
vidhub-control Documentation

Release 0.0.1

Matthew Reid

Nov 18, 2021

CONTENTS:

1	Overview	1
1.1	Links	1
1.2	Dependencies	1
1.3	Installation	2
1.4	Usage	3
2	Reference	5
2.1	vidhubcontrol.config	5
2.2	vidhubcontrol.backends	9
2.3	vidhubcontrol.discovery	16
2.4	vidhubcontrol.common	19
3	Indices and tables	23
	Python Module Index	25
	Index	27

OVERVIEW

Interface with Videohub SDI Matrix Switchers and SmartView Monitors by [Blackmagic Design](#).

The primary purpose is for use as a library in other applications, but a GUI application is included (requires installation of the [Kivy framework](#)

Since neither the devices nor the software for them support presets or macros, a need arose for instantaneous multiple routing changes. This, as well as setting the names for inputs and outputs within a single application can be accomplished using this project.

1.1 Links

Releases	https://pypi.org/project/vidhub-control/
Source code	https://github.com/nocarryr/vidhub-control
Documentation	https://vidhub-control.readthedocs.io/

1.2 Dependencies

This project relies heavily on *asyncio* and other features available in **Python v3.5** or later.

Core

- [python-dispatch](#)
- [json-object-factory](#)
- [zeroconf](#)
- [python-osc](#)
- [pid](#)

User interface (optional)

- [Kivy](#)

1.3 Installation

1.3.1 Download

For basic installation, clone or download the source code:

```
git clone https://github.com/nocarryr/vidhub-control
cd vidhub-control
```

1.3.2 Create virtual environment

(optional, but recommended)

Linux/MacOS

```
virtualenv --python=python3 venv
source venv/bin/activate
```

Windows

```
virtualenv --python=python3 venv
venv/Scripts/activate
```

1.3.3 Install vidhub-control

```
python setup.py install
```

1.3.4 Install Kivy

(optional)

Ensure all dependencies are met for your platform. Instructions can be found on the [kivy download page](#)

Linux (Ubuntu)

Follow the instructions for [Installation in a Virtual Environment](#)

Windows

```
pip install docutils pygments pypiwin32 kivy.deps.sdl2 kivy.deps.glew
pip install kivy.deps.sdl2
pip install kivy
```

MacOS

Follow the instructions for [homebrew](#) or [MacPorts](#).

1.4 Usage

To launch the user interface (Kivy required):

```
vidhubcontrol-ui
```

1.4.1 Note for Windows

The *vidhubcontrol-ui* script may not work. If this is the case, it can be launched by:

```
python vidhubcontrol/kivyui/main.py
```


REFERENCE

2.1 vidhubcontrol.config

class vidhubcontrol.config.**Config**(*args, **kwargs)

Bases: vidhubcontrol.config.ConfigBase

Config store for devices

Handles storage of device connection information and any user-defined values for the backends defined in the *backends module*. Data is stored in JSON format.

During *start()*, all previously stored devices will be loaded and begin communication. Devices are also discovered using *Zeroconf* through the *discovery module*.

Since each device has a unique id, network address changes (due to DHCP, etc) are handled appropriately.

The configuration data is stored when:

- A device is added or removed
- A change is detected for a device's network address
- Any user-defined device value changes (device name, presets, etc)

The recommended method to start Config is through the *load_async()* method.

Example

```
import asyncio
from vidhubcontrol.config import Config

loop = asyncio.get_event_loop()
conf = loop.run_until_complete(Config.load_async())
```

Keyword Arguments **filename** (*str*, optional) – Filename to load/save config data to. If not given, defaults to *DEFAULT_FILENAME*

vidhubs

A *DictProperty* of *VidhubConfig* instances using *device_id* as keys

Type Dict[str, vidhubcontrol.config.VidhubConfig]

smartviews

A *DictProperty* of *SmartViewConfig* instances using *device_id* as keys

Type Dict[str, *vidhubcontrol.config.SmartViewConfig*]

smartscope

A DictProperty of *SmartScopeConfig* instances using *device_id* as keys

Type Dict[str, *vidhubcontrol.config.SmartScopeConfig*]

all_devices

A DictProperty containing all devices from *vidhubs*, *smartviews* and *smartscope*s

Type Dict[str, *vidhubcontrol.config.DeviceConfigBase*]

DEFAULT_FILENAME = '~/.vidhubcontrol.json'

async add_device(backend)

Adds a “backend” instance to the config

A subclass of *DeviceConfigBase* will be either created or updated from the given backend instance.

If the *device_id* exists in the config, the *DeviceConfigBase.backend* value of the matching *DeviceConfigBase* instance will be set to the given backend. Otherwise, a new *DeviceConfigBase* instance will be created using the *DeviceConfigBase.from_existing()* classmethod.

Parameters **backend** – An instance of one of the subclasses of *vidhubcontrol.backends.base.BackendBase* found in *vidhubcontrol.backends*

async build_backend(device_type, backend_name, **kwargs)

Creates a “backend” instance

The supplied keyword arguments are used to create the instance object which will be created using its *create()* classmethod.

The appropriate subclass of *DeviceConfigBase* will be created and stored to the config using *add_device()*.

Parameters

- **device_type** (*str*) – Device type to create. Choices are “vidhub”, “smartview”, “smartscope”
- **backend_name** (*str*) – The class name of the backend as found in *vidhubcontrol.backends*

Returns An instance of a *vidhubcontrol.backends.base.BackendBase* subclass

connection_manager: *vidhubcontrol.common.ConnectionManager*

Connection manager

property connection_state: *vidhubcontrol.common.ConnectionState*

The current *state* of the *connection_manager*

classmethod load(filename=None, **kwargs)

Creates a Config instance, loading data from the given filename

Parameters **filename** (*str*, optional) – The filename to read config data from, defaults to *Config.DEFAULT_FILENAME*

Returns A *Config* instance

async classmethod load_async(filename=None, **kwargs)

Creates a Config instance, loading data from the given filename

This coroutine method creates the Config instance and will await all start-up coroutines and futures before returning.

Parameters **filename** (*str*, optional) – The filename to read config data from, defaults to *DEFAULT_FILENAME*

Returns A *Config* instance

save(*filename=None*)

Saves the config data to the given filename

Parameters **filename** (*str*, optional) – The filename to write config data to. If not supplied, the current filename is used.

