PyOpenVidu Documentation

Release 0.2.1

Marcell Pünkösd

Mar 10, 2022

Contents:

CHAPTER 1

PyOpenVidu

Python interface to the OpenVidu WebRTC videoconference framework.

- Free software: MIT license
- Documentation: https://pyopenvidu.readthedocs.io.
- Flask extension: https://pypi.org/project/flask-openvidu/

1.1 Features

- Use OpenVidu API objects as native Python objects
- Supports Python 3.7 and above
- Depends on nothing more than *requests* and *requests-toolbelt*

1.2 Credits

This implementation is inspired by the original Java implementation by the OpenVidu team. https://openvidu.io/docs/reference-docs/openvidu-java-client/

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install PyOpenVidu, run this command in your terminal:

```
$ pip3 install pyopenvidu
```

This is the preferred method to install PyOpenVidu, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for PyOpenVidu can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/marcsello/pyopenvidu
```

Or download the tarball:

```
$ curl -OJL https://github.com/marcsello/pyopenvidu/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python3 setup.py install
```

CHAPTER 3

Examples

To use PyOpenVidu in a project, first import it:

```
from pyopenvidu import OpenVidu
```

Create a session:

```
openvidu = OpenVidu(OPENVIDU_URL, OPENVIDU_SECRET)
session = openvidu.create_session()
```

Generate a token a session:

```
token = session.create_webrtc_connection().token
```

Fetch information:

```
# Fetch all session info from OpenVidu Server
openvidu.fetch()
sessions = openvidu.sessions  # sessions returns a list of OpenViduSession objects

# Fetch a specific session info from the server
session.fetch()
connections = session.connections

# Fetch a specific connection info from the server
connection.fetch()
subs = connection.subscribers
```

Send signals:

```
# Broadcast signal to session
session.signal("MY_TYPE", "Hello world!")

# Send to a specific connection
session.get_connection("vhdxz7abbfirh2lh").signal("MY_TYPE", "Hello other world!")
```

(continues on next page)

(continued from previous page)

```
# Send to every other connection
# Note: This does not make any subsequent API calls, as the connections information␣
↪is already stored in memory
session.signal("MY_TYPE", "Yolo world!", [conn for i, conn in enumerate(session.
↪connections) if i % 2 == 0])
```

Connect to IP camera:

```
session.create_ipcam_connection("rtsp://mydomain.net:1935/live/stream")
```

Close a session:

```
session.close()
```

Force disconnect users:

```
# Disconnect a specific user
# None: Don't forget to call session.fetch() to work with the updated list of clients
session.get_connection("vhdxz7abbfirh2lh").force_disconnect()

# Disconnect every other user
# Note: This does not make any subsequent API calls, as the connections information␣
↪is already stored in memory
for i, conn in enumerate(session.connections):
    if i % 2 == 0:
        conn.force_disconnect()

# session.fetch() Should be called about here.
```

Force unpublish an user's streams:

```
# Unpublish a single stream (most of the time there is only one, except when sharing␣
↪screen):
session.get_connection("vhdxz7abbfirh2lh").publishers[0].force_unpublish()

