meshctrl Documentation

Release 1.3.2

Josiah Baldwin

Oct 23, 2025

CONTENTS

This is the documentation of **meshctrl**.

CHAPTER

ONE

CONTENTS

1.1 meshctrl

Library for remotely interacting with a MeshCentral server instance

1.1.1 Installation

pip install libmeshctrl

1.1.2 Usage

This module is implemented as a primarily asynchronous library (asyncio), mostly through the Session class. Because the library is asynchronous, you must wait for it to be initialized before interacting with the server. The preferred way to do this is to use the async context manager pattern:

```
import meshctrl

async with meshctrl.Session(url, **options):
    print(await session.list_users())
    ...
```

However, if you prefer to instantiate the object yourself, you can simply use the initialized property:

```
session = meshctrl.Session(url, **options)
await session.initialized.wait()
```

Note that, in this case, you will be rquired to clean up tho session using its close method.

1.1.3 Session Parameters

`url`: URL of meshcentral server to connect to. Should start with either "ws://" or "wss://".

`options`: optional parameters. Described at Read the Docs

1.1.4 API

API is documented in the API Docs

1.2 Note

This project has been set up using PyScaffold 4.6. For details and usage information on PyScaffold see https://pyscaffold.org/.

1.3 Contributing

Welcome to `meshctrl` contributor's guide.

This document focuses on getting any potential contributor familiarized with the development processes, but other kinds of contributions are also appreciated.

If you are new to using git or have never collaborated in a project previously, please have a look at contribution-guide.org. Other resources are also listed in the excellent guide created by FreeCodeCamp.

Please notice, all users and contributors are expected to be **open, considerate, reasonable, and respectful**. When in doubt, Python Software Foundation's Code of Conduct is a good reference in terms of behavior guidelines.

1.3.1 Issue Reports

If you experience bugs or general issues with `meshctrl`, please have a look on the issue tracker. If you don't see anything useful there, please feel free to fire an issue report.

> **Tip**
>
> Please don't forget to include the closed issues in your search. Sometimes a solution was already reported, and the problem is considered **solved**.

New issue reports should include information about your programming environment (e.g., operating system, Python version) and steps to reproduce the problem. Please try also to simplify the reproduction steps to a very minimal example that still illustrates the problem you are facing. By removing other factors, you help us to identify the root cause of the issue.

1.3.2 Documentation Improvements

You can help improve `meshctrl` docs by making them more readable and coherent, or by adding missing information and correcting mistakes.

`meshctrl` documentation uses Sphinx as its main documentation compiler. This means that the docs are kept in the same repository as the project code, and that any documentation update is done in the same way was a code contribution.

> **Tip**
>
> Please notice that the GitHub web interface provides a quick way of propose changes in `meshctrl`'s files. While this mechanism can be tricky for normal code contributions, it works perfectly fine for contributing to the docs, and can be quite handy.
>
> If you are interested in trying this method out, please navigate to the `docs` folder in the source repository, find which file you would like to propose changes and click in the little pencil icon at the top, to open GitHub's code editor. Once you finish editing the file, please write a message in the form at

> the bottom of the page describing which changes have you made and what are the motivations behind them and submit your proposal.

When working on documentation changes in your local machine, you can compile them using `tox`:

```
tox -e docs
```

and use Python's built-in web server for a preview in your web browser (`http://localhost:8000`):

```
python3 -m http.server --directory 'docs/_build/html'
```

Submit an issue

Before you work on any non-trivial code contribution it's best to first create a report in the issue tracker to start a discussion on the subject. This often provides additional considerations and avoids unnecessary work.

Create an environment

Before you start coding, we recommend creating an isolated virtual environment to avoid any problems with your installed Python packages. This can easily be done via `virtualenv`:

```
python -m venv <PATH TO VENV>
source <PATH TO VENV>/bin/activate
```

Clone the repository

1. Create an user account on GitHub if you do not already have one.
2. Fork the project repository: click on the *Fork* button near the top of the page. This creates a copy of the code under your account on GitHub.
3. Clone this copy to your local disk:

   ```
   git clone git@github.com:YourLogin/meshctrl.git
   cd meshctrl
   ```

4. You should run:

   ```
   pip install -U pip setuptools -e .
   ```

   to be able to import the package under development in the Python REPL.

Implement your changes

1. Create a branch to hold your changes:

   ```
   git checkout -b my-feature
   ```

   and start making changes. Never work on the main branch!

2. Start your work on this branch. Don't forget to add docstrings to new functions, modules and classes, especially if they are part of public APIs.
3. Add yourself to the list of contributors in `AUTHORS.rst`.
4. When you're done editing, do:

```
git add <MODIFIED FILES>
git commit
```

to record your changes in git.

Submit your contribution

1. If everything works fine, push your local branch to GitHub with:

```
git push -u origin my-feature
```

2. Go to the web page of your fork and click "Create pull request" to send your changes for review.

   Find more detailed information in creating a PR. You might also want to open the PR as a draft first and mark it as ready for review after the feedbacks from the continuous integration (CI) system or any required fixes.

Troubleshooting

The following tips can be used when facing problems to build or test the package:

1. Make sure to fetch all the tags from the upstream repository. The command `git describe --abbrev=0 --tags` should return the version you are expecting. If you are trying to run CI scripts in a fork repository, make sure to push all the tags. You can also try to remove all the egg files or the complete egg folder, i.e., `.eggs`, as well as the `*.egg-info` folders in the `src` folder or potentially in the root of your project.

2. Sometimes `tox` misses out when new dependencies are added, especially to `setup.cfg` and `docs/requirements.txt`. If you find any problems with missing dependencies when running a command with `tox`, try to recreate the `tox` environment using the `-r` flag. For example, instead of:

```
tox -e docs
```

   Try running:

```
tox -r -e docs
```

3. Make sure to have a reliable `tox` installation that uses the correct Python version (e.g., 3.7+). When in doubt you can run:

```
tox --version
# OR
which tox
```

   If you have trouble and are seeing weird errors upon running `tox`, you can also try to create a dedicated virtual environment with a `tox` binary freshly installed. For example:

```
virtualenv .venv
source .venv/bin/activate
.venv/bin/pip install tox
.venv/bin/tox -e all
```

4. Pytest can drop you in an interactive session in the case an error occurs. In order to do that you need to pass a `--pdb` option (for example by running `tox -- -k <NAME OF THE FALLING TEST> --pdb`). You can also setup breakpoints manually instead of using the `--pdb` option.

1.4 License

1.5 Contributors

- Josiah Baldwin <jbaldwin8889@gmail.com>
- Daan Selen <https://github.com/DaanSelen>

1.6 Changelog

1.6.1 version 1.3.2

Improvments:

- Fix race condition that could occur when running *run_command* or *run_console_command*

1.6.2 version 1.3.1

Improvments:

- Basically just everything in 1.3.0, this is a release fix

1.6.3 version 1.3.0

Improvements:

- Improved how run_commands was handled (#51)
- Added remove device functionality (#52)
- Added run_console_commands functionality (#55)

Bugs:

- Silly documentation being wrong (#53)

1.6.4 version 1.2.2

Improvements:

- Added user agent to websocket headers

Bugs:

- Fixed library's __version__ implementation
- Fixed data from certain devices not showing up due to overloading websocket packet sizes

1.6.5 version 1.2.1

Bugs:

- Fixed handling of meshcentral's list_devices return with details=True

1.6.6 version 1.2.0

Bugs:

- Fixed agent sometimes being None causing an oxception
- Fixed bad code in device_open_url

Features:

- Changed websockets version to 15. This now uses the proxy implemention from that library, instead of the previous hack.
- Added lastaddr and lastconnect to list_devices API

1.6.7 version 1.1.2

Bugs:

- Fixed semver for requirements. New version of websockets broke this library.

Security:

- Updated cryptogaphy to ~44.0.1 to fix ssl vulnerability.