Notes

If the filename argument is provided, it will replace the existing filename value.

async start(***kwargs*)

Starts the device backends and discovery routines

Keyword arguments passed to the initialization will be used here, but can be overridden in this method. They will also be passed to `_initialize_backends()`.

async stop()

Stops all device backends and discovery routines

class vidhubcontrol.config.DeviceConfigBase(**args, **kwargs*)

Bases: vidhubcontrol.config.ConfigBase

Base class for device config storage

config

A reference to the parent *Config* instance

Type *vidhubcontrol.config.Config*

backend

An instance of *vidhubcontrol.backends.base.BackendBase*

Type *vidhubcontrol.backends.base.BackendBase*

backend_name

The class name of the backend, used when loading from saved config data

Type *str*

hostaddr

The IPv4 address of the device

Type *str*

hostport

The port address of the device

Type *int*

device_name

User-defined name to store with the device, defaults to the *device_id* value

Type *str*

device_id

The unique id as reported by the device

Type *str*

async build_backend(*cls=None, **kwargs*)

Creates a backend instance asynchronously

Keyword arguments will be passed to the `vidhubcontrol.backends.base.BackendBase.create_async()` method.

Parameters `cls` (*optional*) – A subclass of `BackendBase`. If not present, the class will be determined from existing values of `device_type` and `backend_name`

Returns An instance of `vidhubcontrol.backends.base.BackendBase`

connection_manager: `vidhubcontrol.common.SynchronizedConnectionManager`

A connection manager that synchronizes its state with the `backend`

property connection_state: `vidhubcontrol.common.ConnectionState`

The current state of the `connection_manager`

async classmethod create(***kwargs*)

Creates device config and backend instances asynchronously

Keyword arguments passed to this classmethod are passed to the `init` method and will be used to set its attributes.

If a “backend” keyword argument is supplied, it should be a running instance of `vidhubcontrol.backends.base.BackendBase`. It will then be used to collect config values from.

If “backend” is not present, the appropriate one will be created using `build_backend()`.

Returns An instance of `DeviceConfigBase`

async classmethod from_existing(`backend`, ***kwargs*)

Creates a device config object from an existing backend

Keyword arguments will be passed to the `create()` method

Parameters `backend` – An instance of `vidhubcontrol.backends.base.BackendBase`

Returns An instance of `DeviceConfigBase`

class `vidhubcontrol.config.SmartScopeConfig`(**args*, ***kwargs*)

Bases: `vidhubcontrol.config.DeviceConfigBase`

Config container for SmartScope devices

class `vidhubcontrol.config.SmartViewConfig`(**args*, ***kwargs*)

Bases: `vidhubcontrol.config.DeviceConfigBase`

Config container for SmartView devices

class `vidhubcontrol.config.VidhubConfig`(**args*, ***kwargs*)

Bases: `vidhubcontrol.config.DeviceConfigBase`

Config container for VideoHub devices

presets

Preset data collected from the device `presets`. Will be used on initialization to populate the preset data to the device

Type `List[Dict]`

async build_backend(`cls=None`, ***kwargs*)

Creates a backend instance asynchronously

Keyword arguments will be passed to the `vidhubcontrol.backends.base.BackendBase.create_async()` method.

Parameters `cls` (*optional*) – A subclass of `BackendBase`. If not present, the class will be determined from existing values of `device_type` and `backend_name`

Returns An instance of `vidhubcontrol.backends.base.BackendBase`

async classmethod create(**kwargs)

Creates device config and backend instances asynchronously

Keyword arguments passed to this classmethod are passed to the init method and will be used to set its attributes.

If a “backend” keyword argument is supplied, it should be a running instance of `vidhubcontrol.backends.base.BackendBase`. It will then be used to collect config values from.

If “backend” is not present, the appropriate one will be created using `build_backend()`.

Returns An instance of `DeviceConfigBase`

async classmethod from_existing(backend, **kwargs)

Creates a device config object from an existing backend

Keyword arguments will be passed to the `create()` method

Parameters backend – An instance of `vidhubcontrol.backends.base.BackendBase`

Returns An instance of `DeviceConfigBase`

2.2 vidhubcontrol.backends

2.2.1 vidhubcontrol.backends.base

class vidhubcontrol.backends.base.**BackendBase**(*args, **kwargs)

Bases: `pydispatch.dispatch.Dispatcher`

Base class for communicating with devices

Events

on_preset_added(backend: `BackendBase` = self, preset: `Preset` = preset)

This `Event` is emitted when a new `Preset` has been added.

on_preset_stored(backend: `BackendBase` = self, preset: `Preset` = preset)

This `Event` is emitted when an existing `Preset` has been recorded (updated).

on_preset_active(backend: `BackendBase`, preset: `Preset` = preset, value: `bool` = value)

This `Event` is emitted when an existing `Preset` has determined that its stored routing information is currently active on the switcher.

connection_manager: `vidhubcontrol.common.ConnectionManager`

Manager for the device’s `ConnectionState`

property connection_state: `vidhubcontrol.common.ConnectionState`

The current `state` of the `connection_manager`

device_id: `str`

The unique id as reported by the device

device_model: `str`

The model name as reported by the device

device_version: `str`

Firmware version reported by the device

```
class vidhubcontrol.backends.base.Preset(*args, **kwargs)
    Bases: pydispatch.dispatch.Dispatcher
    Stores and recalls routing information

    name
        The name of the preset. This is a pydispatch.Property
        Type str

    index
        The index of the preset as it is stored in the presets container.
        Type int

    crosspoints
        The crosspoints that this preset has stored. This is a DictProperty
        Type Dict[int, int]

    active
        A flag indicating whether all of the crosspoints stored in this preset are currently active on the switcher.
        This is a pydispatch.Property
        Type bool

    Events
        on_preset_stored(preset: Preset = self)
            Dispatched after the preset stores its state.

class vidhubcontrol.backends.base.SmartScopeBackendBase(*args, **kwargs)
    Bases: vidhubcontrol.backends.base.SmartViewBackendBase

class vidhubcontrol.backends.base.SmartScopeMonitor(*args, **kwargs)
    Bases: vidhubcontrol.backends.base.SmartViewMonitor
    A single instance of a monitor within a SmartScope device

    scope_mode
        The type of scope to display. Choices are: 'audio_dbfs', 'audio_dbvu', 'histogram', 'parade_rgb', 'parade_yuv', 'video', 'vector_100', 'vector_75', 'waveform'.
        Type str

class vidhubcontrol.backends.base.SmartViewBackendBase(*args, **kwargs)
    Bases: vidhubcontrol.backends.base.BackendBase
    Base class for SmartView devices

    num_monitors
        Number of physical monitors as reported by the device
        Type Optional[int]

    inverted
        True if the device has been mounted in an inverted configuration (to optimize viewing angle).
        Type bool

    monitors
        A list containing instances of SmartViewMonitor or SmartScopeMonitor, depending on device type.
        Type List[vidhubcontrol.backends.base.SmartViewMonitor]
```

