command by its id

Parameters

- **uid** (*str*) – The UID of the device or device class
- **command_name** (*str*) – The name of the user command

get_user_commands (*uid*)

Get a list of user commands objects configured for a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type `list(ZenossUserCommand)`

list_admin_roles (*uid*)

List the admin roles associated with a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type `list(dict)`

list_admins_by_role (*uid*, *role*)

List configured admin users for a device or device class by role.

Parameters

- **uid** (*str*) – The UID of the device or device class
- **role** (*str*) – The role to filter on

Returns

Return type list(dict)

list_available_roles (*uid*)

List the admin roles available to associate with a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type list(str)

list_maintenance_windows (*uid*)

Returns the list of maintenance windows configured for a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type list(dict)

list_user_commands (*uid*)

Get the list of user commands configured for a device or device class.

Parameters **uid** (*str*) – The UID of the device or device class

Returns

Return type list(dict)

list_users (*uid*)

List the users available to associate with a device or device class.

Parameters **uid** (*str*) – the UID of the device or device class

Returns

Return type list(str)

timezone ()

Returns the configured timezone.

Returns

Return type str

```
class zenossapi.routers.devicemanagement.ZenossDeviceManagementAdmin(url,  
                                                                    headers,  
                                                                    ssl_verify,  
                                                                    admin_data)
```

Bases: `zenossapi.routers.devicemanagement.DeviceManagementRouter`

Class for Zenoss user command objects

delete ()

Delete an admin user from a device or device class.

Returns**update** (*role*)

Update the admin user's role.

Parameters *role* (*str*) – New role for the user

Returns

Return type bool

```
class zenossapi.routers.devicemanagement.ZenossMaintenanceWindow(url, headers,
                                                                    ssl_verify,
                                                                    win-
                                                                    dow_data,
                                                                    par-
                                                                    ent=None)
```

Bases: `zenossapi.routers.devicemanagement.DeviceManagementRouter`

Class for Zenoss maintenance window objects

delete ()

Delete a maintenance window from a device or device class.

Returns

Return type dict

disable ()

Set maintenance window to disabled.

Returns

Return type bool

enable ()

Set maintenance window to enabled.

Returns

Return type bool

```
update (start_timestamp=None, start_datetime=None, start_date=None, start_hours=None,
        start_minutes=None, duration_days=None, duration_time=None, duration_hours=None,
        duration_minutes=None, production_state=None, enabled=None, repeat=None, occurrence=None,
        days=None)
```

Update the settings for a maintenance window, with flexible options for specifying the start date/time and duration.

Parameters

- **start_timestamp** (*float*) – Start date and time in UNIX timestamp format
- **start_datetime** (*datetime*) – Start date and time as a `datetime.datetime` object
- **start_date** (*str*) – Start date as a string
- **start_hours** (*str*) – Start hours as a string
- **start_minutes** (*str*) – Start minutes as a string
- **duration_days** (*str*) – Duration days
- **duration_time** (*str*) – Duration time in “HH:MM” format
- **duration_hours** (*str*) – Duration hours

- **duration_minutes** (*str*) – Duration minutes
- **production_state** (*int*) – Production state for the window
- **enabled** (*bool*) – Enabled state of the window
- **occurrence** (*str*) – Repeat occurrence
- **days** (*str*) – Repeat days

Returns**Return type** bool

```
class zenossapi.routers.devicemanagement.ZenossUserCommand(url, headers,
                                                            ssl_verify, com-
                                                            mand_data, par-
                                                            ent=None)
```

Bases: *zenossapi.routers.devicemanagement.DeviceManagementRouter*

Class for Zenoss user command objects

delete ()

Delete a user command for a device or device class.

Returns**Return type** dict**update** (*description=None, command=None, password=None*)

Update a user command.

Parameters

- **description** (*str*) – Description of the user command
- **command** (*str*) – Command line of the command
- **password** (*str*) – Password of the user updating the command.