1.6.8 Version 1.1.1

Bugs:

- Fixed bug when running device_info when user has access to multiple meshes

1.6.9 Version 1.1.0

Features:

- Added overrides for meshcentral files for testing purposes
- Added *users* field to *device* object

Bugs:

- Fixed connection errors not raising immediately
- Fixed run_commands parsing return from multiple devices incorrectly
- Fixed listening to raw not removing its listener correctly

- Fixed javascript timecodes not being handled in gnu environments
- Changed some fstring formatting that locked the library into python >3.13

1.6.10 Version 1.0.0

First release

1.7 meshctrl

1.7.1 meshctrl package

Submodules

meshctrl.constants module

flag meshctrl.constants.**AgentCapabilities**(*value*)

Bases: IntFlag

Flags of capabilities an agent can have. Taken from meshagent.h MeshCommand_AuthInfo_CapabilitiesMask from meshagent repo

Member Type
int

Valid values are as follows:

DESKTOP = <AgentCapabilities.DESKTOP: 1>

TERMINAL = <AgentCapabilities.TERMINAL: 2>

FILES = <AgentCapabilities.FILES: 4>

CONSOLE = <AgentCapabilities.CONSOLE: 8>

JAVASCRIPT = <AgentCapabilities.JAVASCRIPT: 16>

TEMPORARY = <AgentCapabilities.TEMPORARY: 32>

RECOVERY = <AgentCapabilities.RECOVERY: 64>

RESERVED = <AgentCapabilities.RESERVED: 128>

COMPRESSION = <AgentCapabilities.COMPRESSION: 256>

enum meshctrl.constants.**AgentType**(*value*)

Bases: IntEnum

Which type of agent this is. Taken from meshcentral.js obj.meshAgentsArchitectureNumbers

Member Type
int

Valid values are as follows:

UNKNOWN = <AgentType.UNKNOWN: 0>

CONSOLE_WIN_X86_32 = <AgentType.CONSOLE_WIN_X86_32: 1>

CONSOLE_WIN_X86_64 = <AgentType.CONSOLE_WIN_X86_64: 2>

```
SERVICE_WIN_X86_32 = <AgentType.SERVICE_WIN_X86_32:  3>

SERVICE_WIN_X86_64 = <AgentType.SERVICE_WIN_X86_64:  4>

SERVICE_LINUX_X86_32 = <AgentType.SERVICE_LINUX_X86_32:  5>

SERVICE_LINUX_X86_64 = <AgentType.SERVICE_LINUX_X86_64:  6>

SERVICE_LINUX_MIPS = <AgentType.SERVICE_LINUX_MIPS: 7>

SERVICE_LINUX_XEN_X86_32 = <AgentType.SERVICE_LINUX_XEN_X86_32:  8>

SERVICE_LINUX_ARM5 = <AgentType.SERVICE_LINUX_ARM5:  9>

SERVICE_LINUX_ARM_PLUGPC = <AgentType.SERVICE_LINUX_ARM_PLUGPC: 10>

SERVICE_MACOS_X86_32 = <AgentType.SERVICE_MACOS_X86_32:  11>

SERVICE_ANDROID_X86_32 = <AgentType.SERVICE_ANDROID_X86_32:  12>

SERVICE_ANDROID_POGOPLUG = <AgentType.SERVICE_ANDROID_POGOPLUG: 13>

SERVICE_ANDROID_APK = <AgentType.SERVICE_ANDROID_APK: 14>

SERVICE_LINUX_POKY_x86_32 = <AgentType.SERVICE_LINUX_POKY_x86_32:  15>

SERVICE_MACOS_X86_64 = <AgentType.SERVICE_MACOS_X86_64:  16>

SERVICE_CHROMEOS = <AgentType.SERVICE_CHROMEOS: 17>

SERVICE_LINUX_POKY_x86_64 = <AgentType.SERVICE_LINUX_POKY_x86_64:  18>

SERVICE_LINUX_X86_32_NOKVM = <AgentType.SERVICE_LINUX_X86_32_NOKVM: 19>

SERVICE_LINUX_X86_64_NOKVM = <AgentType.SERVICE_LINUX_X86_64_NOKVM: 20>

CONSOLE_WIN_MINICORE_X86_32 = <AgentType.CONSOLE_WIN_MINICORE_X86_32:  21>

SERVICE_WIN_MINICORE_X86_32 = <AgentType.SERVICE_WIN_MINICORE_X86_32:  22>

SERVICE_NODEJS = <AgentType.SERVICE_NODEJS: 23>

SERVICE_LINUX_ARM_LINARO = <AgentType.SERVICE_LINUX_ARM_LINARO: 24>

SERVICE_LINUX_ARM_HARDFLOAT = <AgentType.SERVICE_LINUX_ARM_HARDFLOAT: 25>

SERVICE_LINUX_ARM64 = <AgentType.SERVICE_LINUX_ARM64:  26>

SERVICE_LINUX_ARM_HARDFLOAT_2 = <AgentType.SERVICE_LINUX_ARM_HARDFLOAT_2:  27>

SERVICE_LINUX_MIPS24KC = <AgentType.SERVICE_LINUX_MIPS24KC: 28>

SERVICE_MACOS_ARM64 = <AgentType.SERVICE_MACOS_ARM64:  29>

SERVICE_FREEBSD_X86_64 = <AgentType.SERVICE_FREEBSD_X86_64:  30>

SERVICE_LINUX_ARM64_2 = <AgentType.SERVICE_LINUX_ARM64_2:  32>

SERVICE_OPENWRT_X86_64 = <AgentType.SERVICE_OPENWRT_X86_64:  33>

ASSISTANT_LINUX = <AgentType.ASSISTANT_LINUX: 34>
```

SERVICE_LINUX_ARMADA370_HARDFLOAT = <AgentType.SERVICE_LINUX_ARMADA370_HARDFLOAT: 35>

SERVICE_OPENWRT_X86_64_2 = <AgentType.SERVICE_OPENWRT_X86_64_2: 36>

SERVICE_OPENBSD_X86_64 = <AgentType.SERVICE_OPENBSD_X86_64: 37>

SERVICE_LINUX_MIPSEL24KC = <AgentType.SERVICE_LINUX_MIPSEL24KC: 40>

SERVICE_LINUX_CORTEX_A53 = <AgentType.SERVICE_LINUX_CORTEX_A53: 41>

CONSOLE_WIN_ARM64 = <AgentType.CONSOLE_WIN_ARM64: 42>

SERVICE_WIN_ARM64 = <AgentType.SERVICE_WIN_ARM64: 43>

SERVICE_WIN_X86_32_UNSIGNED = <AgentType.SERVICE_WIN_X86_32_UNSIGNED: 10003>

SERVICE_WIN_X86_64_UNSIGNED = <AgentType.SERVICE_WIN_X86_64_UNSIGNED: 10004>

SERVICE_MACOS_UNIVERSAL_64 = <AgentType.SERVICE_MACOS_UNIVERSAL_64: 10005>

ASSISTANT_WINDOWS = <AgentType.ASSISTANT_WINDOWS: 10006>

COMMAND_WIN_X86_32 = <AgentType.COMMAND_WIN_X86_32: 11000>

COMMAND_WIN_X86_64 = <AgentType.COMMAND_WIN_X86_64: 11001>

flag meshctrl.constants.**ConsentFlags**(*value*)

Bases: IntFlag

Member Type
int

Valid values are as follows:

desktopnotify = <ConsentFlags.desktopnotify: 1>

terminalnotify = <ConsentFlags.terminalnotify: 2>

filesnotify = <ConsentFlags.filesnotify: 4>

desktopprompt = <ConsentFlags.desktopprompt: 8>

terminalprompt = <ConsentFlags.terminalprompt: 16>

filesprompt = <ConsentFlags.filesprompt: 32>

desktopprivacybar = <ConsentFlags.desktopprivacybar: 64>

flag meshctrl.constants.**DeviceRights**(*value*)

Bases: IntFlag

Bitwise flags for device rights Piggy backs on rights for a mesh, but has differet "all" rights.

Member Type
int

Valid values are as follows:

remotecontrol = <DeviceRights.remotecontrol: 8>

agentconsole = <DeviceRights.agentconsole: 16>

serverfiles = <DeviceRights.serverfiles: 32>

wakedevices = <DeviceRights.wakedevices: 64>

notes = <DeviceRights.notes: 128>

desktopviewonly = <DeviceRights.desktopviewonly: 256>

noterminal = <DeviceRights.noterminal: 512>

nofiles = <DeviceRights.nofiles: 1024>

noamt = <DeviceRights.noamt: 2048>

limiteddesktop = <DeviceRights.limiteddesktop: 4096>

limitedevents = <DeviceRights.limitedevents: 8192>

chatnotify = <DeviceRights.chatnotify: 16384>

uninstall = <DeviceRights.uninstall: 32768>

remotecommands = <DeviceRights.remotecommands: 131072>

enum meshctrl.constants.**FileType**(*value*)

Bases: IntEnum

Type numbers used for file types on meshcentral agent.