Events

on_monitor_property_change(*self*: SmartViewBackendBase, *name*: str, *value*: Any, *monitor*: SmartViewMonitor = *monitor*)

Dispatched when any **Property** value changes. The event signature for callbacks is (smartview_device, property_name, value, **kwargs) containing a keyword argument “monitor” containing the **SmartViewMonitor** instance.

async set_monitor_property(*monitor*, *name*, *value*)

Set a property value for the given **SmartViewMonitor** instance

Parameters

- **monitor** – The **SmartViewMonitor** instance to set
- **name** (str) – Property name
- **value** – The new value to set

This method is a coroutine.

class vidhubcontrol.backends.base.**SmartViewMonitor**(*args, **kwargs)

Bases: **pydispatch.dispatch.Dispatcher**

A single instance of a monitor within a SmartView device

index

Index of the monitor (zero-based)

Type int

name

The name of the monitor (can be user-defined)

Type str

brightness

The brightness value of the monitor (0-255)

Type int

contrast

The contrast value of the monitor (0-255)

Type int

saturation

The saturation value of the monitor (0-255)

Type int

widescreen_sd

Aspect ratio setting for SD format. Choices can be: True (stretching enabled), False (pillar-box), or None (auto-detect).

Type Optional[bool]

identify

If set to True, the monitor’s border will be white for a brief duration to physically locate the device.

Type bool

border

Sets the border of the monitor to the given color. Choices are: ‘red’, ‘green’, ‘blue’, ‘white’, or None.

Type Optional[str]

audio_channel

The audio channel pair (Embedded in the SDI input) used when `scope_mode` is set to audio monitoring. Values are from 0 to 7 (0 == Channels 1&2, etc).

Type `int`

class `vidhubcontrol.backends.base.VidhubBackendBase(*args, **kwargs)`

Bases: `vidhubcontrol.backends.base.BackendBase`

Base class for Videohub devices

num_outputs

The number of outputs as reported by the switcher.

Type `int`

num_inputs

The number of inputs as reported by the switcher.

Type `int`

crosspoints

This represents the currently active routing of the switcher. Each element in the `list` represents an output (the zero-based index of the `list`) with its selected index as the value (also zero-based). This is a `pydispatch.properties.ListProperty` and can be observed using the `bind()` method.

Type `List[int]`

output_labels

A `list` containing the names of each output as reported by the switcher This is a `pydispatch.properties.ListProperty` and can be observed using the `bind()` method.

Type `List[str]`

input_labels

A `list` containing the names of each input as reported by the switcher This is a `pydispatch.properties.ListProperty` and can be observed using the `bind()` method.

Type `List[str]`

crosspoint_control

This is similar to `crosspoints` but if modified from outside code, the crosspoint changes will be set on the device (no method calls required). `pydispatch.properties.ListProperty`

Type `List[int]`

output_label_control

This is similar to `output_labels` but if modified from outside code, the label changes will be written to the device (no method calls required). `pydispatch.properties.ListProperty`

Type `List[str]`

input_label_control

This is similar to `input_labels` but if modified from outside code, the label changes will be written to the device (no method calls required). `pydispatch.properties.ListProperty`

Type `List[str]`

presets

The currently available (stored) `list` of `Preset` instances `pydispatch.properties.ListProperty`

Type `List[vidhubcontrol.backends.base.Preset]`

async add_preset(*name=None*)

Adds a new *Preset* instance

This method is used internally and should not normally be called outside of this module. Instead, see `store_preset()`

async set_crosspoint(*out_idx, in_idx*)

Set a single crosspoint on the switcher

Parameters

- **out_idx** (*int*) – The output to be set (zero-based)
- **in_idx** (*int*) – The input to switch the output (*out_idx*) to (zero-based)

async set_crosspoints(**args*)

Set multiple crosspoints in one method call

This is useful for setting many routing changes as it reduces the number of commands sent to the switcher.

Parameters **args* – Any number of output/input pairs to set. These should be given as tuples of (*out_idx*, *in_idx*) as defined in `set_crosspoint()`. They can be discontinuous and unordered.

async set_input_label(*in_idx, label*)

Set the label (name) of an input

Parameters

- **in_idx** (*int*) – The input to be set (zero-based)
- **label** (*str*) – The label for the input

async set_input_labels(**args*)

Set multiple input labels in one method call

This is useful for setting many labels as it reduces the number of commands sent to the switcher.

Parameters **args* – Any number of input/label pairs to set. These should be given as tuples of (*in_idx*, *label*) as defined in `set_input_label()`. They can be discontinuous and unordered.

async set_output_label(*out_idx, label*)

Set the label (name) of an output

Parameters

- **out_idx** (*int*) – The output to be set (zero-based)
- **label** (*str*) – The label for the output

async set_output_labels(**args*)

Set multiple output labels in one method call

This is useful for setting many labels as it reduces the number of commands sent to the switcher.

