Special objects for scripts

Overview

Special objects are available within processing scripts to provide enhanced management capabilities for **Devices** and their **Settings**. These objects can be referenced directly in your scripts to perform various operations and modify device attributes.

The `device` Object

The script execution context includes a special `device` object that allows you to modify attributes of the managed **Device** and perform related actions and operations.

Attributes

The `device` object provides the following attributes:

Name	Type/Access	Description
`obj_id`	integer (read-only)	Numeric unique identifier of the current device
`name`	string (read-write)	Device name field
`customer_id`	string (read-write)	Customer ID field of the current device
`type_id`	integer (read-only)	Numeric identifier of the device's profile
`organization_id`	integer (read-only)	Numeric identifier of the device's organization
`serial_number`	string (read-write)	Serial number field of the current device
`serial_number_alt`	string (read-write)	Alternate serial number field of the current device
`status`	boolean (read-only)	Operational status of the device

Name	Type/Access	Description
`status_change`	datetime (read-only)	Date/time of the last status change
`last_connection`	datetime (read-only)	Date/time of the last connection from the device to the platform
`update_frequency`	integer (read-write)	Device's update frequency or periodic report interval (in seconds). Changes to this attribute will trigger a connection request
`software_version`	string (read-only)	Software version information as reported by the device
`hardware_version`	string (read-only)	Hardware version information as reported by the device
`address`	string (read-only)	Management address used by the device to connect to the platform
`description`	string (read-write)	Description field of the current device
`description2`	string (read-write)	Second description field of the current device
`description3`	string (read-write)	Third description field of the current device
`description4`	string (read-write)	Fourth description field of the current device
`reboot`	boolean (read-write)	Activate the 'Reboot' action on the device or detect if a 'Reboot' action is pending
`factory`	boolean (read-write)	Activate the 'Factory reset' action on the device or detect if a 'Factory reset' action is pending
`device_factory`	boolean (read-write)	Activate the 'Remote (device only) factory reset' action or detect if the action is pending
`factory_remote`	boolean (read-write)	Alias for `device_factory`. Activate the 'Remote (device only) factory reset' action or detect if the action is pending
`sync`	boolean (read-write)	Activate the configuration 'Synchronization' action with the device or detect if the action is pending
`reconf`	boolean (read-write)	Activate the device 'Reconfiguration' action or detect if the action is pending

Name	Type/Access	Description
`location_id`	integer (read-write)	Normalized location ID assigned to the device
`location_name`	string (read-write)	Non-normalized location name assigned to the device
`latitude`	float (read-write)	Latitude geographical coordinate of the device
`longitude`	float (read-write)	Longitude geographical coordinate of the device
`firmware_image_id`	integer (read-write)	Numeric identifier of the firmware image that can be applied to the device
`firmware_image_is_pending`	boolean (read-write)	Indicates whether the configured firmware image will be applied to the managed device

Methods

The `device` object provides the following methods:

`save()`

Save changes made to the device's read-write attributes.

Returns:

- `status` (boolean)
- `message` (string)

Example:

```
status, message = device.save()
```

`set()`

Change the value of a specified **Parameter** by its `parameter_name`, `variable_name`, or `parameter_id`.

Parameters:

- `parameter_name` (string) – Parameter name identifier
- `variable_name` (string) – Variable name identifier
- `parameter_id` (integer) – Numeric parameter identifier
- `value` (string) – New value to set

Returns:

- `status` (boolean) – Operation status
- `message` (string) – Operation message
- `created` (boolean) – Whether a new parameter was created
- `changed` (boolean) – Whether the value was changed
- `old_value` (string) – Previous value

Examples:

```
status, message, created, changed, old_value = device.set(parameter_id=100, value='test')


device.set(parameter_name='Device Name', value='test')


device.set(variable_name='Device.SystemID', value='test')
```

`get()`

Retrieve the value of a specified **Parameter** by its `parameter_name`, `variable_name`, or `parameter_id` .