Member Type
int

Valid values are as follows:

DRIVE = <FileType.DRIVE: 1>

DIRECTORY = <FileType.DIRECTORY: 2>

FILE = <FileType.FILE: 3>

enum meshctrl.constants.**Icon**(*value*)

Bases: IntEnum

Which icon to use for a device

Member Type
int

Valid values are as follows:

desktop = <Icon.desktop: 1>

laptop = <Icon.laptop: 2>

phone = <Icon.phone: 3>

server = <Icon.server: 4>

htpc = <Icon.htpc: 5>

router = <Icon.router: 6>

embedded = <Icon.embedded: 7>

virtual = <Icon.virtual: 8>

enum meshctrl.constants.**InteruserScope**(*value*)

Bases: StrEnum

String constants used to determine the scope of a received *InteruserMessage*

Member Type
str

Valid values are as follows:

user = <InteruserScope.user: 'user'>

session = <InteruserScope.session: 'session'>

flag meshctrl.constants.**MeshFeatures**(*value*)

Bases: IntFlag

Member Type
int

Valid values are as follows:

autoremove = <MeshFeatures.autoremove: 1>

hostnamesync = <MeshFeatures.hostnamesync: 2>

recordsessions = <MeshFeatures.recordsessions: 4>

flag meshctrl.constants.**MeshRights**(*value*)

Bases: IntFlag

Bitwise flags for mesh rights

Member Type
int

Valid values are as follows:

editgroup = <MeshRights.editgroup: 1>

manageusers = <MeshRights.manageusers: 2>

managedevices = <MeshRights.managedevices: 4>

remotecontrol = <MeshRights.remotecontrol: 8>

agentconsole = <MeshRights.agentconsole: 16>

serverfiles = <MeshRights.serverfiles: 32>

wakedevices = <MeshRights.wakedevices: 64>

notes = <MeshRights.notes: 128>

desktopviewonly = <MeshRights.desktopviewonly: 256>

noterminal = <MeshRights.noterminal: 512>

nofiles = <MeshRights.nofiles: 1024>

noamt = <MeshRights.noamt: 2048>

limiteddesktop = <MeshRights.limiteddesktop: 4096>

limitedevents = <MeshRights.limitedevents: 8192>

chatnotify = <MeshRights.chatnotify: 16384>

uninstall = <MeshRights.uninstall: 32768>

noremotedesktop = <MeshRights.noremotedesktop: 65536>

remotecommands = <MeshRights.remotecommands: 131072>

resetpoweroff = <MeshRights.resetpoweroff: 262144>

enum meshctrl.constants.**MeshType**(*value*)

Bases: IntEnum

Which type of Mesh this is.

Member Type
int

Valid values are as follows:

AMT = <MeshType.AMT: 1>

AGENT = <MeshType.AGENT: 2>

LOCAL = <MeshType.LOCAL: 3>

enum meshctrl.constants.**Protocol**(*value*)

Bases: IntEnum

Protocol to use for a tunnel. There are others, but these are what we implement.

Member Type
int

Valid values are as follows:

TERMINAL = <Protocol.TERMINAL: 1>

FILES = <Protocol.FILES: 5>

enum meshctrl.constants.**SharingType**(*value*)

Bases: StrEnum

String constants used to determine which type of device share to create

Member Type
str

Valid values are as follows:

desktop = <SharingType.desktop: 'desktop'>

terminal = <SharingType.terminal: 'terminal'>

enum meshctrl.constants.**SharingTypeInt**(*value*)

Bases: IntEnum

Internal enum used to map SHARINGTYPE to the number used by MeshCentral

Member Type
int

Valid values are as follows:

desktop = <SharingTypeInt.desktop: 1>

terminal = <SharingTypeInt.terminal: 2>

flag meshctrl.constants.**UserRights**(*value*)

Bases: IntFlag

Bitwise flags for user rights

Member Type
int

Valid values are as follows:

backup = <UserRights.backup: 1>

manageusers = <UserRights.manageusers: 2>

restore = <UserRights.restore: 4>

fileaccess = <UserRights.fileaccess: 8>

update = <UserRights.update: 16>

locked = <UserRights.locked: 32>

nonewgroups = <UserRights.nonewgroups: 64>

notools = <UserRights.notools: 128>

usergroups = <UserRights.usergroups: 256>

recordings = <UserRights.recordings: 512>

locksettings = <UserRights.locksettings: 1024>

meshctrl.device module

class meshctrl.device.**Device**(*nodeid*, *session*, *agent=None*, *name=None*, *desc=None*, *description=None*, *tags=None*, *users=None*, *agct=None*, *created_at=None*, *rname=None*, *computer_name=None*, *icon=Icon.desktop*, *mesh=None*, *mtype=None*, *meshtype=None*, *groupname=None*, *meshname=None*, *domain=None*, *host=None*, *ip=None*, *conn=None*, *connected=None*, *pwr=None*, *powered_on=None*, *osdesc=None*, *os_description=None*, *lastaddr=None*, *lastconnect=None*, *links=None*, *details=None*, ***kwargs*)

Bases: object

Object to represent a device. This object is a rough wrapper; it is not guarunteed to be up to date with the state on the server, for instance.

Parameters

- **nodeid** (*str*) – id of the device on the server
- **session** (Session) – Parent session used to run commands
- **agent** (Agent | *dict* | *None*) – Information about the agent. Meshcentral returns this data in an unreadable way, so if the dict doesn't match *Agent*, we will attempt to convert to our format.
- **name** (*str* | *None*) – Device name as it is shown on the meshcentral server
- **description** (*str* | *None*) – Device description as it is shown on the meshcentral server. Also accepted as desc.
- **tags** (*list[str]* | *None*) – tags associated with device.
- **users** (*list[str]* | *None*) – latest known usernames which have logged in.
- **created_at** (*datetime.Datetime* | *int* | *None*) – Time at which device mas created. Also accepted as agct.
- **computer_name** (*str* | *None*) – Device name as reported from the agent. This may be different from name. Also accepted as rname.
- **icon** (Icon) – Icon displayed on the website
- **mesh** (Mesh | *None*) – Mesh object under which this device exists. Is None for individual device access.
- **meshtype** (MeshType | *None*) – Type of mesh this device is connected to. Also accepted as mtype.
- **meshname** (*str* | *None*) – Name of the mesh to which this device is connected. Also accepted as groupname.
- **domain** (*str* | *None*) – Domain on server to which device is connected.
- **host** (*str*) – reachable hostname of device. Not meaningful for agent meshes.
- **ip** (*str*) – IP from which device connected.
- **connected** – (bool): Whether the device is currently connected. Also accepted as conn.
- **powered_on** (*bool*) – Whether the device is currently powered on. Also accepted as pwr.
- **os_description** (*str* | *None*) – Description of the underlying OS. Also accepted as osdesc.
- **lastaddr** (*str* | *None*) – IP from which the agent most recently connected. This may be set even if ip is not.
- **lastconnect** (*datetime.Datetime* | *int* | *None*) – Last time at which the agent was connected to the server
- **links** (*dict[str, UserLink]* | *None*) – Collection of links for the device,
- **details** (*dict[str, dict]* | *None*) – Extra details about the device. These are not well defined, but are filled by calling *list_devices()* with *details=True*.

Returns

Object representing a device on the meshcentral server.

Return type

Device

nodeid

id of the device on the server

Type
str

agent

Information about the agent. Meshcentral returns this data in an unreadable way, so if the dict doesn't match *Agent*, we will attempt to convert to our format.

Type
Agent|dict|None

name

Device name as it is shown on the meshcentral server

Type
str|None

description

Device description as it is shown on the meshcentral server.

Type
str|None

tags

tags associated with device.

Type
list[str]

users

latest known usernames which have logged in.

Type
list[str]

computer_name

Device name as reported from the agent. This may be different from name. Also accepted as rname.

Type
str|None

icon

Icon displayed on the website

Type
Icon

mesh

Mesh object under which this device exists. Is None for individual device access.

Type
Mesh|None

meshtype

Type of mesh this device is connected to. Also accepted as mtype.

Type
MeshType|None

meshname

Name of the mesh to which this device is connected. Also accepted as groupname.

Type
str|None

domain

Domain on server to which device is connected.

Type
str|None

host

reachable hostname of device. Not meaningful for agent meshes.

Type
str

ip

IP from which device connected.

Type
str

connected

(bool): Whether the device is currently connected. Also accepted as conn.

powered_on

Whether the device is currently powered on. Also accepted as pwr.

Type
bool

os_description

Description of the underlying OS. Also accepted as osdesc.

Type
str|None

lastaddr

IP from which the agent most recently connected. This may be set even if ip is not.

Type
str|None

lastconnect

Last time at which the agent was connected to the server

Type
datetime.Datetime|None

links

Collection of links for the device

Type
dict[str, UserLink]|None

details

Extra details about the device. These are not well defined, but are filled by calling *list_devices()* with *details=True*.