# Unpublish all streams of an user:
session.get_connection("vhdxz7abbfirh2lh").force_unpublish_all_streams()
```

CHAPTER 4

About fetching...

Warning: In PyOpenVidu version 0.2.0 the dynamic and static objects are changed. See *Migrating* for more information.

This document contains updated information.

In PyOpenVidu there are two kind of objects: dynamic and static. This section aim to explain the details about this design.

4.1 Dynamic and static objects

The **dynamic objects** implement a *fetch()* method that can be used to update it's internal representation.

Table 1: Dynamic objects, and methods that change their internal representation

Object	Method	Notes
OpenVidu	*__init__(...)*	Upon instantiating this object makes a call to *self.fetch()* to collect initial data. (if not disabled)
OpenVidu	*fetch()*	Updates the internal representation of the that single **OpenVidu** object.
OpenViduSession	*fetch()*	Updates the internal representation of that single **OpenViduSession** object.
OpenViduConnection	*fetch()*	Updates the internal representation of that single **OpenViduConnection** object.

Please note, that although *OpenViduSession.create_webrtc_connection(...)*, *OpenViduSession.create_ipcam_connection(...)* and *OpenVidu.create_session(...)* returns with a new connection or session object, which is even begging added to the appropriate list. It does not update the internal representation of it's parent

(*OpenViduSession.fetch()* or *OpenVidu.fetch()* must be called to update the info of other connections). The reason for this is that the API returns the full object, so a fetch() int the background is not required.

Static objects are not designed to update their internal representation, thus not implementing a *fetch()* method. Such objects should not be reused at all, and must be considered invalid after any changes made to them by other calls. A new version of those objects could be requested by calling the *fetch()* method of the dynamic object that provides them.

Table 2: Static objects and methods that turns them, or their parent invalid

Object	Method	Notes
OpenViduPublisher	*force_unpublish()*	Affects other **OpenViduConnection** objects because subscriptions may change as well.
OpenViduSubscriber		

4.2 Reasons

The reason behind this architecture is mainly comes from the fact, that the original Java library, that this library took inspiration from uses the same solution.

Because the OpenVidu server does not expose other endpoints than *GET sessions*, *GET sessions/<SESSION_ID>* and *GET sessions/<SESSION_ID>/connections/<CONNECTION_ID>* by making every object "dynamic" would cause a lot of network overhead by transfering the unwanted information (e.g.: updating a Subscriber object would cause downloading all session data and using only a fraction of it).

Another approach would be to not have an internal representation at all, meaning that every method of every object would cause an API call in the background. This would be a bad idea for the following reasons:

- The server exposes only the few endpoint mentioned above, causing an excessive overhead.
- Accessing to multiple properties of an object would cause an equal amount of API calls (mostly to the same endpoint in our case) instead of one or zero calls when instantiating the object.
- Error handling would be more complicated. The programmer who uses this library must handle HTTP communication errors when accessing to properties as well as deal with the possibility that the represented object changes between API calls (e.g.: The user disconnects between accessing to *created_at* and *role* properties).

CHAPTER 5

Advanced Usage

5.1 Initial Fetching

It is possible, to disable fetching upon the creation of the *OpenVidu* object. This can be achieved by setting the *initial_fetch* attribute to *False* when creating the object. Doing so will cause the newly created *OpenVidu* object to not know anything about the state of the *OpenVidu* server. Not even knowing if it's possible to connect to it.

Without fetching it is still possible to use the following methods:

- *create_session()*: No state information is required to start a new session (existance of the session is validated by the server).
- *get_config()*: The configuration of the server is not stored at all, so this call will always request this from the server.
- *fetch()*: Well, that one is kind of obvious.

If you are confident that's something you need. You can use this to reduce the number of requests in certain cases. Here are som

- You are making something, that calls *fetch()* before every operation anyways. This would make an initial fetch pointless, so you can disable it.
- It is not possible to connect to the *OpenVidu* server during the object creation. Later you will call *fetch()* when it's possible.
- After creating the *OpenVidu* object you will call *fetch()* instantly.
- Your program will create a session as soon as the *OpenVidu* object created, and only use that.
- All you need is getting the config using *get_config()*.

5.2 Timeouts

It is possible to setup a timeout for every request instantiated by the *OpenVidu* object. This can be done by setting the *timeout* attribute when creating the *OpenVidu* instance.

This value is passed to the underlying *Requests* session. For more information of the possible combinations and how they work, see requests's documentation on this topic.

The Timeout exceptions raised by *Requests* are not modified in any way, so you have to catch the exceptions raised directly by requests.

Here is an example on using timeouts:

```
from pyopenvidu import OpenVidu
import requests.exceptions

#  Set both connect and read timeout to 2 sec
openvidu = OpenVidu(OPENVIDU_URL, OPENVIDU_SECRET, initial_fetch=False, timeout=2)

try:
    session = openvidu.create_session()

except requests.exceptions.Timeout:
    print("This didn't work: Operation timed out")
```

CHAPTER 6

Migrating

This page summarizes breaking changes between versions in order to help developers migrating to a newer version of PyOpenVidu.

6.1 From 0.1.4 to 0.2.0

With the release of OpenVidu 2.16.0, the REST API of the OpenVidu server changed a lot in various ways. This has proven that it is not safe make any assumptions about how stable this API will be in the future.

Without any guarantee about said stability. The architecture of the PyOpenVidu library had to be made a little loose in order help adopting later changes.

Because this release must bring changes to the interface of this library (because of the above described reasons), a few more other modifications are added in, to make make the library more maintainable, straightforward to use and less prone to errors.

6.1.1 Behavioural changes

Although many things changed, the followings only affect your program in a few edge-cases. And only minor changes are required in those cases as well.

Fetching the OpenVidu object no longer updates the existing session objects

This was a broken design to begin with. Originally if you called *fetch()* on the OpenVidu instance, it would update all other instances of *OpenViduSession*. This required the OpenVidu object to keep the reference for each *OpenViduSession* object. Which meant very tight coupling of those objects. This tight coupling caused more problems than it should.

First of all thread safety caused a huge issue when those objects are accessed from different threads (Python's GIL saved most of the headaches, but this still could cause strange behavior). In order to solve this problem, a shared lock

between objects was used (See the next section) which not just caused performance issues, but was hard to maintain as well.

Second was the rigid architecture of how the updating of internal representations processed, was an awkward point when the OpenVidu REST API changed in version 2.16.0. Because OpenVidu REST API is not stable, and can change even in minor versions, such a rigid architecture is not a good idea to keep.

Thirdly, this behaviour also caused confusion for software developers, because it is different from what one might expect.

Because of the above outlined issues, this feature was removed: A *fetch()* call on the "parent" object (Not in terms of OOP) no longer cause the update of the information stored in their "children".

If you need to update many OpenVidu session objects in batch, you should call *fetch()* on the OpenVidu instance itself, and use the *sessions* property, to access the updated list of sessions.

If you only need to update a few *OpenViduSession* objects, you should use their *fetch()* method instead, as it's more efficient.

The interface is no longer thread safe.

This was more like side-effect of the above mentioned (and now removed) half-solution to a problem that should not exist. With that feature removed, using locks no longer made sense.

Instead of that the developer who is using the library should have a control over how and when locks are used. This is a better approach in many ways:

- The developer can use any type of lock they want, so that it will work on any framework they might want to use (Multiprocessing, Qt's QThreads, etc.) not just plain Python threads.
- Because of the nature of this library it won't be unlikely that after some call that changes the data of a class (e.g.: *fetch()*) one may access to more than one properties. Because an internal lock would lock those individually it might be possible for another thread to change the value of the properties before they are read after changing it on the other thread. In this scenario an external lock would be needed anyway, so an internal one would be useless.
- No unwanted side effects of the unexpected locks.

Because of the above mentioned reasons, this feature is now removed.

Connection object is now dynamic

Well this is not a breaking change, but it's important to mention. As of OpenVidu release 2.16.0 it is possible to fetch info about a connection directly. This made it possible to implement *fetch()* for the connection object as well.

Base URL must include the */api/* part

Originally the */api/* part was appended to the base url by the library itself. This was needed because there was a single endpoint which could not be reached under */api* (*config*). But with OpenVidu release 2.16.0 this endpoint was moved bellow */api*. This allows more freedom of your reverse proxy and server configuration.

If you previously passed *https://my.openvidu.instance.com:4443/* to the constructor of the *OpenVidu* object. Now you have to pass *https://my.openvidu.instance.com:4443/openvidu/api/*. (The */openvidu/* part was added in the new release of OpenVidu as well)

Creating a new session no longer does an internal fetch() call

As of OpenVidu release 2.16.0 the API returns the created session object. This allows PyOpenVidu to return with the newly created *OpenViduSession* object without fetching it first.

The new object is also being added to the *OpenVidu.sessions* list, but the other sessions are not being updated at this point.