Parameters:

- `parameter_name` (string) – Parameter name identifier
- `variable_name` (string) – Variable name identifier
- `parameter_id` (integer) – Numeric parameter identifier

Returns:

- `value` (string) – Parameter value
- `found` (boolean) – Whether the parameter was found

Examples:

```
value, found = device.get(parameter_id=100)


value, _ = device.get(parameter_name="Uptime")


value, _ = device.get(variable_name='Device.SystemID')
```

`delete()`

Delete a setting from the device database specified by its parameter's `parameter_name`, `variable_name`, or `parameter_id`.

Parameters:

- `parameter_name` (string) – Parameter name identifier
- `variable_name` (string) – Variable name identifier
- `parameter_id` (integer) – Numeric parameter identifier

Returns:

- `deleted` (boolean) – Whether the parameter was deleted
- `message` (string) – Operation message

Examples:

```
deleted, message = device.delete(parameter_id=100)


device.delete(parameter_name='Uptime')


device.delete(variable_name='Device.SystemID')
```

`cwmp_set()`

Perform a CWMP SetParameterValues operation on devices managed by CWMP/TR-069.

Parameters:

- `variable_name` (string, required) – Variable name to set
- `value` (string, required) – Value to assign
- `variable_type` (string, optional) – Type of parameter (default: `'string'`). Options: `'string'`, `'boolean'`, `'integer'`, `'positive'`
- `log` (boolean, optional) – Enable operation logging (default: `False`)
- `disable_connreq` (boolean, optional) – Disable CWMP connection request after the operation (default: `False`)

Examples:

```
device.cwmp_set(variable_name='Device.SystemID', value='test')


device.cwmp_set(variable_name='Device.SystemID', value='test', variable_type='string',
log=True, disable_connreq=True)
```

`connection_request()`

Perform a connection request to the managed device.

Parameters:

- `wait` (boolean, optional) – Wait for connection request response or timeout (default: `False`)
- `sleep_time` (float, optional) – Wait time between operations in seconds (default: `0.05`)
- `type` (string, optional) – Connection request type: `'regular'` or `'light'` (default: `'regular'`)

Example:

```
device.connection_request(wait=False, type='light')
```

`alert_clear()`

Clear any active alert on the managed device.

Example:

```
device.alert_clear()
```

`ping()`

Perform an ICMP ping from the platform to the device's management address.

Parameters:

- `count` (integer, optional) – Number of packets to send (default: `2`, maximum: `10`)
- `timeout` (integer, optional) – Timeout in seconds (default: `1`, maximum: `10`)

Returns:

- `status` (boolean) – Ping operation status
- `message` (string) – Operation message
- `packet_loss` (integer) – Packet loss percentage

Example:

```
status, message, packet_loss = device.ping()
```

`get_or_create()`

Get or create a device with the specified parameters. Commonly used with the `csv_row` variable.

Parameters:

- `name` (string, optional) – Device name
- `serial_number` (string, optional) – Serial number
- `serial_number_alt` (string, optional) – Alternate serial number
- `username` (string, optional) – Username
- `location_id` (integer, optional) – Location ID

Returns:

- `device_obj` (device object) – Device object on which you can perform operations

Examples:

```
device_obj = device.get_or_create(name=csv_row[0])


device_obj = device.get_or_create(name="device_name", username="username")
```

`get_obj()`

Retrieve a device with the specified parameters. Commonly used with the `csv_row` variable.

Parameters:

- `name` (string, optional) – Device name
- `serial_number` (string, optional) – Serial number
- `serial_number_alt` (string, optional) – Alternate serial number
- `username` (string, optional) – Username
- `location_id` (integer, optional) – Location ID
- `organization_id` (integer, optional) – Organization ID

Returns:

- `found` (boolean) – Whether the device was found

Note: When a device is found, the `device` variable is converted to the corresponding device object.