Type
dict[str, dict]

async **add_users**(*userids*, *rights=None*, *timeout=None*)

Add a user to an existing node

Parameters

- **userids** (*str|list[str]*) – Unique user id(s)
- **rights** (DeviceRights) – Bitwise mask for the rights on the given device
- **timeout** (*int*) – duration in milliseconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async **edit**(*name=None*, *description=None*, *tags=None*, *icon=None*, *consent=None*, *timeout=None*)

Edit properties of this device

Parameters

- **name** (*str*) – New name for device
- **description** (*str*) – New description for device
- **tags** (*str|list[str]]*) – New tags for device
- **icon** (Icon) – New icon for device
- **consent** (ConsentFlags) – New consent flags for device
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

property **id**

Alias to "nodeid" to be consistent accross types.

async **info**(*timeout=None*)

Get all info for this device. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Object representing the state of the device. This will be a new device, it will not update this device.

Return type
Device

Raises

- **ValueError** – *Invalid device id* if device is not found
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async move_to_device_group(*meshid*, *isname=False*, *timeout=None*)

Move this device another device group

Parameters

- **meshid** (*str*) – Unique mesh id
- **isname** (*bool*) – treat "meshid" as a name instead of an id
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async power_off(*timeout=None*)

Power off device

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async remove(*timeout=None*)

Remove device from MeshCentral

Parameters

- **nodeids** (*str|list[str]*) – nodeid(s) of the device(s) that have to be removed

- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async remove_users(*userids*, *timeout=None*)

Remove users from an this node

Parameters

- **userids** (*str* | *list[str]*) – Unique user id(s)
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async reset(*timeout=None*)

Reset device

Parameters

- **nodeids** (*str* | *list[str]*) – Unique ids of nodes which to reset
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async run_command(*command*, *powershell=False*, *runasuser=False*, *runasuseronly=False*, *ignore_output=False*, *timeout=None*)

Run a command on this device. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **command** (*str*) – Command to run

- **powershell** (*bool*) – Use powershell to run command. Only available on Windows.
- **runasuser** (*bool*) – Attempt to run as a user instead of the root permissions given to the agent. Fall back to root if we cannot.
- **ignore_output** (*bool*) – Don't bother trying to get the output. Every device will return an empty string for its result.
- **runasuseronly** (*bool*) – Error if we cannot run the command as the logged in user.
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Output of command

Return type
RunCommandResponse

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **ValueError** – *Invalid device id* if device is not found
- **asyncio.TimeoutError** – Command timed out

async shell()

Get a terminal shell on this device

Returns
Newly created *Shell*

Return type
Shell

property short_nodeid

nodeid without "node/" or the included domain

async sleep(*timeout=None*)

Sleep device

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async smart_shell(*regex*)

Get a smart terminal shell on this device

Parameters
regex (*regex*) – Regex to watch for to signify that the shell is ready for new input.

Returns
Newly created *SmartShell*

Return type
SmartShell

async wake(*timeout=None*)

Wake up this device

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

meshctrl.exceptions module

exception meshctrl.exceptions.**FileTransferCancelled**(*message*, *stats*)

Bases: *FileTransferError*

Represents a canceled file transfer

exception meshctrl.exceptions.**FileTransferError**(*message*, *stats*)

Bases: *MeshCtrlError*

Represents a failed file transfer

stats

{"result" (str): Human readable result, "size" (int): number of bytes successfully transferred}

Type
dict

exception meshctrl.exceptions.**MeshCtrlError**(*message*, **args*, ***kwargs*)

Bases: Exception

Base class for Meshctrl errors

exception meshctrl.exceptions.**ServerError**(*message*, **args*, ***kwargs*)

Bases: *MeshCtrlError*

Represents an error thrown from the server

exception meshctrl.exceptions.**SocketError**(*message*, **args*, ***kwargs*)

Bases: *MeshCtrlError*

Represents an error in the websocket

meshctrl.files module

class meshctrl.files.**Files**(*session*, *node*)

Bases: *Tunnel*

async close()

async download(*source*, *target*, *skip_http_attempt=False*, *skip_ws_attempt=False*, *timeout=None*)

Download a file from a device into a writable stream.

Parameters

- **source** (*str*) – Path from which to download from device
- **target** (*io.IOBase*) – Stream to which to write data. If None, create new BytesIO which is both readable and writable.
- **skip_http_attempt** (*bool*) – Meshcentral has a way to download files through http(s) instead of through the websocket. This method tends to be much faster than using the websocket, so we try it first. Setting this to True will skip that attempt and just use the established websocket connection.
- **skip_ws_attempt** (*bool*) – Like skip_http_attempt, except just throw an error if the http attempt fails instead of trying with the websocket
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- ***FileTransferError*** – File transfer failed. Info available on the *stats* property
- ***FileTransferCancelled*** – File transfer cancelled. Info available on the *stats* property
- **asyncio.TimeoutError** – Command timed out

Returns

{result: bool whether download succeeded, size: number of bytes downloaded}

Return type

dict

async ls(*directory*, *timeout=None*)

Return a directory listing from the device

Parameters

- **directory** (*str*) – Path to the directory you wish to list
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns

The directory listing

Return type

list[FilesLSItem]

Raises

- ***ServerError*** – Error from server
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async mkdir(*directory*, *timeout=None*)

Create a directory on the device

Parameters

- **directory** (*str*) – Path of directory to create
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- *ServerError* – Error from server
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

Returns
True if directory was created

Return type
bool

async rename(*path*, *name*, *new_name*, *timeout=None*)

Rename a file or folder on the device. This API doesn't error if the file doesn't exist.

Parameters

- **path** (*str*) – Directory from which to rename the file
- **name** (*str*) – File to rename
- **new_name** (*str*) – New name to give the file
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- *ServerError* – Error from server
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

Returns
Info about file renamed. Something along the lines of 'Rename: "/path/to/file" to "newfile"'.

Return type
str

async rm(*path*, *files*, *recursive=False*, *timeout=None*)

Remove a set of files or directories from the device. This API doesn't error if the file doesn't exist.

Parameters

- **path** (*str*) – Directory from which to delete files
- **files** (*str|list[str]*) – File or files to remove from the directory
- **recursive** (*bool*) – Whether to delete the files recursively
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- *ServerError* – Error from server
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

Returns
Info about the files removed. Something along the lines of Delete: "/path/to/file", or 'Delete recursive: "/path/to/dir", n element(s) removed'.

Return type
str

async upload(*source*, *target*, *name=None*, *timeout=None*)

Upload a stream to a device.

Parameters

- **source** (*io.IOBase*) – An IO instance from which to read the data. Must be open for reading.
- **target** (*str*) – Path which to upload stream to on remote device
- **name** (*str*) – Pass if target points at a directory instead of the file path. In that case, this will be the name of the file.
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- ***FileTransferError*** – File transfer failed. Info available on the *stats* property
- ***FileTransferCancelled*** – File transfer cancelled. Info available on the *stats* property
- **asyncio.TimeoutError** – Command timed out

Returns

{result: bool whether upload succeeded, size: number of bytes uploaded}

Return type

dict

meshctrl.mesh module

class meshctrl.mesh.**Mesh**(*meshid*, *session*, *creation=None*, *created_at=None*, *name=None*, *mtype=None*, *meshtype=None*, *creatorid=None*, *desc=None*, *description=None*, *domain=None*, *creatorname=None*, *links=None*, ***kwargs*)

Bases: object

Object to represent a device mesh. This object is a rough wrapper; it is not guarunteed to be up to date with the state on the server, for instance.

Parameters

- **meshid** (*str*) – id of the device mesh on the server
- **session** (Session) – Parent session used to run commands
- **created_at** (*datetime.Datetime*|*int*) – Time at which mesh mas created. Also accepted as creation.
- **name** (*str*|*None*) – Mesh name as it is shown on the meshcentral server
- **description** (*str*|*None*) – Mesh description as it is shown on the meshcentral server. Also accepted as desc.
- **meshtype** (MeshType|*None*) – Type of mesh this device is connected to. Also accepted as mtype.
- **creatorid** (*str*) – User id of the user who created the mesh.
- **creatorname** (*str*) – Display name of the user who created the mesh.
- **domain** (*str*|*None*) – Domain on server to which device is connected.
- **links** (*dict[str, UserLink]*|*None*) – Collection of links for the device group

Returns

Object representing a device group on the meshcentral server.

Return type
Mesh

`meshid`

id of the device mesh on the server

Type
str

`created_at`

Time at which mesh mas created.