2.4 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="", event_class='Status',
            **kwargs)
```

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

2.5 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)

2.6 Monitor Router

Zenoss monitor_router

class zenossapi.routers.monitor.**MonitorRouter** (*url, headers, ssl_verify*)

Bases: *zenossapi.routers.ZenossRouter*

Class for interacting with the Zenoss monitor router

get_hub (*name*)

Get a ZenossHub object

Parameters **name** (*str*) – Name of the hub to get

Returns

Return type *ZenossHub*

get_hubs ()

Get the configured hubs as objects

Returns

Return type list(*ZenossHub*)

list_collectors ()

Returns the list of configured collectors

Returns

Return type list(dict)

list_hubs()

Returns the list of configured zenhubs

Returns

Return type list(dict)

tree()

Returns the full tree of hubs and collectors

Returns

Return type list(dict)

class zenossapi.routers.monitor.**ZenossCollector** (*url, headers, ssl_verify, collector_data, collector_params=None*)

Bases: *zenossapi.routers.monitor.MonitorRouter*

Class for Zenoss collector objects

class zenossapi.routers.monitor.**ZenossHub** (*url, headers, ssl_verify, hub_data*)

Bases: *zenossapi.routers.monitor.MonitorRouter*

Class for Zenoss hub objects

add_collector (*name, source=None, pool=None*)

Add a new collector to the hub.

Parameters

- **name** (*str*) – Name of the new collector
- **source** (*str*) – Name of the existing collector to use as a template
- **pool** (*str*) – The resource pool to place the collector in

Returns

Return type *ZenossCollector*

get_collector (*name*)

Get a ZenossCollector object

Parameters **name** (*str*) – Name of the collector to get

Returns

Return type *ZenossCollector*

get_collectors ()

Get the hub's collectors as objects.

Returns

Return type list(*ZenossCollector*)

2.7 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

2.8 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 ()

Delete the graph.

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_graph (*graph*)

Delete a particular graph.

Parameters **graph** (*str*) – The name of the graph to delete.

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, thresh-  
old_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.devicemanagement`, 17
 - `zenossapi.routers.events`, 22
 - `zenossapi.routers.jobs`, 24
 - `zenossapi.routers.monitor`, 26
 - `zenossapi.routers.properties`, 27
 - `zenossapi.routers.template`, 31

A

`abort()` (*zenossapi.routers.jobs.ZenossJob* method), 26
`ack()` (*zenossapi.routers.events.ZenossEvent* method), 24
`add_admin()` (*zenossapi.routers.devicemanagement.DeviceManagementRouter* method), 17
`add_collector()` (*zenossapi.routers.monitor.ZenossHub* method), 27
`add_data_point()` (*zenossapi.routers.template.ZenossDataSource* method), 34
`add_data_point_to_graph()` (*zenossapi.routers.template.TemplateRouter* method), 31
`add_data_source()` (*zenossapi.routers.template.ZenossTemplate* method), 36
`add_device()` (*zenossapi.routers.device.ZenossDeviceClass* method), 13
`add_event()` (*zenossapi.routers.events.EventsRouter* method), 22
`add_graph()` (*zenossapi.routers.template.ZenossTemplate* method), 36
`add_graph_threshold()` (*zenossapi.routers.template.ZenossGraph* method), 34
`add_local_template()` (*zenossapi.routers.device.ZenossDevice* method), 7
`add_local_template()` (*zenossapi.routers.template.TemplateRouter* method), 31
`add_maintenance_window()` (*zenossapi.routers.devicemanagement.DeviceManagementRouter* method), 17
`add_point()` (*zenossapi.routers.template.ZenossGraph* method), 34
`add_subclass()` (*zenossapi.routers.device.ZenossDeviceClass* method), 14
`add_template()` (*zenossapi.routers.template.TemplateRouter* method), 31
`add_threshold()` (*zenossapi.routers.template.ZenossTemplate* method), 36
`add_to_graph()` (*zenossapi.routers.template.ZenossDataPoint* method), 33
`add_user_command()` (*zenossapi.routers.devicemanagement.DeviceManagementRouter* method), 18

B