Parameters **args* – Any number of output/label pairs to set. These should be given as tuples of (*out_idx*, *label*) as defined in `set_output_label()`. They can be discontinuous and unordered.

async store_preset(*outputs_to_store=None, name=None, index=None, clear_current=True*)

Store the current switcher state to a *Preset*

Parameters

- **outputs_to_store** (*optional*) – An iterable of the output numbers (zero-based) that should be saved in the preset. If given, only these outputs will be recorded and when recalled, any output not in this argument will be unchanged. If not given or `None`, all outputs will be recorded.
- **name** (*optional*) – The name to be given to the preset. If not provided or `None` the preset will be given a name based off of its index.
- **index** (*optional*) – The index for the preset. If given and the preset exists in the `presets` list, that preset will be updated. If there is no preset found with the index, a new one will be created. If not given or `None`, the next available index will be used and a new preset will be created.
- **clear_current** (*bool*) – If `True`, any previously existing data will be removed from the preset (if it exists). If `False`, the data (if existing) will be merged with the current switcher state. Default is `True`

Returns The `Preset` instance that was created or updated

This method is a coroutine

2.2.2 vidhubcontrol.backends.telnet

```
class vidhubcontrol.backends.telnet.SmartScopeTelnetBackend(*args, **kwargs)
    Bases: vidhubcontrol.backends.telnet.SmartViewTelnetBackendBase, vidhubcontrol.
           backends.base.SmartScopeBackendBase
```

```
class vidhubcontrol.backends.telnet.SmartViewTelnetBackend(*args, **kwargs)
    Bases: vidhubcontrol.backends.telnet.SmartViewTelnetBackendBase, vidhubcontrol.
           backends.base.SmartViewBackendBase
```

```
class vidhubcontrol.backends.telnet.SmartViewTelnetBackendBase
    Bases: vidhubcontrol.backends.telnet.TelnetBackendBase
```

```
class vidhubcontrol.backends.telnet.TelnetBackend(*args, **kwargs)
    Bases: vidhubcontrol.backends.telnet.TelnetBackendBase, vidhubcontrol.backends.base.
           VidhubBackendBase
```

Base class for backends implementing telnet

```
async set_crosspoint(out_idx, in_idx)
    Set a single crosspoint on the switcher
```

Parameters

- **out_idx** (*int*) – The output to be set (zero-based)
- **in_idx** (*int*) – The input to switch the output (out_idx) to (zero-based)

```
async set_crosspoints(*args)
    Set multiple crosspoints in one method call
```

This is useful for setting many routing changes as it reduces the number of commands sent to the switcher.

Parameters **args* – Any number of output/input pairs to set. These should be given as tuples of (out_idx, in_idx) as defined in `set_crosspoint()`. They can be discontinuous and unordered.

```
async set_input_label(in_idx, label)
    Set the label (name) of an input
```

Parameters

- **in_idx** (*int*) – The input to be set (zero-based)
- **label** (*str*) – The label for the input

async set_input_labels(*args)

Set multiple input labels in one method call

This is useful for setting many labels as it reduces the number of commands sent to the switcher.

Parameters *args – Any number of input/label pairs to set. These should be given as tuples of (in_idx, label) as defined in set_input_label(). They can be discontinuous and unordered.

async set_output_label(out_idx, label)

Set the label (name) of an output

Parameters

- **out_idx** (*int*) – The output to be set (zero-based)
- **label** (*str*) – The label for the output

async set_output_labels(*args)

Set multiple output labels in one method call

This is useful for setting many labels as it reduces the number of commands sent to the switcher.

Parameters *args – Any number of output/label pairs to set. These should be given as tuples of (out_idx, label) as defined in set_output_label(). They can be discontinuous and unordered.

class vidhubcontrol.backends.telnet.TelnetBackendBase

Bases: *object*

Mix-in class for backends implementing telnet

hostaddr

IPv4 address of the device

Type *str*

hostport

Port address of the device

Type *int*

read_enabled

Internal flag to keep the read_loop() running

Type *bool*

rx_bfr

Data received from the device to be parsed

Type *bytes*

client

Instance of vidhubcontrol.aiotelnetlib._Telnet

Type vidhubcontrol.aiotelnetlib._Telnet

2.3 vidhubcontrol.discovery

class vidhubcontrol.discovery.AddedMessage(*info*: vidhubcontrol.discovery.ServiceInfo)

Bases: vidhubcontrol.discovery.BrowserMessage

class vidhubcontrol.discovery.BMDDiscovery(*args, **kwargs)

Bases: vidhubcontrol.discovery.Listener

Zeroconf listener for Blackmagic devices

vidhubs

Contains discovered Videohub devices. This DictProperty can be used to subscribe to changes.

Type Dict[str, vidhubcontrol.discovery.ServiceInfo]

smart_views

Contains discovered SmartView devices. This DictProperty can be used to subscribe to changes.

Type Dict[str, vidhubcontrol.discovery.ServiceInfo]

smart_scopes

Contains discovered SmartScope devices. This DictProperty can be used to subscribe to changes.

Type Dict[str, vidhubcontrol.discovery.ServiceInfo]

class vidhubcontrol.discovery.BrowserMessage(*info*: vidhubcontrol.discovery.ServiceInfo)

Bases: vidhubcontrol.discovery.Message

class vidhubcontrol.discovery.Listener(*args, **kwargs)

Bases: pydispatch.dispatch.Dispatcher

An async zeroconf service listener

Allows async communication with zeroconf.Zeroconf through asyncio.AbstractEventLoop.run_in_executor() calls.

Parameters

- **mainloop** (asyncio.BaseEventLoop) – asyncio event loop instance
- **service_type** (*str*) – The fully qualified service type name to subscribe to

services

All services currently discovered as instances of ServiceInfo. Stored using ServiceInfo.id as keys

Type Dict[str, vidhubcontrol.discovery.ServiceInfo]

message_queue

Used to communicate actions and events with instances of Message

Type asyncio.queue.Queue

published_services

Stores services that have been published using publish_service() as ServiceInfo instances.