Type
datetime.Datetime

`name`

Mesh name as it is shown on the meshcentral server

Type
str|None

`description`

Mesh description as it is shown on the meshcentral server

Type
str|None

`meshtype`

Type of mesh this is.

Type
MeshType|None

`creatorid`

User id of the user who created the mesh.

Type
str|None

`creatorname`

Display name of the user who created the mesh.

Type
str|None

`domain`

Domain on server to which device is connected.

Type
str|None

`links`

Collection of links for the device group

Type
dict[str, UserLink]|None

`async add_users`(*userids*, *rights=0*, *timeout=None*)

Add a user to an existing mesh

Parameters

- **userids** (*str* | *list[str]*) – Unique user id(s)
- **rights** (MeshRights) – Bitwise mask for the rights to give to the users
- **timeout** (*int*) – duration in milliseconds to wait for a response before throwing an error

Returns
Object showing which were added correctly and which were not, along with their result messages. str is userid to map response.

Return type
dict[str, AddUsersToDeviceGroupResponse]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

property id

Alias to "meshid" to be consistent accross types.

property short_meshid

meshid without "mesh/" or the included domain

meshctrl.session module

class meshctrl.session.**Session**(*url*, *user=None*, *domain=None*, *password=None*, *loginkey=None*, *proxy=None*, *token=None*, *ignore_ssl=False*, *auto_reconnect=False*, *user_agent_header=None*)

Bases: object

Class for MeshCentral control session

Parameters

- **url** (*str*) – URL of meshcentral server to connect to. Should start with either "ws://" or "wss://".
- **user** (*str*) – Username of to use for connecting. Can also be username generated from token.
- **domain** (*str*) – Domain to connect to
- **password** (*str*) – Password with which to connect. Can also be password generated from token.
- **loginkey** (*str* | *bytes*) – Key from already handled login. Overrides username/password.
- **proxy** (*str*) – "url:port" to use for proxy server
- **token** (*str*) – Login token. This appears to be superfluous
- **ignore_ssl** (*bool*) – Ignore SSL errors
- **auto_reconnect** (*bool*) – In case of server failure, attempt to auto reconnect. All outstanding requests will be killed.

Returns
Session connected to url

Return type
Session

url

url to which the session is connected

Type
str

initialized

Event marking if the Session initialization has finished. Wait on this to wait for a connection.

Type
asyncio.Event

alive

Whether the session connection is currently alive

Type
bool

closed

Event that occurs when the session closes permanently

Type
asyncio.Event

async add_device_group(*name*, *description=''*, *amtonly=False*, *features=0*, *consent=0*, *timeout=None*)

Create a new device group

Parameters

- **name** (*str*) – Name of device group
- **description** (*str*) – Description of device group
- **amtonly** (*bool*)
- **features** (MeshFeatures) – Bitwise features to enable on the group
- **consent** (ConsentFlags) – Bitwise consent flags to use for the group
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Newly created device group.

Return type
Mesh

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_device_share(*nodeid*, *name*, *type=SharingType.desktop*, *consent=None*, *start=None*, *end=None*, *duration=3600*, *timeout=None*)

Add device share to given node. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences. TODO: This has no tests for it

Parameters

- **nodeid** (*str*) – Unique id of nodes of which to list shares
- **name** (*str*) – Name of guest with which to share

- **type** (SharingType) – Type of share thise should be
- **consent** (ConsentFlags) – Consent flags for share. Defaults to "notify" for your given constants.SharingType
- **start** (*int* | *datetime.datetime*) – When to start the share
- **end** (*int* | *datetime.datetime*) – When to end the share. If None, use duration instead
- **duration** (*int*) – Duration in seconds for share to exist
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Info about the newly created share

Return type
dict

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **ValueError** – If 'end' is some time before 'start'
- **asyncio.TimeoutError** – Command timed out

async add_login_token(*name*, *expire=None*, *timeout=None*)

Create login token for current user. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **name** (*str*) – Name of token
- **expire** (*int*) – Minutes until expiration. 0 or None for no expiration.
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Created token

Return type
LoginToken

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_user(*name*, *password=None*, *randompass=False*, *domain=None*, *email=None*, *emailverified=False*, *resetpass=False*, *realname=None*, *phone=None*, *rights=None*, *timeout=None*)

Add a new user

Parameters

- **name** (*str*) – username
- **password** (*str*) – user's starting password
- **randompass** (*bool*) – Generate a random password for the user. Overrides password
- **domain** (*str*) – Domain to which to add the user

- **email** (*str*) – User's email address
- **emailverified** (*bool*) – Pre-verify the user's email address
- **resetpass** (*bool*) – Force the user to reset their password on first login
- **realname** (*str*) – User's real name
- **phone** (*str*) – User's phone number
- **rights** (UserRights) – Bitwise mask of user's rights on the server
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_user_group(*name*, *domain=None*, *description=None*, *timeout=None*)

Create a new user group

Parameters

- **name** (*str*) – Name of usergroup
- **domain** (*str*) – Domain to which to add the user
- **description** (*str*) – Description of user group
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
New user group

Return type
UserGroup

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_users_to_device(*userids*, *nodeid*, *rights=None*, *timeout=None*)

Add a user to an existing node

Parameters

- **userids** (*str*|*list*[*str*]) – Unique user id(s)
- **nodeid** (*str*) – Node to add the given user to
- **rights** (DeviceRights) – Bitwise mask for the rights on the given device
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_users_to_device_group(*userids*, *meshid*, *isname=False*, *rights=0*, *timeout=None*)

Add a user to an existing mesh

Parameters

- **userids** (*str* | *list[str]*) – Unique user id(s)
- **meshid** (*str*) – Mesh to add the given user to
- **isname** (*bool*) – Read meshid as a name rather than an id
- **rights** (MeshRights) – Bitwise mask for the rights on the given mesh
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Dict showing which were added correctly and which were not, along with their result messages. str is userid to map response.

Return type
dict[str, AddUsersToDeviceGroupResponse]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async add_users_to_user_group(*usernames*, *groupid*, *domain=None*, *timeout=None*)

Add user(s) to an existing user group. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **usernames** (*str* | *list[str]*) – Unique user name(s). This API will not work with the full user ID, but we will try to turn it into something that makes sense.
- **groupid** (*str*) – Group to add the given user to
- **domain** (*str*) – Domain containing the group
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of users that were successfully added. str is username.

Return type
dict[str, AddUsersToUserGroupResponse]

Raises

- ***ServerError*** – Error text from server if there is a failure

- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async broadcast(*message*, *userid=None*, *timeout=None*)

Broadcast a message to all users or a single user TODO: This has no tests for it

Parameters

- **message** (*str*) – Message to broadcast
- **userid** (*str*) – Optional user to which to send the message
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async close()

async classmethod create(**args*, ***kwargs*)

async device_info(*nodeid*, *timeout=None*)

Get all info for a given device. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **nodeid** (*str*) – Unique id of desired node
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Object representing the state of the device

Return type
Device

Raises

- ValueError – *Invalid device id* if device is not found
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async device_message(*nodeid*, *message*, *title='MeshCentral'*, *timeout=None*)

Display a message on remote device. TODO: This has no tests for it

Parameters

- **nodeid** (*str*) – Unique node from which to remove the share
- **message** (*str*) – message to display
- **title** (*str*) – message title

- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful, raile otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async device_open_url(*nodeid*, *url*, *timeout=None*)

Open url in browser on device. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences. TODO: This has no tests for it

Parameters

- **nodeid** (*str*) – Unique node from which to remove the share
- **url** (*str*) – url to open
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- **Exception** – *Failed to open url* if failure occurs
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async device_toast(*nodeids*, *message*, *title='MeshCentral'*, *timeout=None*)

Popup a toast a message on remote device. TODO: This has no tests for it

Parameters

- **nodeids** (*str*|*list*[*str*]) – Unique node from which to remove the share
- **message** (*str*) – message to display
- **title** (*str*) – message title
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure

- asyncio.TimeoutError – Command timed out
- **@todo This function returns True even if it fails, because the server tells us it succeeds before it actually knows, then later tells us it failed, but it's hard to find that because it looks exactly like a success.** –

async **download**(*node*, *source*, *target=None*, *skip_http_attempt=False*, *skip_ws_attempt=False*, *unique_file_tunnel=False*, *timeout=None*)

Download a file from a device into a writable stream. This creates an *Files* and destroys it every call. If you need to upload multiple files, use *file_explorer* instead.