Properties no longer fail if the object is invalid

Before this release. An object might raised an exception when it was invalid and specific properties are accessed. (e.g.: connection_count of the OpenViduSession instance, and stuff like that).

This was a kind of inconsistent behaviour because not every property implemented it. And by rewriting parts of the code to use dataclasses instead of properties implemented one-by-one it would require ugly hacks to achieve.

This also introduced an unnecessary limitation, because invalid object's properties could not be used while in some scenarios this would be useful. Also programmers had to write unnecessary try-except blocks around all code that access properties/getters because it was not clear which ones may fail and which may not. Because of this inconsistency this was a perfectly useless restriction.

Because of the above mentioned reasons this behaviour is removed. From now on getters/properties won't raise any exception if they are being used. Allowing the use of those values even if the object became invalid.

6.1.2 Changes of the interface

Two type of connections

The following tables summarize the changes of various classe's methods and properties.

OpenViduSession

Old attribute	New attribute	Notes
(method) `generate_token(role, data, video_max_recv_bandwidth, video_min_recv_bandwidth, video_max_send_bandwidth, video_min_send_bandwidth, allowed_filters -> str`	*(method)* `create_webrtc_connection(role, data, video_max_recv_bandwidth, video_min_recv_bandwidth, video_max_send_bandwidth, video_min_send_bandwidth, allowed_filters) -> OpenViduWEBRTCConnection`	Token is now a property of the *OpenViduWEBRTCConnection* returned.
(method) `publish(rtsp_uri, data, adaptive_bitrate, only_play_with_subscribers, type_) -> OpenViduConnection`	*(method)* `create_ipcam_connection(rtsp_uri, data, adaptive_bitrate, only_play_with_subscribers, network_cache) -> OpenViduIPCAMConnection`	*type_* is removed A new parameter *network_cache* is added. Also default values not provided.
(property) `connections -> Iterator[OpenViduConnection]`	*(property)* `connections -> List[OpenViduConnection]`	This property is changed to a *List* from *Iterator*.

OpenViduConnection

Old attribute	New attribute	Notes
N/A	*(property)* `fetch() -> bool`	Connection objects became dynamic.
N/A	*(property)* `is_valid -> bool`	Connection objects became dynamic.
N/A	*(property)* `publisher_count -> int`	Added for convenience.
N/A	*(property)* `subscriber_count -> int`	Added for convenience.

OpenViduPublisher

Old attribute	New attribute	Notes
(property) `rtsp_uri -> str`	*(property)* `OpenViduIPCAMConnection.rtsp_uri -> str`	This property is moved from the publisher to the connection object itself.

CHAPTER 7

Module overview

Top-level package for PyOpenVidu.

7.1 Classes

7.1.1 OpenVidu

OpenVidu class.

class pyopenvidu.openvidu.**OpenVidu**(*url: str*, *secret: str*, *initial_fetch: bool = True*, *timeout: Union[int, tuple, None] = None*, *verify: Union[str, bool, None] = None*, *cert: Union[tuple, str, None] = None*)

Bases: object

This object represents a OpenVidu server instance.

Parameters

- **url** – The url to reach your OpenVidu Server instance. Typically, something like https://localhost:4443/
- **secret** – Secret for your OpenVidu Server
- **initial_fetch** – Enable the initial fetching on object creation. Defaults to *True*. If set to *False* a *fetch()* must be called before doing anything with the object. In most scenarios you won't need to change this.
- **timeout** – Set the *timeout* property of the underlying requests call. Default: None = No timeout. See https://2.python-requests.org/en/latest/user/advanced/#timeouts for possible values.
- **verify** – Set the *verify* property of the underlying requests call. Default: None = Use certifi. See https://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification.
- **cert** – Set the *cert* property of the underlying requests call. Default: None = No client cert. See https://docs.python-requests.org/en/master/user/advanced/#ssl-cert-verification

create_session(*custom_session_id: str = None*, *media_mode: str = None*) → pyopenvidu.openvidusession.OpenViduSession

Creates a new OpenVidu session.

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#post-openviduapisessions

Parameters

- **custom_session_id** – You can fix the sessionId that will be assigned to the session with this parameter.
- **media_mode** – ROUTED (default) or RELAYED

Returns The created OpenViduSession instance.

fetch() → bool

Updates every property of every active Session with the current status they have in OpenVidu Server. After calling this method you can access the updated list of active sessions through the *sessions* property.

Returns true if the Session status has changed with respect to the server, false if not. This applies to any property or sub-property of the object.

get_config() → dict

Get OpenVidu active configuration.

Unlike session related calls. This call does not require prior calling of the fetch() method. Using this function will always result an API call to the backend.

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#get-openviduapiconfig

Returns The exact response from the server as a dict.

get_session(*session_id: str*) → pyopenvidu.openvidusession.OpenViduSession