`bind_or_unbind_template()` (*zenossapi.routers.device.ZenossDevice* method), 7

C

`clear_heartbeat()` (*zenossapi.routers.events.EventsRouter* method), 23
`clear_heartbeats()` (*zenossapi.routers.events.EventsRouter* method), 23
`Client` (class in *zenossapi.apiclient*), 3
`close()` (*zenossapi.routers.events.ZenossEvent* method), 24
`copy()` (*zenossapi.routers.template.ZenossTemplate* method), 36

D

`delete()` (*zenossapi.routers.device.ZenossComponent* method), 17

method), 6

delete() (zenossapi.routers.device.ZenossDevice method), 7

delete() (zenossapi.routers.devicemanagement.ZenossDeviceManagementAdmin method), 20

delete() (zenossapi.routers.devicemanagement.ZenossMaintenanceWindow method), 21

delete() (zenossapi.routers.devicemanagement.ZenossUserCommand method), 22

delete() (zenossapi.routers.jobs.ZenossJob method), 26

delete() (zenossapi.routers.properties.ZenossCustomProperty method), 30

delete() (zenossapi.routers.properties.ZenossProperty method), 30

delete() (zenossapi.routers.template.ZenossDataPoint method), 33

delete() (zenossapi.routers.template.ZenossDataSource method), 34

delete() (zenossapi.routers.template.ZenossGraph method), 35

delete() (zenossapi.routers.template.ZenossTemplate method), 36

delete() (zenossapi.routers.template.ZenossThreshold method), 38

delete_data_point() (zenossapi.routers.template.ZenossDataSource method), 34

delete_data_source() (zenossapi.routers.template.ZenossTemplate method), 36

delete_graph() (zenossapi.routers.template.ZenossTemplate method), 37

delete_local_template() (zenossapi.routers.device.ZenossDevice method), 7

delete_local_template() (zenossapi.routers.template.TemplateRouter method), 31

delete_point() (zenossapi.routers.template.ZenossGraph method), 35

delete_property() (zenossapi.routers.device.ZenossDevice method), 7

delete_property() (zenossapi.routers.device.ZenossDeviceClass method), 14

delete_property() (zenossapi.routers.properties.PropertiesRouter method), 27

delete_template() (zenossapi.routers.template.TemplateRouter method), 31

delete_threshold() (zenossapi.routers.template.ZenossTemplate method), 31

DeviceManagementRouter (class in zenossapi.routers.devicemanagement), 17

DeviceRouter (class in zenossapi.routers.device), 5

DeviceRouter (class in zenossapi.routers.devicemanagement.ZenossMaintenanceWindow), 21

E

enable() (zenossapi.routers.devicemanagement.ZenossMaintenanceWindow method), 21

EventsRouter (class in zenossapi.routers.events), 22

G

get_active_templates() (zenossapi.routers.device.ZenossDevice method), 7

get_admin_by_id() (zenossapi.routers.devicemanagement.DeviceManagementRouter method), 18

get_admin_by_name() (zenossapi.routers.devicemanagement.DeviceManagementRouter method), 18

get_admins() (zenossapi.routers.devicemanagement.DeviceManagementRouter method), 18

get_admins_by_role() (zenossapi.routers.devicemanagement.DeviceManagementRouter method), 18

get_all_templates() (zenossapi.routers.template.TemplateRouter method), 32

get_bound_templates() (zenossapi.routers.device.ZenossDevice method), 8

get_collector() (zenossapi.routers.monitor.ZenossHub method), 27

get_collectors() (zenossapi.routers.monitor.ZenossHub method), 27

get_component() (zenossapi.routers.device.ZenossDevice method), 8

get_components() (zenossapi.routers.device.ZenossDevice method), 8

get_config() (zenossapi.routers.events.EventsRouter method), 23

get_custom_properties() (zenossapi.routers.device.ZenossDevice method), 31

8		37	
get_custom_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 14	get_hub()	(zenossapi.routers.monitor.MonitorRouter method), 26
get_custom_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 28	get_hubs()	(zenoss-api.routers.monitor.MonitorRouter method), 26
get_custom_property()	(zenoss-api.routers.device.ZenossDevice method), 8	get_job()	(zenossapi.routers.jobs.JobsRouter method), 25
get_custom_property()	(zenoss-api.routers.device.ZenossDeviceClass method), 15	get_jobs()	(zenossapi.routers.jobs.JobsRouter method), 25