Parameters

- **node** (Device / *str*) – Device or id of device from which to download the file. If it is a device, it must have a ~meshctrl.mesh.Mesh device associated with it (the default). If it is a string, the device will be fetched prior to tunnel creation.
- **source** (*str*) – Path from which to download from device
- **target** (*io.IOBase*) – Stream to which to write data. If None, create new BytesIO which is both readable and writable.
- **skip_http_attempt** (*bool*) – Meshcentral has a way to download files through http(s) instead of through the websocket. This method tends to be much faster than using the websocket, so we try it first. Setting this to True will skip that attempt and just use the established websocket connection.
- **skip_ws_attempt** (*bool*) – Like skip_http_attempt, except just throw an error if the http attempt fails instead of trying with the websocket
- **unique_file_tunnel** (*bool*) – True: Create a unique *Files* for this call, which will be cleaned up on return, else use cached or cache *Files*
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- *FileTransferError* – File transfer failed. Info available on the *stats* property
- *FileTransferCancelled* – File transfer cancelled. Info available on the *stats* property

Returns

The stream which has been downloaded into. Cursor will be at the beginning of where the file is downloaded.

Return type

io.IOBase

async **download_file**(*node*, *source*, *filepath*, *skip_http_attempt=False*, *skip_ws_attempt=False*, *unique_file_tunnel=False*, *timeout=None*)

Friendly wrapper around *download* to download to a filepath. Creates a WritableStream and calls download.

Parameters

- **node** (Device / *str*) – Device or id of device from which to download the file. If it is a device, it must have a ~meshctrl.mesh.Mesh device associated with it (the default). If it is a string, the device will be fetched prior to tunnel creation.
- **source** (*str*) – Path from which to download from device
- **filepath** (*str*) – Path to which to download data

- **skip_http_attempt** (*bool*) – Meshcentral has a way to download files through http(s) instead of through the websocket. This method tends to be much faster than using the websocket, so we try it first. Setting this to True will skip that attempt and just use the established websocket connection.
- **skip_ws_attempt** (*bool*) – Like skip_http_attempt, except just throw an error if the http attempt fails instead of trying with the websocket
- **unique_file_tunnel** (*bool*) – True: Create a unique *Files* for this call, which will be cleaned up on return, else use cached or cache *Files*
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- ***FileTransferError*** – File transfer failed. Info available on the *stats* property
- ***FileTransferCancelled*** – File transfer cancelled. Info available on the *stats* property

Returns
None

async edit_device(*nodeid*, *name=None*, *description=None*, *tags=None*, *icon=None*, *consent=None*, *timeout=None*)

Edit properties of an existing device

Parameters

- **nodeid** (*str*) – Unique id of desired node
- **name** (*str*) – New name for device
- **description** (*str*) – New description for device
- **tags** (*str|list[str]*) – New tags for device
- **icon** (Icon) – New icon for device
- **consent** (ConsentFlags) – New consent flags for device
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async edit_device_group(*meshid*, *isname=False*, *name=None*, *description=None*, *flags=None*, *consent=None*, *invite_codes=None*, *backgroundonly=False*, *interactiveonly=False*, *timeout=10*)

Edit an existing device group. WARNING: This command will just hang if you do not have permissions. Because of this, timeout is defaulted to 10 seconds. Be wary if you remove the timeout.

Parameters

- **meshid** (*str*) – Unique id of device group

- **isname** (*bool*) – treat "meshid" as a name instead of an id
- **name** (*str*) – New name for group
- **description** (*str*) – New description
- **flags** (MeshFeatures) – Features to enable on the group
- **consent** (ConsentFlags) – Which consent flags to use for the group
- **invite_codes** (*list[str] | True*) – Create new invite codes. If True, pass "*". I don't know what this means.
- **backgroundonly** (*bool*) – Flag for invite codes
- **interactiveonly** (*bool*) – Flag for invite codes
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async edit_user(*userid*, *domain=None*, *email=None*, *emailverified=False*, *resetpass=False*, *realname=None*, *phone=None*, *rights=None*, *timeout=None*)

Edit an existing user

Parameters

- **userid** (*str*) – Unique userid
- **domain** (*str*) – Domain to which to add the user
- **email** (*str*) – User's email address
- **emailverified** (*bool*) – Verify or unverify the user's email address
- **resetpass** (*bool*) – Force the user to reset their password on next login
- **realname** (*str*) – User's real name
- **phone** (*str*) – User's phone number
- **rights** (UserRights) – Bitwise mask of user's rights on the server
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure

- asyncio.TimeoutError – Command timed out

async events(*filter=None*)

Listen to events from the server

Parameters
filter (*dict*) – dict to filter events with. Only trigger for events that deep-match this dict. Use sets for "array.contains" and arrays for equality of lists.

Returns
A generator with the events that match the given filter, or all events if no filter is given

Return type
generator(data)

file_explorer(*node*)

Create, initialize, and return an *Files* object for the given node

Parameters
node (Device / *str*) – Device or id of device on which to open file explorer. If it is a device, it must have a ~meshctrl.mesh.Mesh device associated with it (the default). If it is a string, the device will be fetched prior to tunnel creation.

Returns
A newly initialized file explorer.

Return type
Files

async generate_invite_link(*group*, *hours*, *flags=None*, *meshid=None*, *timeout=None*)

Generate an invite link for a group or mesh TODO: This has no tests for it

Parameters

- **group** (*str*) – Name of group to add
- **hours** (*int*) – Hours until link expires
- **flags** (MeshRights / DeviceRights) – Bitwise flags representing rights for device/mesh
- **meshid** (*str*) – ID of mesh which to invite user. Overrides "group"
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Invite link information

Return type
dict

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async interuser(*data*, *session=None*, *user=None*)

Fire off an interuser message. This is a fire and forget api, we have no way of checking if the user got the message. User will recieve an *InteruserMessage* if they are allowed to receive interuser messages from you.

Parameters

- **data** (*serializable*) – Any sort of serializable data you want to send to the user
- **session** (*str*) – Direct session to send to. Use this after you have made connection with a specific user session.
- **user** (*str*) – Send message to all sessions of a particular user. One of these must be set.

Raises

- **ValueError** – Value error if neither user nor session are given.
- ***SocketError*** – Info about socket closure

async list_device_groups(*timeout=None*)

Get device groups. Only returns meshes to which the logged in user has access

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of meshes

Return type
list[Mesh]

Raises

- ***ServerError*** – Error from server
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_device_shares(*nodeid*, *timeout=None*)

List device shares of given node. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **nodeid** (*str*) – Unique id of nodes of which to list shares
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Array of dicts representing device shares

Return type
list[dict]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_devices(*details=False*, *group=None*, *meshid=None*, *timeout=None*)

Get devices to which the user has access. Different options will fill different properties in the resultant device objects, based on what is returned from meshcentral. Documenting these changes is beyond the scope of this documentation.

Parameters

- **details** (*bool*) – Get device details, overrides group and meshid
- **group** (*str*) – Get devices from specific group by name. Overrides meshid
- **meshid** (*str*) – Get devices from specific group by id

- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of nodes

Return type
list[Device]

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_events(*userid=None*, *nodeid=None*, *limit=None*, *timeout=None*)

List events visible to the currect user

Parameters

- **userid** (*str*) – Filter by user. Overrides nodeid. Only works for admin, otherwise ignored.
- **nodeid** (*str*) – Filter by node
- **limit** (*int*) – Limit to the N most recent events
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of events

Return type
list[dict]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_login_tokens(*timeout=None*)

List login tokens for current user. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of tokens

Return type
list[RetrievedLoginToken]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_user_groups(*timeout=None*)

Get user groups. Admin will get all user groups, otherwise get limited user groups

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of groups. If you are not a member, you'l just get the names and ids.

Return type
list[UserGroup]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_user_sessions(*timeout=None*)

Get list of connected users. Admin Only.

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Number of sessions per user

Return type
dict[str, int]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async list_users(*timeout=None*)

List users on server. Admin Only.

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of users

Return type
list[ListUsersResponseItem]

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async move_to_device_group(*nodeids*, *meshid*, *isname=False*, *timeout=None*)

Move a device from one group to another

Parameters

- **nodeids** (*str* | *list[str]*) – Unique node id(s)
- **meshid** (*str*) – Unique mesh id
- **isname** (*bool*) – treat "meshid" as a name instead of an id
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async ping(*timeout=None*)

Ping the server. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters
timeout (*int*) – duration in seconds to wait for a response before throwing an error

Returns
{"action": "pong"}

Return type
dict

Raises

- *ServerError* – Error from server
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async power_off_devices(*nodeids*, *timeout=None*)

Power off given devices TODO: This has no tests for it

Parameters

- **nodeids** (*str|list[str]*) – Unique ids of nodes which to power off
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async raw_messages()

Listen to raw messages from the server. These will be strings that have not been parsed at all. Consider this an emergency fallback if meshcentral sends something odd. You will get every message from the websocket.