Type Dict[str, vidhubcontrol.discovery.ServiceInfo]

async add_message(*msg*: vidhubcontrol.discovery.Message)

Adds a message to the message_queue

Parameters *msg* (Message) – Message to send

async publish_service(*type_*: *str*, *port*: *int*, *name*: *Optional[str] = None*, *addresses*: *Optional[Union[str, bytes, ipaddress.IPv4Address]] = None*, *properties*: *Optional[Dict] = None*, *ttl*: *Optional[int] = 60*)

Publishes a service on the network

Parameters

- **type** (*str*) – Fully qualified service type
- **port** (*int*) – The service port
- **name** (*str*, *optional*) – Fully qualified service name. If not provided, this will be generated from the *type_* and the hostname detected by `get_local_hostname()`
- **addresses** (*optional*) – If provided, an iterable of IP addresses to publish. Can be `ipaddress.IPv4Address` or any type that can be parsed by `ipaddress.ip_address()`
- **properties** (*dict*, *optional*) – Custom properties for the service
- **ttl** (*int*, *optional*) – The TTL value to publish. Defaults to `PUBLISH_TTL`

async republish_service(*type_*: *str*, *port*: *int*, *name*: *Optional[str] = None*, *addresses*: *Optional[Union[str, bytes, ipaddress.IPv4Address]] = None*, *properties*: *Optional[Dict] = None*, *ttl*: *Optional[int] = 60*)

Update an existing `ServiceInfo` and republish it

async run()

Main loop for communicating with `zeroconf.Zeroconf`

Waits for messages on the `message_queue` and processes them. The loop will exit if an object placed on the queue is not an instance of `Message`.

run_zeroconf()

Starts `zeroconf.Zeroconf` and `zeroconf.ServiceBrowser` instances

async start()

Starts the service listener

async stop()

Stops the service listener

async stop_zeroconf()

Closes the `zeroconf.Zeroconf` instance

async unpublish_service(*type_*: *str*, *name*: *Optional[str] = None*)

Removes a service published through `publish_service()`

Parameters

- **type** (*str*) – Fully qualified service type
- **name** (*str*, *optional*) – Fully qualified service name. If not provided, this will be generated from the *type_* and the hostname detected by `get_local_hostname()`

class vidhubcontrol.discovery.Message(*info*: `vidhubcontrol.discovery.ServiceInfo`)

Bases: `object`

A message to communicate actions to and from `Listener`

info

The `ServiceInfo` related to the message

Type `vidhubcontrol.discovery.ServiceInfo`

Note: This class and its subclasses are not meant to be used directly. They are used internally in [Listener](#) methods.

```
class vidhubcontrol.discovery.PublishMessage(info: vidhubcontrol.discovery.ServiceInfo)
    Bases: vidhubcontrol.discovery.RegistrationMessage

class vidhubcontrol.discovery.RegistrationMessage(info: vidhubcontrol.discovery.ServiceInfo)
    Bases: vidhubcontrol.discovery.Message

class vidhubcontrol.discovery.RemovedMessage(info: vidhubcontrol.discovery.ServiceInfo)
    Bases: vidhubcontrol.discovery.BrowserMessage

class vidhubcontrol.discovery.RepublishMessage(info: vidhubcontrol.discovery.ServiceInfo)
    Bases: vidhubcontrol.discovery.RegistrationMessage

class vidhubcontrol.discovery.ServiceInfo(*args, **kwargs)
    Bases: pydispatch.dispatch.Dispatcher

    Container for Zeroconf service information
    Closely related to zeroconf.ServiceInfo

    type
        Fully qualified service type
        Type str

    name
        Fully qualified service name
        Type str

    server
        Fully qualified name for service host (defaults to name)
        Type str

    addresses
        The service ip address

    port
        The service port
        Type int

    properties
        Custom properties for the service
        Type Dict[str, str]

    property address: Optional[ipaddress.IPv4Address]
        The first element of addresses

    classmethod from_zc_info(info: zeroconf._services.info.ServiceInfo) →
        vidhubcontrol.discovery.ServiceInfo
        Creates an instance from a zeroconf.ServiceInfo object

        Parameters info (zeroconf.ServiceInfo) –

        Returns An instance of ServiceInfo

    property id: Tuple[str, str]
        Unique id for the service as a tuple of (type, name)
```

to_zc_info() → `zeroconf._services.info.ServiceInfo`
 Creates a copy as an instance of `zeroconf.ServiceInfo`

update(*other*: `vidhubcontrol.discovery.ServiceInfo`)
 Updates the *properties* from another *ServiceInfo* instance

class `vidhubcontrol.discovery.UnPublishMessage`(*info*: `vidhubcontrol.discovery.ServiceInfo`)
 Bases: `vidhubcontrol.discovery.RegistrationMessage`

class `vidhubcontrol.discovery.UpdateMessage`(*info*: `vidhubcontrol.discovery.ServiceInfo`)
 Bases: `vidhubcontrol.discovery.BrowserMessage`

2.4 vidhubcontrol.common

class `vidhubcontrol.common.ConnectionManager`(*args, **kwargs)
 Bases: `pydispatch.dispatch.Dispatcher`

A manager for tracking and waiting for *connection states*

A `asyncio.Condition` is used to to notify any waiting tasks of changes to *state*. This requires the underlying lock to be *acquired* before calling any of the waiter or setter methods and *released* afterwards.

This class supports the asynchronous context manager protocol for use in `async with` statements.

Events

state_changed(*self*: `ConnectionManager`, *state*: `ConnectionState`)
 Emitted when the value of *state* has changed

async acquire()
 Acquire the lock

This method blocks until the lock is unlocked, then sets it to locked and returns True.

failure_exception: `Optional[Exception]`
 The `Exception` raised if an error occurred

failure_reason: `Optional[str]`
 A message describing errors (if encountered)

locked() → `bool`
 True if the lock is acquired

release()
 Release the lock

Raises `RuntimeError` – if called on an unlocked lock

async set_failure(*reason*: *Any*, *exc*: `Optional[Exception] = None`, *state*:
`Optional[Union[vidhubcontrol.common.ConnectionState, str]] =`
`ConnectionState.None`)

Set *state* to indicate a failure

Parameters

- **reason** – A description of the failure
- **exc** – The `Exception` that caused the failure (if available)
- **state** – The new state to set. Must include `ConnectionState.failure`

Raises `RuntimeError` – If the lock is not *acquired* before calling this method

async set_state(*state*: Union[vidhubcontrol.common.ConnectionState, str])

Set the *state* to the given value

The *state* argument may be either a *ConnectionState* member or a string. (see *ConnectionState.from_str()*)

Raises *RuntimeError* – If the lock is not *acquired* before calling this method

property state: vidhubcontrol.common.ConnectionState

The current state

async synchronize(*other*: vidhubcontrol.common.ConnectionManager)

Copy the *state* and failure values of another *ConnectionManager*

Note: The lock must **not** be acquired before calling this method.

async wait(*timeout*: Optional[float] = None) → vidhubcontrol.common.ConnectionState

Block until the next time *state* changes and return the value

Parameters *timeout* – If given, the number of seconds to wait. Otherwise, this will wait indefinitely

Raises

- *asyncio.TimeoutError* – If *timeout* is given and no state changes occurred
- *RuntimeError* – If the lock is not *acquired* before calling this method

async wait_for(*state*: Union[vidhubcontrol.common.ConnectionState, str], *timeout*: Optional[float] = None) → vidhubcontrol.common.ConnectionState

Wait for a specific state

The *state* argument may be a *ConnectionState* member or string as described in *ConnectionState.from_str()*.