Get a currently active session to the server.

Parameters **session_id** – The ID of the session to acquire.

Returns An OpenViduSession object.

session_count

Get the number of active sessions on the server.

Returns The number of active sessions.

sessions

Get a list of currently active sessions to the server.

Returns A list of OpenViduSession objects.

7.1.2 OpenViduSession

OpenViduSession class.

class pyopenvidu.openvidusession.**OpenViduSession**(*session: requests_toolbelt.sessions.BaseUrlSession*, *data: dict*)

Bases: object

This object represents an OpenVidu Session. A session is a group of users sharing communicating each other.

Direct instantiation of this class is not supported! Use *OpenVidu.get_session* to get an instance of this class.

close()

Gracefully closes the Session: unpublishes all streams and evicts every participant. Further calls to this object will fail.

connection_count
Get the number of active connections to the session.

Returns The number of active connections.

create_ipcam_connection(*rtsp_uri: str, data: str = None, adaptive_bitrate: bool = None, only_play_with_subscribers: bool = None, network_cache: int = None*) → pyopenvidu.openviduconnection.OpenViduIPCAMConnection
Publishes a new IPCAM rtsp stream to the session.

Keep in mind, that if you want the newly created Connection to appear in the *connections* list, you should call fetch() before accessing the list!

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#post-openviduapisessionsltsession_idgtconnection

Parameters

- **rtsp_uri** – RTSP URI of the IP camera. For example: *rtsp://your.camera.ip:7777/path*.
- **data** – Metadata you want to associate to the camera's participant.
- **adaptive_bitrate** – Whether to use adaptive bitrate or not.
- **only_play_with_subscribers** – Enable the IP camera stream only when some user is subscribed to it.
- **network_cache** – Size of the buffer of the endpoint receiving the IP camera's stream, in milliseconds.

Returns An OpenVidu connection object represents the newly created connection.

create_webrtc_connection(*role: str = 'PUBLISHER', data: str = None, video_max_recv_bandwidth: int = None, video_min_recv_bandwidth: int = None, video_max_send_bandwidth: int = None, video_min_send_bandwidth: int = None, allowed_filters: list = None*) → pyopenvidu.openviduconnection.OpenViduWEBRTCConnection
Creates a new Connection object of WEBRTC (Regular user) type to the session.

In the video bandwidth settings 0 means unconstrained. Setting any of them (other than None) overrides the values configured in for the server.

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#post-openviduapisessionsltsession_idgtconnection

Parameters

- **role** – Allowed values: *SUBSCRIBER*, *PUBLISHER* or *MODERATOR*
- **data** – metadata associated to this token (usually participant's information)
- **video_max_recv_bandwidth** – Maximum number of Kbps that the client owning the token will be able to receive from Kurento Media Server.
- **video_min_recv_bandwidth** – Minimum number of Kbps that the client owning the token will try to receive from Kurento Media Server.
- **video_max_send_bandwidth** – Maximum number of Kbps that the client owning the token will be able to send to Kurento Media Server.
- **video_min_send_bandwidth** – Minimum number of Kbps that the client owning the token will try to send to Kurento Media Server.

- **allowed_filters** – Array of strings containing the names of the filters the user owning the token will be able to apply.

Returns An OpenVidu connection object represents the newly created connection.

fetch()
Updates every property of the OpenViduSession with the current status it has in OpenVidu Server. This is especially useful for getting the list of active connections to the OpenViduSession through the *connections* property.

Returns True if the OpenViduSession status has changed with respect to the server, False if not. This applies to any property or sub-property of the object

get_connection(*connection_id: str*) → pyopenvidu.openviduconnection.OpenViduConnection
Get a currently active connection to the server.

Parameters **connection_id** – Connection id.

Returns A OpenViduConnection objects.

signal(*type_: str = None*, *data: str = None*, *to: Optional[List[pyopenvidu.openviduconnection.OpenViduConnection]] = None*)
Sends a signal to all participants in the session or specific connections if the *to* property defined. OpenViduConnection objects also implement this method.

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#post-openviduapisignal

Parameters

- **type** – Type of the signal. In the body example of the table above, only users subscribed to Session.on('signal:MY_TYPE') will trigger that signal. Users subscribed to Session.on('signal') will trigger signals of any type.
- **data** – Actual data of the signal.
- **to** – List of OpenViduConnection objects to which you want to send the signal. If this property is not set (None) the signal will be sent to all participants of the session.

7.1.3 OpenViduConnection

OpenViduConnection class.

class pyopenvidu.openviduconnection.**OpenViduConnection**(*session: requests_toolbelt.sessions.BaseUrlSession*, *data: dict*)

Bases: object

This is a base class for connection objects.

Direct instantiation of this class is not supported! Use *OpenViduSession.connections* to get an instance of this class.

fetch() → bool