Returns
A generator which will generate every message the server sends

Return type
generator(data)

async remove_device_group(*meshid*, *isname=False*, *timeout=None*)

Remove an existing device group

Parameters

- **meshid** (*str*) – Unique id of device group
- **isname** (*bool*) – treat "meshid" as a name instead of an id
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns

True on success, raise otherwise

Return type

bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async remove_device_share(*nodeid*, *shareid*, *timeout=None*)

Remove a device share TODO: This has no tests for it

Parameters

- **nodeid** (*str*) – Unique node from which to remove the share
- **shareid** (*str*) – Unique share id to be removed
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns

True if successful, raise otherwise

Return type

bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async remove_devices(*nodeids*, *timeout=None*)

Remove device(s) from MeshCentral

Parameters

- **nodeids** (*str* | *list[str]*) – nodeid(s) of the device(s) that have to be removed
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns

True on success, raise otherwise

Return type

bool

Raises

- *ServerError* – Error text from server if there is a failure

- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_login_token(*names*, *timeout=None*)

Remove login token for current user. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **name** (*str*) – Name of token or token username
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
List of remaining tokens

Return type
list[RetrievedLoginToken]

Raises

- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_user(*userid*, *domain=None*, *timeout=None*)

Remove an existing user

Parameters

- **userid** (*str*) – Unique userid
- **domain** (*str*) – Domain to which to add the user
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_user_from_user_group(*userid*, *groupid*, *domain=None*, *timeout=None*)

Remove user from an existing user group

Parameters

- **userid** (*str*) – Unique user id
- **groupid** (*str*) – Group to remove the given user from
- **domain** (*str*) – Domain containing the group
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_user_group(*groupid*, *domain=None*, *timeout=None*)

Remove an existing user group

Parameters

- **userid** (*str*) – Unique userid
- **domain** (*str*) – Domain to which to add the user
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_users_from_device(*nodeid*, *userids*, *timeout=None*)

Remove users from an existing node

Parameters

- **nodeid** (*str*) – Node to remove the given users from
- **userids** (*str*|*list[str]*) – Unique user id(s)
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async remove_users_from_device_group(*userids*, *meshid*, *isname=False*, *timeout=None*)

Remove users from an existing mesh

Parameters

- **userids** (*str*|*list[str]*) – Unique user id(s)

- **meshid** (*str*) – Mesh to add the given user to
- **isname** (*bool*) – Read meshid as a name rather than an id
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Dict showing which were removed correctly and which were not, along with their result messages. str is userid to map response.

Return type
dict[str, AddUsersToDeviceGroupResponse]

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async reset_devices(*nodeids*, *timeout=None*)

Reset given devices TODO: This has no tests for it

Parameters

- **nodeids** (*str*|*list[str]*) – Unique ids of nodes which to reset
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

async run_command(*nodeids*, *command*, *powershell=False*, *runasuser=False*, *runasuseronly=False*, *ignore_output=False*, *timeout=None*)

Run a command on any number of nodes. WARNING: Non namespaced call on older versions of meshcentral (<1.0.22). Calling this function on those versions again before it returns may cause unintended consequences.

Parameters

- **nodeids** (*str*|*list[str]*) – Unique ids of nodes on which to run the command
- **command** (*str*) – Command to run
- **powershell** (*bool*) – Use powershell to run command. Only available on Windows.
- **runasuser** (*bool*) – Attempt to run as a user instead of the root permissions given to the agent. Fall back to root if we cannot.
- **ignore_output** (*bool*) – Don't bother trying to get the output. Every device will return an empty string for its result.
- **runasuseronly** (*bool*) – Error if we cannot run the command as the logged in user.
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Dict containing mapped output of the commands by device

Return type
dict[str, RunCommandResponse]

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **ValueError** – *Invalid device id* if device is not found
- **asyncio.TimeoutError** – Command timed out

async run_console_command(*nodeids*, *command*, *powershell=False*, *runasuser=False*, *runasuseronly=False*, *ignore_output=False*, *timeout=None*)

Run a mesh console command on any number of nodes. WARNING: Non namespaced call. Calling this function again before it returns may cause unintended consequences.

Parameters

- **nodeids** (*str|list[str]*) – Unique ids of nodes on which to run the command
- **command** (*str*) – Command to run
- **ignore_output** (*bool*) – Don't bother trying to get the output. Every device will return an empty string for its result.
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
Dict containing mapped output of the commands by device

Return type
dict[str, RunCommandResponse]

Raises

- ***ServerError*** – Error text from server if there is a failure
- ***SocketError*** – Info about socket closure
- **ValueError** – *Invalid device id* if device is not found
- **asyncio.TimeoutError** – Command timed out

async send_invite_email(*group*, *email*, *name=None*, *message=None*, *meshid=None*, *timeout=None*)

Send an invite email for a group or mesh TODO: This has no tests for it

Parameters

- **group** (*str*) – Name of mesh to which to invite email
- **email** (*str*) – Email of user to invite
- **name** (*str*) – User's name. For display purposes.
- **message** (*str*) – Message to send to user in invite email
- **meshid** (*str*) – ID of mesh which to invite user. Overrides "group"
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True on success, raise otherwise

Return type
bool

Raises

- *ServerError* – Error text from server if there is a failure
- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

async server_info()

Get server information

Returns
(dict) Server info

shell(*nodeid*)

Get a terminal shell on the given device

Parameters
nodeid (*str*) – Unique id of node on which to open the shell

Returns
Newly created and initialized *Shell* or cached *Shell* if unique is False and a shell is currently active

Return type
Shell

async sleep_devices(*nodeids*, *timeout=None*)

Sleep given devices TODO: This has no tests for it

Parameters

- **nodeids** (*str*|*list[str]*) – Unique ids of nodes which to sleep
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- *SocketError* – Info about socket closure
- asyncio.TimeoutError – Command timed out

smart_shell(*nodeid*, *regex*)

Get a smart terminal shell on the given device

Parameters

- **nodeid** (*str*) – Unique id of node on which to open the shell
- **regex** (*regex*) – Regex to watch for to signify that the shell is ready for new input.

Returns
Newly created and initialized *SmartShell* or cached *SmartShell* if unique is False and a smartshell with regex is currently active

Return type
SmartShell

async upload(*node*, *source*, *target*, *unique_file_tunnel=False*, *timeout=None*)

Upload a stream to a device.

Parameters

- **node** (Device / *str*) – Device or id of device to which to upload the file. If it is a device, it must have a ~meshctrl.mesh.Mesh device associated with it (the default). If it is a string, the device will be fetched prior to tunnel creation.
- **source** (*io.IOBase*) – An IO instance from which to read the data. Must be open for reading.
- **target** (*str*) – Path which to upload stream to on remote device
- **unique_file_tunnel** (*bool*) – True: Create a unique *Files* for this call, which will be cleaned up on return, else use cached or cache *Files*
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- ***FileTransferError*** – File transfer failed. Info available on the *stats* property
- ***FileTransferCancelled*** – File transfer cancelled. Info available on the *stats* property

Returns

{result: bool whether upload succeeded, size: number of bytes uploaded}

Return type

dict

async upload_file(*node*, *filepath*, *target*, *unique_file_tunnel=False*, *timeout=None*)

Friendly wrapper around *upload* to upload from a filepath. Creates a ReadableStream and calls upload.

Parameters

- **node** (Device / *str*) – Device or id of device to which to upload the file. If it is a device, it must have a ~meshctrl.mesh.Mesh device associated with it (the default). If it is a string, the device will be fetched prior to tunnel creation.
- **filepath** (*str*) – Path from which to read the data
- **target** (*str*) – Path which to upload file to on remote device
- **unique_file_tunnel** (*bool*) – True: Create a unique *Files* for this call, which will be cleaned up on return, else use cached or cache *Files*
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Raises

- ***FileTransferError*** – File transfer failed. Info available on the *stats* property
- ***FileTransferCancelled*** – File transfer cancelled. Info available on the *stats* property

Returns

{result: bool whether upload succeeded, size: number of bytes uploaded}

Return type

dict

async user_info()

Get user information

Returns

(dict) User info

async wake_devices(*nodeids*, *timeout=None*)

Wake up given devices TODO: This has no tests for it

Parameters

- **nodeids** (*str* | *list[str]*) – Unique ids of nodes which to wake
- **timeout** (*int*) – duration in seconds to wait for a response before throwing an error

Returns
True if successful

Return type
bool

Raises

- ***SocketError*** – Info about socket closure
- **asyncio.TimeoutError** – Command timed out

meshctrl.shell module

class meshctrl.shell.**Shell**(*session*, *nodeid*)

Bases: *Tunnel*

async expect(*regex*, *timeout=None*)

Read data from the shell until *regex* is seen

Parameters

- **regex** (*str* | *re.Pattern*) – Regex to check for match
- **timeout** (*int*) – Milliseconds to wait for data. None == read until *length* bytes are read, or shell is closed.