Examples:

```
found = device.get_obj(name=csv_row[0])


found = device.get_obj(name="device_name", username="username")
```

`delete_obj()`

Delete the current device object.

Returns:

- `deleted` (boolean) – Whether the device was deleted

Example:

```
deleted = device.delete_obj()
```

`update_location()`

Update or clear the device location.

Parameters:

- `id` (integer, optional) – Location ID
- `name` (string, optional) – Location name
- `short_name` (string, optional) – Location short name

Returns:

- `updated` (boolean) – Whether the location was updated
- `message` (string) – Operation message

Examples:

```
updated, message = device.update_location(name="Location name")


updated, message = device.update_location(short_name="loc-short-name")


# Clear the current location
updated, message = device.update_location()
```

`group_list()`

List the groups the device belongs to.

Parameters:

- `short_name` (boolean, optional) – Return short names instead of IDs (default: `False`)

Returns:

- `group_list` (list) – List of group IDs or short names

Examples:

```
group_list = device.group_list()


group_list = device.group_list(short_name=True)
```

`group_add()`

Add the device to a specified group.

Parameters:

- `id` (integer, optional) – Group ID
- `short_name` (string, optional) – Group short name

Returns:

- `updated` (boolean) – Whether the device was added to the group
- `message` (string) – Operation message

Examples:

```
updated, message = device.group_add(id=45)


updated, message = device.group_add(short_name="group-short-name")
```

`group_remove()`

Remove the device from a specified group.

Parameters:

- `id` (integer, optional) – Group ID
- `short_name` (string, optional) – Group short name

Returns:

- `updated` (boolean) – Whether the device was removed from the group
- `message` (string) – Operation message

Examples:

```
updated, message = device.group_remove(id=45)


updated, message = device.group_remove(short_name="group-short-name")
```

Script Examples

Example 1: Update device description and set parameter

```
result = value
device.description = value
device.save()
status, message, created, changed, old_value = device.set(variable_name='Device.SystemID',
value=value)
if changed:
    device.connection_request()
```

Example 2: Update alternate serial number based on variable name

```
result = value
if variable_name == 'Device.X_LTE_INFO.imei':
    device.serial_alt = value
    device.save()
```

Example 3: Build description from multiple parameters

```
result = value
if variable_name == 'Device.SystemInfo.Manufacturer':
    imsi = device.get(variable_name='Device.X_LTE_INFO.imsi')
    if imsi:
        device.description2 = '%s - imsi: %s' % (value, imsi)
    else:
        device.description2 = value
    device.save()
```

Example 4: Conditional connection request based on ping results

```
if device.status is False:
    ping_status, ping_message, ping_packet_loss = device.ping()
```

```
    if ping_status is True and ping_packet_loss < 50:
        # Send connection request if the device is reachable and packet loss is less than 50%
        device.connection_request()
```

The `csv_row` Object

When a script is executed over a CSV file, the code defined in the **script** field is applied to every row of the CSV file. The `csv_row` object (array) is available within the script context, containing the values from the current row.

CSV Processing Example

The following example demonstrates updating device coordinates from a CSV file with the format: `[device_name, latitude, longitude]`

```
# Updating device coordinates with info from CSV file
# CSV format: [device_name, latitude, longitude]

# Look up the device using the serial number (or "name")
found = device.get_obj(serial_number=csv_row[0])
latitude = csv_row[1]
longitude = csv_row[2]

if found:
    # Set latitude and longitude
    device.latitude = latitude
    device.longitude = longitude
    # Save changes
    device.save()
    # Show success message
    out = "Updated the coordinates on Device: %s" % csv_row[0]
else:
    # Device not found
    out = "Not found serial_number: %s" % csv_row[0]
```

Revision #2
Created 2026-02-13 22:29:23 UTC by ipena@zequenze.com
Updated 2026-04-09 03:13:15 UTC by mauro@zequenze.com