Updates every property of the connection object.

Returns true if the Connection object status has changed with respect to the server, false if not. This applies to any property or sub-property of the object.

force_disconnect()
Forces the disconnection from the session. Remember to call fetch() after this call to fetch the current properties of the Session from OpenVidu Server!

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#delete-openviduapisessionsltsession_idgtconnectionltconnection_idgt

force_unpublish_all_streams()
Forces the user to unpublish all of their Stream. OpenVidu Browser will trigger the proper events on the client-side (streamDestroyed) with reason set to "forceUnpublishByServer". After this call, the instance of the object, should be considered invalid. Remember to call fetch() after this call to fetch the actual properties of the Session from OpenVidu Server!

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#delete-openviduapisessionsltsession_idgtstreamltstream_idgt

publisher_count

signal(*type_: str = None*, *data: str = None*)
Sends a signal to this connection.

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#post-openviduapisignal

Parameters

- **type** – Type of the signal. In the body example of the table above, only users subscribed to Session.on('signal:MY_TYPE') will trigger that signal. Users subscribed to Session.on('signal') will trigger signals of any type.
- **data** – Actual data of the signal.

subscriber_count

class pyopenvidu.openviduconnection.**OpenViduIPCAMConnection**(*session: requests_toolbelt.sessions.BaseUrlSession*, *data: dict*)

Bases: *pyopenvidu.openviduconnection.OpenViduConnection*

This object represents an OpenVidu IPCAM type of Connection. This is a connection between an IPCAM and a session.

Direct instantiation of this class is not supported! Use *OpenViduSession.connections* to get an instance of this class.

class pyopenvidu.openviduconnection.**OpenViduWEBRTCConnection**(*session: requests_toolbelt.sessions.BaseUrlSession*, *data: dict*)

Bases: *pyopenvidu.openviduconnection.OpenViduConnection*

This object represents an OpenVidu WEBRTC type of Connection. This is a connection between an user and a session.

Direct instantiation of this class is not supported! Use *OpenViduSession.connections* to get an instance of this class.

7.1.4 OpenViduPublisher

OpenViduPublisher class.

class pyopenvidu.openvidupublisher.**OpenViduPublisher**(*session: requests_toolbelt.sessions.BaseUrlSession*, *session_id: str*, *data: dict*)

Bases: `object`

Direct instantiation of this class is not supported! Use *OpenViduConnection.publishers* to get an instance of this class.

force_unpublish()
Forces some user to unpublish a Stream. OpenVidu Browser will trigger the proper events on the client-side (streamDestroyed) with reason set to "forceUnpublishByServer". After this call, the instace of the object and the parent OpenViduConnection instance should be considered invalid. Remember to call fetch() after this call to fetch the current actual properties of the Session from OpenVidu Server!

https://docs.openvidu.io/en/2.16.0/reference-docs/REST-API/#delete-openviduapisessionsltsession_idgtstreamltstream_idgt

7.1.5 OpenViduSubscriber

OpenViduSubscriber class.

class pyopenvidu.openvidusubscriber.**OpenViduSubscriber**(*session: requests_toolbelt.sessions.BaseUrlSession*, *session_id: str*, *data: dict*)

Bases: object

Direct instantiation of this class is not supported! Use *OpenViduConnection.subscribers* to get an instance of this class.

7.1.6 Exceptions

exception pyopenvidu.exceptions.**OpenViduConnectionDoesNotExistsError**
Bases: *pyopenvidu.exceptions.OpenViduConnectionError*

exception pyopenvidu.exceptions.**OpenViduConnectionError**
Bases: *pyopenvidu.exceptions.OpenViduSessionError*

exception pyopenvidu.exceptions.**OpenViduError**
Bases: BaseException

exception pyopenvidu.exceptions.**OpenViduSessionDoesNotExistsError**
Bases: *pyopenvidu.exceptions.OpenViduSessionError*

exception pyopenvidu.exceptions.**OpenViduSessionError**
Bases: *pyopenvidu.exceptions.OpenViduError*

exception pyopenvidu.exceptions.**OpenViduSessionExistsError**
Bases: *pyopenvidu.exceptions.OpenViduSessionError*

exception pyopenvidu.exceptions.**OpenViduStreamDoesNotExistsError**
Bases: *pyopenvidu.exceptions.OpenViduStreamError*

exception pyopenvidu.exceptions.**OpenViduStreamError**
Bases: *pyopenvidu.exceptions.OpenViduConnectionError*

CHAPTER 8

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

8.1 Types of Contributions

8.1.1 Report Bugs

Report bugs at https://github.com/marcsello/pyopenvidu/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

8.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

8.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

8.1.4 Write Documentation

PyOpenVidu could always use more documentation, whether as part of the official PyOpenVidu docs, in docstrings, or even on the web in blog posts, articles, and such.

8.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/marcsello/pyopenvidu/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

8.2 Get Started!

Ready to contribute? Here's how to set up *pyopenvidu* for local development.

1. Fork the *pyopenvidu* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pyopenvidu.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ python3 -m venv venv
$ cd pyopenvidu/
$ python setup.py develop
```