If the given state is *compound* or the *state* is set as compound, this will wait until all members from the *state* argument are contained within the *state* value.

Parameters

- *state* – The state to wait for
- *timeout* – If given, the number of seconds to wait. Otherwise, this will wait indefinitely

Raises

- *asyncio.TimeoutError* – If *timeout* is given and no matching state changes were found
- *RuntimeError* – If the lock is not *acquired* before calling this method

async wait_for_disconnected(*timeout*: Optional[float] = None) → vidhubcontrol.common.ConnectionState

Wait for *ConnectionState.not_connected*

Parameters *timeout* – If given, the number of seconds to wait. Otherwise, this will wait indefinitely

Raises

- *asyncio.TimeoutError* – If *timeout* is given and no matching state changes were found
- *RuntimeError* – If the lock is not *acquired* before calling this method

async wait_for_established(*timeout: Optional[float] = None*) →

vidhubcontrol.common.ConnectionState

Wait for either a success (*ConnectionState.connected*) or failure (*ConnectionState.failure*)

Parameters **timeout** – If given, the number of seconds to wait. Otherwise, this will wait indefinitely

Raises

- **asyncio.TimeoutError** – If *timeout* is given and no matching state changes were found
- **RuntimeError** – If the lock is not *acquired* before calling this method

class *vidhubcontrol.common.ConnectionState*(*value*)

Bases: *enum.IntFlag*

Enum to describe various connection states

Members may be combined using bitwise operators (&, |, ^, ~)

connected = 8

Indicates the connection is active

connecting = 2

Indicates an attempt to connect is being made

disconnecting = 4

Indicates the connection is being closed

failure = 16

Indicates an error occurred

classmethod **from_str**(*s: str*) → *vidhubcontrol.common.ConnectionState*

Create a *ConnectionState* member by name(s)

Combined states can be created by separating their names with a “|”

```
>>> from vidhubcontrol.common import ConnectionState
>>> ConnectionState.connected | ConnectionState.not_connected
<ConnectionState.connected|not_connected: 9>
>>> ConnectionState.disconnecting | ConnectionState.failure
<ConnectionState.failure|disconnecting: 20>
>>> # This combination is already defined as "waiting"
>>> ConnectionState.connecting | ConnectionState.disconnecting
<ConnectionState.waiting: 6>
```

property **is_compound**: **bool**

This will evaluate to True for states combined using bitwise operators

property **is_connected**: **bool**

Convenience property evaluating as True if *self* == *ConnectionState.connected*

not_connected = 1

Indicates there is no connection and no connection attempts are being made

waiting = 6

Indicates the connection is either *connecting* or *disconnecting*

class *vidhubcontrol.common.SynchronizedConnectionManager*(*args, **kwargs)

Bases: *vidhubcontrol.common.ConnectionManager*

A connection manager that synchronizes itself with another

property other: Optional[[vidhubcontrol.common.ConnectionManager](#)]

The manager currently being synchronized to

async set_other(*other*: Optional[[vidhubcontrol.common.ConnectionManager](#)])

Set the manager to synchronize with

This binds to the `state_changed()` event of *other* and calls the [synchronize\(\)](#) method whenever the state of the other manager changes.

If None is given, *state* is set to [not_connected](#)

Note: The lock must *not* be acquired before calling this method

INDICES AND TABLES

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

PYTHON MODULE INDEX

V

- `vidhubcontrol.backends.base`, 9
- `vidhubcontrol.backends.telnet`, 14
- `vidhubcontrol.common`, 19
- `vidhubcontrol.config`, 5
- `vidhubcontrol.discovery`, 16

INDEX

A

`acquire()` (*vidhubcontrol.common.ConnectionManager* method), 19
`active` (*vidhubcontrol.backends.base.Preset* attribute), 10
`add_device()` (*vidhubcontrol.config.Config* method), 6
`add_message()` (*vidhubcontrol.discovery.Listener* method), 16
`add_preset()` (*vidhubcontrol.backends.base.VidhubBackendBase* method), 12
`AddedMessage` (class in *vidhubcontrol.discovery*), 16
`address` (*vidhubcontrol.discovery.ServiceInfo* property), 18
`addresses` (*vidhubcontrol.discovery.ServiceInfo* attribute), 18
`all_devices` (*vidhubcontrol.config.Config* attribute), 6
`audio_channel` (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), 11

B

`backend` (*vidhubcontrol.config.DeviceConfigBase* attribute), 7
`backend_name` (*vidhubcontrol.config.DeviceConfigBase* attribute), 7
`BackendBase` (class in *vidhubcontrol.backends.base*), 9
`BackendBase.on_preset_active()` (in module *vidhubcontrol.backends.base*), 9
`BackendBase.on_preset_added()` (in module *vidhubcontrol.backends.base*), 9
`BackendBase.on_preset_stored()` (in module *vidhubcontrol.backends.base*), 9
`BMDDiscovery` (class in *vidhubcontrol.discovery*), 16
`border` (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), 11
`brightness` (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), 11
`BrowserMessage` (class in *vidhubcontrol.discovery*), 16
`build_backend()` (*vidhubcontrol.config.Config* method), 6

`build_backend()` (*vidhubcontrol.config.DeviceConfigBase* method), 7
`build_backend()` (*vidhubcontrol.config.VidhubConfig* method), 8