get_custom_property()	(zenoss-api.routers.properties.PropertiesRouter method), 28	get_local_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 28
get_data_point()	(zenoss-api.routers.template.ZenossDataSource method), 34	get_local_templates()	(zenoss-api.routers.device.ZenossDevice method), 8
get_data_points()	(zenoss-api.routers.template.ZenossDataSource method), 34	get_log()	(zenossapi.routers.jobs.ZenossJob method), 26
get_data_points()	(zenoss-api.routers.template.ZenossTemplate method), 37	get_maintenance_window()	(zenoss-api.routers.devicemanagement.DeviceManagementRouter method), 19
get_data_source()	(zenoss-api.routers.template.ZenossTemplate method), 37	get_maintenance_windows()	(zenoss-api.routers.devicemanagement.DeviceManagementRouter method), 19
get_data_source_types()	(zenoss-api.routers.template.TemplateRouter method), 32	get_object_templates()	(zenoss-api.routers.template.TemplateRouter method), 32
get_data_sources()	(zenoss-api.routers.template.ZenossTemplate method), 37	get_open_events()	(zenoss-api.routers.events.EventsRouter method), 23
get_device()	(zenoss-api.routers.device.ZenossDeviceClass method), 15	get_open_production_events()	(zenoss-api.routers.events.EventsRouter method), 23
get_device_class()	(zenoss-api.routers.device.DeviceRouter method), 5	get_overridable_templates()	(zenoss-api.routers.device.ZenossDevice method), 9
get_device_class_templates()	(zenoss-api.routers.template.TemplateRouter method), 32	get_points()	(zenoss-api.routers.template.ZenossGraph method), 35
get_devices()	(zenoss-api.routers.device.ZenossDeviceClass method), 15	get_properties()	(zenoss-api.routers.device.ZenossDevice method), 9
get_event_by_evid()	(zenoss-api.routers.events.EventsRouter method), 23	get_properties()	(zenoss-api.routers.device.ZenossDeviceClass method), 15
get_graph()	(zenoss-api.routers.template.ZenossTemplate method), 37	get_properties()	(zenoss-api.routers.properties.PropertiesRouter method), 28
get_graphs()	(zenoss-api.routers.template.ZenossTemplate method), 37	get_property()	(zenoss-api.routers.device.ZenossDevice method), 9
		get_property()	(zenoss-api.routers.device.ZenossDeviceClass method), 15

<code>get_property()</code> (zenoss- <i>api.routers.properties.PropertiesRouter</i> method), 29	<code>list_bound_templates()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> 9	(zenoss- method),
<code>get_router()</code> (zenossapi.apiclient.Client method), 3	<code>list_collectors()</code> (zenoss- <i>api.routers.device.DeviceRouter</i> 5	(zenoss- method),
<code>get_router_methods()</code> (zenossapi.apiclient.Client method), 3	<code>list_collectors()</code> (zenoss- <i>api.routers.monitor.MonitorRouter</i> 26	(zenoss- method),
<code>get_routers()</code> (zenossapi.apiclient.Client method), 3	<code>list_components()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> 9	(zenoss- method),
<code>get_template()</code> (zenoss- <i>api.routers.template.TemplateRouter</i> method), 32	<code>list_custom_properties()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> 10	(zenoss- method),
<code>get_threshold()</code> (zenoss- <i>api.routers.template.ZenossTemplate</i> method), 37	<code>list_custom_properties()</code> (zenoss- <i>api.routers.device.ZenossDeviceClass</i> method), 16	(zenoss- method),
<code>get_threshold_types()</code> (zenoss- <i>api.routers.template.TemplateRouter</i> method), 32	<code>list_custom_properties()</code> (zenoss- <i>api.routers.properties.PropertiesRouter</i> method), 29	(zenoss- method),
<code>get_thresholds()</code> (zenoss- <i>api.routers.template.ZenossTemplate</i> method), 37	<code>list_data_points()</code> (zenoss- <i>api.routers.template.ZenossDataSource</i> method), 34	(zenoss- method),
<code>get_tree()</code> (zenossapi.routers.device.DeviceRouter method), 5	<code>list_data_points()</code> (zenoss- <i>api.routers.template.ZenossTemplate</i> method), 38	(zenoss- method),
<code>get_unbound_templates()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> method), 9	<code>list_data_sources()</code> (zenoss- <i>api.routers.template.ZenossTemplate</i> method), 38	(zenoss- method),