Returns
Data read.

Return type
str

Raises
asyncio.TimeoutError – Regex not matched within timeout

async read(*length=None*, *block=True*, *timeout=None*)

Read data from the shell

Parameters

- **length** (*int*) – Number of bytes to read. None == read until closed or timeout occurs.
- **block** (*bool*) – block until n bytes are available or timeout occurs. If not, read at most until no data is returned. This may return an indeterminate amount of data.
- **timeout** (*int*) – Milliseconds to wait for data. None == read until *length* bytes are read, or shell is closed.

Returns
Data read. In the case of timeout, this will return all data read up to the timeout

Return type
str

async write(*command*)

Write to the shell

Parameters
command (*str*) – Command to send

Returns
None

class meshctrl.shell.**SmartShell**(*shell*, *regex*)

Bases: object

property alive

async close()

property closed

property initialized

async send_command(*command*, *timeout=None*)

meshctrl.tunnel module

class meshctrl.tunnel.**Tunnel**(*session*, *node_id*, *protocol*)

Bases: object

async close()

meshctrl.types module

This module attempts to define the various structures of objects returned by meshcentral calls. These types are expected, but not guarunteed, as the meshcentral API is not well defined.

class meshctrl.types.**AddDeviceGroupResponse**

Bases: TypedDict

Response proffered when device group is added

links: dict[str, list[*UserLink*]]

Users of this device group, and their rights

meshid: str

ID of the created device group

class meshctrl.types.**AddUsersToDeviceGroupResponse**

Bases: TypedDict

Response proffered when a user is added to a device group

message: str

Any interesting information about adding the user. "Added user {username}" on success, otherwise defined by server

success: bool

Whether the user was added successfully

class meshctrl.types.**AddUsersToUserGroupResponse**

Bases: TypedDict

Response item from add_users_to_user_group execution

message: str

Message from server if user was not added correctly

success: bool

Whether the user was added successfully

class meshctrl.types.**Agent**

Bases: TypedDict

Information about an agent running on a machine

capabilities: *AgentCapabilities*

Capabilities of this agent. This can change over time based on the connection state of the agent

id: *AgentType*

Type of agent this is

version: int

Agent version

class meshctrl.types.**DeviceGroup**

Bases: TypedDict

Device group information

links: dict[str, list[*UserLink*]]

Users of this device group, and their rights

meshid: str

ID of the created device group

class meshctrl.types.**FilesLSItem**

Bases: TypedDict

Dict representing a file or directory on a mesh device, as returned from meshcentral server

d: str

UTC timestamp for when the file/directory was edited

dt: str | None

Drive type, in the case t == *DRIVE*

f: int | None

Free bytes on the drive, if t == *DRIVE*

n: str

Name of file or dir

s: int | None

Size of the file if t == *FILE*

t: *FileType*

Type of file

class meshctrl.types.**InteruserMessage**

Bases: TypedDict

Message received from another user through interuser messaging

action: **interuser**

Identifier of event type

data: **str**

Any data the user sent

scope: *InteruserScope*

"user" if the message was sent to your username, "session" if it was sent to this specific session.

sessionid: **str**

Session from which the message originated

class meshctrl.types.**ListUsersResponseItem**

Bases: TypedDict

Item contained in response from *list_users()*

creation: **int**

When the user was created. UTC timestamp

links: **dict[str, list[*UserLink*]]**

Set of links connected to this user. str is the ID of the thing referenced, the type of which is hinted at by the beginning of the string. These describe numerous things, the enumeration of which is beyond this documentation

name: **str**

User's username

class meshctrl.types.**LoginToken**

Bases: TypedDict

Login token created on the server

created: **int**

UTC timestamp representing when the token was created. In milliseconds

expire: **int**

UTC timestamp representing when the token expires. In milliseconds. 0 means no expirery.

name: **str**

Name of the token

tokenPass: **str**

Password substitute for the token

tokenUser: **str**

Username substitute of the token

class meshctrl.types.**RemoveUsersFormDeviceGroupResponse**

Bases: *AddUsersToDeviceGroupResponse*

Response proffered when a user is removed from a device group

message: **str**

success: bool

class meshctrl.types.**RetrievedLoginToken**

Bases: TypedDict

Login token created on the server and retrieved. This will not include the password.

created: int

UTC timestamp representing when the token was created. In milliseconds

expire: int

UTC timestamp representing when the token expires. In milliseconds. 0 means no expirery.

name: str

Name of the token

tokenUser: str

Username substitute of the token

class meshctrl.types.**RunCommandResponse**

Bases: TypedDict

Response item from run_command execution

command: str

The command which was run

complete: bool

Whether the command completed correctly

result: str

Output of command

class meshctrl.types.**UserLink**

Bases: TypedDict

Represents a link to a user account

name: str

Username of the user this link references

rights: *MeshRights* | *DeviceRights*

User's rights on the containing object

meshctrl.user_group module

class meshctrl.user_group.**UserGroup**(*ugrpid*, *session*, *name=None*, *desc=None*, *description=None*, *domain=None*, *links=None*, ***kwargs*)

Bases: object

Object to represent a user group. This object is a rough wrapper; it is not guarunteed to be up to date with the state on the server, for instance.

Parameters

- **ugrpid** (*str*) – id of the user group on the server
- **session** (Session) – Parent session used to run commands
- **name** (*str* | *None*) – Mesh name as it is shown on the meshcentral server

- **description** (*str|None*) – Mesh description as it is shown on the meshcentral server. Also accepted as desc.
- **domain** (*str|None*) – Domain on server to which device is connected.
- **links** (*dict[str, UserLink]|None*) – Collection of links for the device group

Returns
Object representing a device group on the meshcentral server.

Return type
Mesh

ugrpid
id of the device mesh on the server

Type
str

name
Mesh name as it is shown on the meshcentral server

Type
str|None

description
Mesh description as it is shown on the meshcentral server

Type
str|None

domain
Domain on server to which device is connected.

Type
str|None

links
Collection of links for the device group

Type
dict[str, UserLink]|None

async add_users(*userids*, *timeout=None*)
Add a user to an existing mesh

Parameters

- **userids** (*str|list[str]*) – Unique user id(s)
- **rights** (*MeshRights*) – Bitwise mask for the rights to give to the users
- **timeout** (*int*) – duration in milliseconds to wait for a response before throwing an error

Returns
Object showing which were added correctly and which were not, along with their result messages. str is userid to map response.

Return type
dict[str, AddUsersToDeviceGroupResponse]

Raises

- ***SocketError*** – Info about socket closure

- asyncio.TimeoutError – Command timed out

property id

Alias to "ugrpid" to be consistent accross types.

property short_ugrpid

ugrpid without "ugrp/" or the included domain

meshctrl.util module

class meshctrl.util.Eventer

Bases: object

Eventer object to allow pub/sub interactions with a Session object

async emit(*event*, *data*)

Emit *event* with *data*. All subscribed functions will be called (order is nonsensical).

Parameters

- **event** (*str*) – Event name emit
- **data** (*object*) – Data to pass to all the bound functions

off(*event*, *func*)

Unsubscribe from *event*. *func* is the object originally passed during the bind.

Parameters

- **event** (*str*) – Event name to unsubscribe from
- **func** (*object*) – Function which was originally passed when subscribing.

on(*event*, *func*)

Subscribe to *event*. *func* will be called when that event is emitted.

Parameters

- **event** (*str*) – Event name to subscribe to
- **(function(data** (*func*) – object)): Function to call when event is emitted. *data* could be of any type. Also used as a key to remove this subscription.

once(*event*, *func*)

Subscribe to *event* once. *func* will be called when that event is emitted. The binding will then be removed.

Parameters

- **event** (*str*) – Event name to subscribe to
- **(function(data** (*func*) – object)): Function to call when event is emitted. *data* could be of any type. Also used as a key to remove this subscription.

meshctrl.util.compare_dict(*dict1*, *dict2*)

Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W