C

`client` (*vidhubcontrol.backends.telnet.TelnetBackendBase* attribute), 15
`Config` (class in *vidhubcontrol.config*), 5
`config` (*vidhubcontrol.config.DeviceConfigBase* attribute), 7
`connected` (*vidhubcontrol.common.ConnectionState* attribute), 21
`connecting` (*vidhubcontrol.common.ConnectionState* attribute), 21
`connection_manager` (*vidhubcontrol.backends.base.BackendBase* attribute), 9
`connection_manager` (*vidhubcontrol.config.Config* attribute), 6
`connection_manager` (*vidhubcontrol.config.DeviceConfigBase* attribute), 8
`connection_state` (*vidhubcontrol.backends.base.BackendBase* property), 9
`connection_state` (*vidhubcontrol.config.Config* property), 6
`connection_state` (*vidhubcontrol.config.DeviceConfigBase* property), 8
`ConnectionManager` (class in *vidhubcontrol.common*), 19
`ConnectionManager.state_changed()` (in module *vidhubcontrol.common*), 19
`ConnectionState` (class in *vidhubcontrol.common*), 21
`contrast` (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), 11
`create()` (*vidhubcontrol.config.DeviceConfigBase* class method), 8
`create()` (*vidhubcontrol.config.VidhubConfig* class method), 9

`crosspoint_control` (*vidhubcontrol.backends.base.VidhubBackendBase attribute*), 12

`crosspoints` (*vidhubcontrol.backends.base.Preset attribute*), 10

`crosspoints` (*vidhubcontrol.backends.base.VidhubBackendBase attribute*), 12

D

`DEFAULT_FILENAME` (*vidhubcontrol.config.Config attribute*), 6

`device_id` (*vidhubcontrol.backends.base.BackendBase attribute*), 9

`device_id` (*vidhubcontrol.config.DeviceConfigBase attribute*), 7

`device_model` (*vidhubcontrol.backends.base.BackendBase attribute*), 9

`device_name` (*vidhubcontrol.config.DeviceConfigBase attribute*), 7

`device_version` (*vidhubcontrol.backends.base.BackendBase attribute*), 9

`DeviceConfigBase` (*class in vidhubcontrol.config*), 7

`disconnecting` (*vidhubcontrol.common.ConnectionState attribute*), 21

F

`failure` (*vidhubcontrol.common.ConnectionState attribute*), 21

`failure_exception` (*vidhubcontrol.common.ConnectionManager attribute*), 19

`failure_reason` (*vidhubcontrol.common.ConnectionManager attribute*), 19

`from_existing()` (*vidhubcontrol.config.DeviceConfigBase class method*), 8

`from_existing()` (*vidhubcontrol.config.VidhubConfig class method*), 9

`from_str()` (*vidhubcontrol.common.ConnectionState class method*), 21

`from_zc_info()` (*vidhubcontrol.discovery.ServiceInfo class method*), 18

H

`hostaddr` (*vidhubcontrol.backends.telnet.TelnetBackendBase attribute*), 15

`hostaddr` (*vidhubcontrol.config.DeviceConfigBase attribute*), 7

`hostport` (*vidhubcontrol.backends.telnet.TelnetBackendBase attribute*), 15

`hostport` (*vidhubcontrol.config.DeviceConfigBase attribute*), 7

I

`id` (*vidhubcontrol.discovery.ServiceInfo property*), 18

`identify` (*vidhubcontrol.backends.base.SmartViewMonitor attribute*), 11

`index` (*vidhubcontrol.backends.base.Preset attribute*), 10

`index` (*vidhubcontrol.backends.base.SmartViewMonitor attribute*), 11

`info` (*vidhubcontrol.discovery.Message attribute*), 17

`input_label_control` (*vidhubcontrol.backends.base.VidhubBackendBase attribute*), 12