<code>get_user_command_by_id()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 19	<code>list_device_class_templates()</code> (zenoss- <i>api.routers.template.TemplateRouter</i> method), 32	(zenoss- method),
<code>get_user_command_by_name()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 19	<code>list_device_classes()</code> (zenoss- <i>api.routers.device.DeviceRouter</i> method), 6	(zenoss- method),
<code>get_user_commands()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 19	<code>list_devices()</code> (zenoss- <i>api.routers.device.ZenossDeviceClass</i> method), 16	(zenoss- method),
J	<code>list_graphs()</code> (zenoss- <i>api.routers.template.ZenossTemplate</i> method), 38	(zenoss- method),
<code>JobsRouter</code> (class in zenossapi.routers.jobs), 24	<code>list_groups()</code> (zenoss- <i>api.routers.device.DeviceRouter</i> method), 6	(zenoss- method),
L	<code>list_hubs()</code> (zenoss- <i>api.routers.monitor.MonitorRouter</i> method), 26	(zenoss- method),
<code>list_active_templates()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> method), 9	<code>list_jobs()</code> (zenossapi.routers.jobs.JobsRouter method), 25	(zenoss- method),
<code>list_admin_roles()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 19	<code>list_local_properties()</code> (zenoss- <i>api.routers.device.ZenossDevice</i> method), 10	(zenoss- method),
<code>list_admins_by_role()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 19	<code>list_local_properties()</code> (zenoss-	
<code>list_all_templates()</code> (zenoss- <i>api.routers.template.TemplateRouter</i> method), 32		
<code>list_available_roles()</code> (zenoss- <i>api.routers.devicemanagement.DeviceManagementRouter</i> method), 20		

<i>api.routers.device.ZenossDeviceClass</i> method), 16	<i>method</i>), 6
<i>list_local_properties()</i> (<i>zenoss-api.routers.properties.PropertiesRouter</i> method), 29	<i>lock()</i> (<i>zenossapi.routers.device.ZenossDevice</i> method), 11
<i>list_local_templates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 10	<i>lock_for_deletion()</i> (<i>zenoss-api.routers.device.ZenossComponent</i> method), 6
<i>list_locations()</i> (<i>zenoss-api.routers.device.DeviceRouter</i> method), 6	<i>lock_for_deletion()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
<i>list_maintenance_windows()</i> (<i>zenoss-api.routers.devicemanagement.DeviceManagementRouter</i> method), 20	<i>lock_for_updates()</i> (<i>zenoss-api.routers.device.ZenossComponent</i> method), 6
<i>list_open_events()</i> (<i>zenoss-api.routers.events.EventsRouter</i> method), 24	<i>lock_for_updates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
<i>list_open_production_events()</i> (<i>zenoss-api.routers.events.EventsRouter</i> method), 24	M
<i>list_overridable_templates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 10	<i>make_counter()</i> (<i>zenoss-api.routers.template.ZenossDataPoint</i> method), 33
<i>list_points()</i> (<i>zenoss-api.routers.template.ZenossGraph</i> method), 35	<i>make_gauge()</i> (<i>zenoss-api.routers.template.ZenossDataPoint</i> method), 33
<i>list_properties()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 11	<i>MonitorRouter</i> (class in <i>zenossapi.routers.monitor</i>), 26
<i>list_properties()</i> (<i>zenoss-api.routers.device.ZenossDeviceClass</i> method), 16	<i>move()</i> (<i>zenossapi.routers.device.ZenossDevice</i> method), 12
<i>list_properties()</i> (<i>zenoss-api.routers.properties.PropertiesRouter</i> method), 29	P
<i>list_systems()</i> (<i>zenoss-api.routers.device.DeviceRouter</i> method), 6	<i>PropertiesRouter</i> (class in <i>zenoss-api.routers.properties</i>), 27
<i>list_thresholds()</i> (<i>zenoss-api.routers.template.ZenossTemplate</i> method), 38	R
<i>list_unbound_templates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 11	<i>reidentify()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
<i>list_user_commands()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 11	<i>remodel()</i> (<i>zenossapi.routers.device.ZenossDevice</i> method), 12