4. Create a branch for local development (sould be branched from *dev*. Prefix it with *dev-*):

```
$ git checkout dev
$ git checkout -b dev-name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pyopenvidu tests
$ python3 setup.py pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin dev-name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

8.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/marcsello/pyopenvidu/pull_requests and make sure that the tests pass for all supported Python versions.

8.4 Tips

To run a subset of tests:

```
$ pytest tests.test_pyopenvidu
```

8.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
```

After this, merge dev into master, and create a tag with the version as name. Travis will then deploy to PyPI if tests pass.

CHAPTER 9

Credits

9.1 Development Lead

- Marcell Pünkösd <punkosdmarcell@rocketmail.com>

9.2 Contributors

None yet. Why not be the first?

CHAPTER 10

History

10.1 0.2.1 (2022-03-10)

- Added *cert* and *verify* options to the OpenVidu object constructor.
- Various fixes in the documentation
- Included Python 3.10 in the test suite

10.2 0.2.0 (2020-12-30)

- Implemented OpenVidu REST API version 2.16.0.
- Removed inter-object update
- Changed Base URL
- Removed broken "Thread safety" approach
- See the "Migrating" section of the documentation on how to update your code.

10.3 0.1.4 (2020-05-24)

- Added timeouts.
- Added possibility to disable initial fetching.
- Fixed some mistakes in the documentation.
- Reached 100% code coverage.

10.4 0.1.3 (2020-04-26)

- Implemented thread safety for the dynamic objects.
- Added IPCAM publishing option.
- Updated and restructured documentation.

10.5 0.1.2 (2020-04-07)

- Removed publisher property of the OpenViduSubscriber object, as it was removed from the OpenVidu Server as well.

10.6 0.1.1 (2020-04-04)

- Fixed dependencies not being automatically installed.
- Updated classifiers and URLs for PyPI.
- Added more tests and updated existing ones.

10.7 0.1.0 (2020-04-03)

- First release on PyPI.
- Implemented support for most of the endpoints except recording and IPCam stuff.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index