`input_labels` (*vidhubcontrol.backends.base.VidhubBackendBase attribute*), 12

`inverted` (*vidhubcontrol.backends.base.SmartViewBackendBase attribute*), 10

`is_compound` (*vidhubcontrol.common.ConnectionState property*), 21

`is_connected` (*vidhubcontrol.common.ConnectionState property*), 21

L

`Listener` (*class in vidhubcontrol.discovery*), 16

`load()` (*vidhubcontrol.config.Config class method*), 6

`load_async()` (*vidhubcontrol.config.Config class method*), 6

`locked()` (*vidhubcontrol.common.ConnectionManager method*), 19

M

`Message` (*class in vidhubcontrol.discovery*), 17

`message_queue` (*vidhubcontrol.discovery.Listener attribute*), 16

`module`

`vidhubcontrol.backends.base`, 9

`vidhubcontrol.backends.telnet`, 14

`vidhubcontrol.common`, 19

`vidhubcontrol.config`, 5

`vidhubcontrol.discovery`, 16

`monitors` (*vidhubcontrol.backends.base.SmartViewBackendBase attribute*), 10

N

`name` (*vidhubcontrol.backends.base.Preset attribute*), 10

`name` (*vidhubcontrol.backends.base.SmartViewMonitor attribute*), 11

`name` (*vidhubcontrol.discovery.ServiceInfo attribute*), 18

not_connected (*vidhubcontrol.common.ConnectionState* attribute), 21
num_inputs (*vidhubcontrol.backends.base.VidhubBackendBase* attribute), 12
num_monitors (*vidhubcontrol.backends.base.SmartViewBackendBase* attribute), 10
num_outputs (*vidhubcontrol.backends.base.VidhubBackendBase* attribute), 12
O
other (*vidhubcontrol.common.SynchronizedConnectionManager* property), 21
output_label_control (*vidhubcontrol.backends.base.VidhubBackendBase* attribute), 12
output_labels (*vidhubcontrol.backends.base.VidhubBackendBase* attribute), 12
P
port (*vidhubcontrol.discovery.ServiceInfo* attribute), 18
Preset (class in *vidhubcontrol.backends.base*), 9
Preset.on_preset_stored() (in module *vidhubcontrol.backends.base*), 10
presets (*vidhubcontrol.backends.base.VidhubBackendBase* attribute), 12
presets (*vidhubcontrol.config.VidhubConfig* attribute), 8
properties (*vidhubcontrol.discovery.ServiceInfo* attribute), 18
publish_service() (*vidhubcontrol.discovery.Listener* method), 16
published_services (*vidhubcontrol.discovery.Listener* attribute), 16
PublishMessage (class in *vidhubcontrol.discovery*), 18
R
read_enabled (*vidhubcontrol.backends.telnet.TelnetBackendBase* attribute), 15
RegistrationMessage (class in *vidhubcontrol.discovery*), 18
release() (*vidhubcontrol.common.ConnectionManager* method), 19
RemovedMessage (class in *vidhubcontrol.discovery*), 18
republish_service() (*vidhubcontrol.discovery.Listener* method), 17
RepublishMessage (class in *vidhubcontrol.discovery*), 18
run() (*vidhubcontrol.discovery.Listener* method), 17
run_zeroconf() (*vidhubcontrol.discovery.Listener* method), 17
rx_bfr (*vidhubcontrol.backends.telnet.TelnetBackendBase* attribute), 15
S
saturation (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), 11
save() (*vidhubcontrol.config.Config* method), 7
scope_mode (*vidhubcontrol.backends.base.SmartScopeMonitor* attribute), 10
server (*vidhubcontrol.discovery.ServiceInfo* attribute), 18
ServiceInfo (class in *vidhubcontrol.discovery*), 18
services (*vidhubcontrol.discovery.Listener* attribute), 16
set_crosspoint() (*vidhubcontrol.backends.base.VidhubBackendBase* method), 13
set_crosspoint() (*vidhubcontrol.backends.telnet.TelnetBackend* method), 14
set_crosspoints() (*vidhubcontrol.backends.base.VidhubBackendBase* method), 13
set_crosspoints() (*vidhubcontrol.backends.telnet.TelnetBackend* method), 14
set_failure() (*vidhubcontrol.common.ConnectionManager* method), 19
set_input_label() (*vidhubcontrol.backends.base.VidhubBackendBase* method), 13
set_input_label() (*vidhubcontrol.backends.telnet.TelnetBackend* method), 14
set_input_labels() (*vidhubcontrol.backends.base.VidhubBackendBase* method), 13
set_input_labels() (*vidhubcontrol.backends.telnet.TelnetBackend* method), 15
set_monitor_property() (*vidhubcontrol.backends.base.SmartViewBackendBase* method), 11
set_other() (*vidhubcontrol.common.SynchronizedConnectionManager* method), 22
set_output_label() (*vidhubcontrol.backends.base.VidhubBackendBase* method), 13

[set_output_label\(\)](#) (*vidhubcontrol.backends.telnet.TelnetBackend* method), [15](#)
[set_output_labels\(\)](#) (*vidhubcontrol.backends.base.VidhubBackendBase* method), [13](#)
[set_output_labels\(\)](#) (*vidhubcontrol.backends.telnet.TelnetBackend* method), [15](#)
[set_state\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [19](#)
[smart_scopes](#) (*vidhubcontrol.discovery.BMDDiscovery* attribute), [16](#)
[smart_views](#) (*vidhubcontrol.discovery.BMDDiscovery* attribute), [16](#)
[SmartScopeBackendBase](#) (class in *vidhubcontrol.backends.base*), [10](#)
[SmartScopeConfig](#) (class in *vidhubcontrol.config*), [8](#)
[SmartScopeMonitor](#) (class in *vidhubcontrol.backends.base*), [10](#)
[smartscores](#) (*vidhubcontrol.config.Config* attribute), [6](#)
[SmartScopeTelnetBackend](#) (class in *vidhubcontrol.backends.telnet*), [14](#)
[SmartViewBackendBase](#) (class in *vidhubcontrol.backends.base*), [10](#)
[SmartViewBackendBase.on_monitor_property_change\(\)](#) (in module *vidhubcontrol.backends.base*), [11](#)
[SmartViewConfig](#) (class in *vidhubcontrol.config*), [8](#)
[SmartViewMonitor](#) (class in *vidhubcontrol.backends.base*), [11](#)
[smartviews](#) (*vidhubcontrol.config.Config* attribute), [5](#)
[SmartViewTelnetBackend](#) (class in *vidhubcontrol.backends.telnet*), [14](#)
[SmartViewTelnetBackendBase](#) (class in *vidhubcontrol.backends.telnet*), [14](#)
[start\(\)](#) (*vidhubcontrol.config.Config* method), [7](#)
[start\(\)](#) (*vidhubcontrol.discovery.Listener* method), [17](#)
[state](#) (*vidhubcontrol.common.ConnectionManager* property), [20](#)
[stop\(\)](#) (*vidhubcontrol.config.Config* method), [7](#)
[stop\(\)](#) (*vidhubcontrol.discovery.Listener* method), [17](#)
[stop_zeroconf\(\)](#) (*vidhubcontrol.discovery.Listener* method), [17](#)
[store_preset\(\)](#) (*vidhubcontrol.backends.base.VidhubBackendBase* method), [13](#)
[synchronize\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [20](#)
[SynchronizedConnectionManager](#) (class in *vidhubcontrol.common*), [21](#)

T
[TelnetBackend](#) (class in *vidhubcontrol.backends.telnet*), [14](#)
[TelnetBackendBase](#) (class in *vidhubcontrol.backends.telnet*), [15](#)
[to_zc_info\(\)](#) (*vidhubcontrol.discovery.ServiceInfo* method), [18](#)
[type](#) (*vidhubcontrol.discovery.ServiceInfo* attribute), [18](#)

U
[unpublish_service\(\)](#) (*vidhubcontrol.discovery.Listener* method), [17](#)
[UnPublishMessage](#) (class in *vidhubcontrol.discovery*), [19](#)
[update\(\)](#) (*vidhubcontrol.discovery.ServiceInfo* method), [19](#)
[UpdateMessage](#) (class in *vidhubcontrol.discovery*), [19](#)

V
[VidhubBackendBase](#) (class in *vidhubcontrol.backends.base*), [12](#)
[VidhubConfig](#) (class in *vidhubcontrol.config*), [8](#)
[vidhubcontrol.backends.base](#) module, [9](#)
[vidhubcontrol.backends.telnet](#) module, [14](#)
[vidhubcontrol.common](#) module, [19](#)
[vidhubcontrol.config](#) module, [5](#)
[vidhubcontrol.discovery](#) module, [16](#)
[vidhubs](#) (*vidhubcontrol.config.Config* attribute), [5](#)
[vidhubs](#) (*vidhubcontrol.discovery.BMDDiscovery* attribute), [16](#)

W
[wait\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [20](#)
[wait_for\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [20](#)
[wait_for_disconnected\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [20](#)
[wait_for_established\(\)](#) (*vidhubcontrol.common.ConnectionManager* method), [20](#)
[waiting](#) (*vidhubcontrol.common.ConnectionState* attribute), [21](#)
[widescreen_sd](#) (*vidhubcontrol.backends.base.SmartViewMonitor* attribute), [11](#)