<i>list_user_commands()</i> (<i>zenoss-api.routers.devicemanagement.DeviceManagementRouter</i> method), 20	<i>reopen()</i> (<i>zenossapi.routers.events.ZenossEvent</i> method), 24
<i>list_users()</i> (<i>zenoss-api.routers.devicemanagement.DeviceManagementRouter</i> method), 20	<i>reset_bound_templates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
<i>lock()</i> (<i>zenossapi.routers.device.ZenossComponent</i> method), 6	<i>reset_ip_address()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
	S
	<i>set_bound_templates()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12
	<i>set_collector()</i> (<i>zenoss-api.routers.device.ZenossDevice</i> method), 12

`set_graph_properties()` (*zenoss-api.routers.template.ZenossGraph method*), 35
`set_max()` (*zenossapi.routers.template.ZenossThreshold method*), 38
`set_min()` (*zenossapi.routers.template.ZenossThreshold method*), 38
`set_monitored()` (*zenoss-api.routers.device.ZenossComponent method*), 7
`set_point_sequence()` (*zenoss-api.routers.template.ZenossGraph method*), 35
`set_priority()` (*zenoss-api.routers.device.ZenossDevice method*), 12
`set_production_state()` (*zenoss-api.routers.device.ZenossDevice method*), 13
`set_properties()` (*zenoss-api.routers.template.TemplateRouter method*), 33
`set_property()` (*zenoss-api.routers.device.ZenossDevice method*), 13
`set_property()` (*zenoss-api.routers.device.ZenossDeviceClass method*), 17
`set_property_value()` (*zenoss-api.routers.properties.PropertiesRouter method*), 30
`set_threshold()` (*zenoss-api.routers.template.ZenossDataPoint method*), 33
`set_value()` (*zenoss-api.routers.properties.ZenossCustomProperty method*), 30
`set_value()` (*zenoss-api.routers.properties.ZenossProperty method*), 30
`set_zero_baseline()` (*zenoss-api.routers.template.ZenossGraph method*), 35

T

`TemplateRouter` (*class in zenoss-api.routers.template*), 31
`timezone()` (*zenoss-api.routers.devicemanagement.DeviceManagementRouter method*), 20
`tree()` (*zenossapi.routers.monitor.MonitorRouter method*), 27

U

`update()` (*zenossapi.routers.devicemanagement.ZenossDeviceManagement method*), 21
`update()` (*zenossapi.routers.devicemanagement.ZenossMaintenanceWindow method*), 21
`update()` (*zenossapi.routers.devicemanagement.ZenossUserCommand method*), 22
`update_config()` (*zenoss-api.routers.events.EventsRouter method*), 24
`update_log()` (*zenossapi.routers.events.ZenossEvent method*), 24

Z

`zenossapi.apiclient` (*module*), 3
`zenossapi.routers` (*module*), 5
`zenossapi.routers.device` (*module*), 5
`zenossapi.routers.devicemanagement` (*module*), 17
`zenossapi.routers.events` (*module*), 22
`zenossapi.routers.jobs` (*module*), 24
`zenossapi.routers.monitor` (*module*), 26
`zenossapi.routers.properties` (*module*), 27
`zenossapi.routers.template` (*module*), 31
`ZenossAPIClientAuthenticationError`, 3
`ZenossAPIClientError`, 3
`ZenossCollector` (*class in zenoss-api.routers.monitor*), 27
`ZenossComponent` (*class in zenoss-api.routers.device*), 6
`ZenossCustomProperty` (*class in zenoss-api.routers.properties*), 30
`ZenossDataPoint` (*class in zenoss-api.routers.template*), 33
`ZenossDataSource` (*class in zenoss-api.routers.template*), 33
`ZenossDevice` (*class in zenossapi.routers.device*), 7
`ZenossDeviceClass` (*class in zenoss-api.routers.device*), 13
`ZenossDeviceManagementAdmin` (*class in zenoss-api.routers.devicemanagement*), 20
`ZenossEvent` (*class in zenossapi.routers.events*), 24
`ZenossGraph` (*class in zenossapi.routers.template*), 34
`ZenossHub` (*class in zenossapi.routers.monitor*), 27
`ZenossJob` (*class in zenossapi.routers.jobs*), 25
`ZenossMaintenanceWindow` (*class in zenoss-api.routers.devicemanagement*), 21
`ZenossProperty` (*class in zenoss-api.routers.properties*), 30
`ZenossRouter` (*class in zenossapi.routers*), 5
`ZenossTemplate` (*class in zenoss-api.routers.template*), 35
`ZenossThreshold` (*class in zenoss-api.routers.template*), 38

ZenossUserCommand (class in *zenoss-api.routers.devicemanagement*), [22](#)