
ipywebrtc Documentation

Release 0.4.2

Maarten Breddels

Nov 21, 2018

1	Installation	3
1.1	VideoStream	3
1.2	CameraStream	4
1.3	AudioStream	7
1.4	WidgetStream	7
1.5	VideoRecorder	9
1.6	ImageRecorder	9
1.7	AudioRecorder	13
1.8	API docs	13
2	Demos	19
2.1	WebRTC and ipyvolume	19
2.2	ImageRecorder	19
2.3	WidgetStream	19
3	Indices and tables	21
	Python Module Index	23

WebRTC and MediaStream API exposed in the Jupyter notebook/lab.

IPyWebRTC's [GitHub repo](#).

See [this tweet](#) for a demo screencast.

Using ipywebrtc you can create a **MediaStream** out of:

- Any ipywidget.
- A [video](#) file.
- An [image](#) file.
- An [audio](#) file.
- Your [webcam/camera](#).

From this MediaStream you can:

- [Record a movie](#).
- [Record an image snapshot](#).
- [Record an audio fragment](#).
- Stream it to peers using the simple [chat function](#).
- Use it as a texture in [ipyvolume](#).

CHAPTER 1

Installation

Pip users:

```
$ pip install ipywebRTC # will auto enable for notebook >
↪ = 5.3
$ jupyter labextension install jupyter-webrtc # for jupyter lab
```

Conda users:

```
$ conda install -c conda-forge ipywebRTC
$ jupyter labextension install jupyter-webrtc # for jupyter lab
```

1.1 VideoStream

A *VideoStream* is a *MediaStream* from an attached video file or url.

```
In [1]: from ipywebRTC import VideoStream
```

1.1.1 Local file

You can create a video stream from a local file, note that the content of the file is embedded in the widget, meaning your notebook file can become quite large.

```
In [2]: # commented out since it increases the size of the notebook a lot
        # video = VideoStream.from_file('./Big.Buck.Bunny.mp4')
        # video

In [3]: # video
```

1.1.2 URL

A URL is also supported, but it must respect the same-origin policy (e.g. it must be hosted from the same server as the Javascript is executed from).

```
In [4]: # video2 = VideoStream.from_url('http://localhost:8888/path_to_your_hosted_file.mp4')
        video2 = VideoStream.from_url('./Big.Buck.Bunny.mp4')
        video2

VideoStream(video=Video(value=b'./Big.Buck.Bunny.mp4', format='url'))
```

In this example, video2 does not include the data of the video itself, only the url.

1.1.3 Download

For convenience, if a video is not same-origin, the below code will download it and put the content of the file in the widget (note again that the notebook will be large).

```
In [5]: # commented out since it increases the size of the notebook a lot
        # video3 = VideoStream.from_download('https://webrtc.github.io/samples/src/video/chrome.webm')
        # video3
```

1.1.4 Controlling

You can control a video for instance by linking a ToggleButton to a VideoStream:

```
In [6]: import ipywidgets as widgets

        play_button = widgets.ToggleButton(description="Play")
        widgets.jslink((play_button, 'value'), (video2, 'playing'))
        widgets.VBox(children=[video2, play_button])

VBox(children=(VideoStream(video=Video(value=b'./Big.Buck.Bunny.mp4', format='url')), ToggleButton(v

In [ ]:
```

1.2 CameraStream

A *CameraStream* is a *MediaStream* from an attached camera device or webcam.

```
In [1]: from ipywebrtc import CameraStream, ImageRecorder
```

1.2.1 With constraints

You can pass *constraints* to the camera:

```
In [2]: camera = CameraStream(constraints=
                                {'facing_mode': 'user',
                                 'audio': False,
                                 'video': {'width': 640, 'height': 480 }
                                })

        camera

CameraStream(constraints={'facing_mode': 'user', 'audio': False, 'video': {'width': 640, 'height': 480}}
```


1.2.2 Front and back camera

Or use the two convenience methods:

- `CameraStream.facing_user`
- `CameraStream.facing_environment`

```
In [3]: # this is a shorter way to get the user facing camera
        front_camera = CameraStream.facing_user(audio=False)
        # or the back facing camera
        back_camera = CameraStream.facing_environment(audio=False)
```

```
In [4]: back_camera
```

```
CameraStream(constraints={'audio': False, 'video': {'facingMode': 'environment'}})
```

1.2.3 Record images from the camera

```
In [5]: image_recorder = ImageRecorder(stream=camera)
        image_recorder
```

```
ImageRecorder(image=Image(value=b''), stream=CameraStream(constraints={'facing_mode': 'user', 'audio': False}))
```

```
In [6]: import PIL.Image
        import PIL.ImageFilter
        import io
        im = PIL.Image.open(io.BytesIO(image_recorder.image.value))
```

```
In [7]: im.filter(PIL.ImageFilter.BLUR)
```



```
In [8]: import numpy as np
        im_array = np.array(im)
        im_array

Out[8]: array([[[ 84,  76,  73, 255],
                 [ 84,  76,  73, 255],
                 [ 87,  80,  76, 255],
                 ...,
                 [ 55,  63,  65, 255],
                 [ 61,  68,  70, 255],
                 [ 64,  72,  73, 255]],

                [[ 84,  76,  73, 255],
                 [ 86,  78,  76, 255],
                 [ 86,  78,  76, 255],
                 ...,
                 [ 55,  62,  65, 255],
                 [ 63,  71,  72, 255],
                 [ 72,  79,  80, 255]],

                [[ 86,  78,  77, 255],
                 [ 87,  79,  78, 255],
                 [ 85,  77,  76, 255],
                 ...,
                 [ 60,  67,  70, 255],
                 [ 66,  73,  75, 255],
                 [ 70,  76,  78, 255]],
```

```

...
[[232, 255, 255, 255],
 [232, 255, 255, 255],
 [232, 255, 255, 255],
...
[ 37,  29,  30, 255],
[ 36,  28,  29, 255],
[ 36,  28,  29, 255]],

[[231, 255, 255, 255],
 [231, 255, 255, 255],
 [231, 255, 255, 255],
...
[ 37,  29,  30, 255],
[ 37,  29,  30, 255],
[ 37,  29,  30, 255]],

[[228, 252, 252, 255],
 [228, 252, 252, 255],
 [228, 252, 252, 255],
...
[ 36,  28,  29, 255],
[ 37,  29,  30, 255],
[ 37,  29,  30, 255]]], dtype=uint8)

```

```
In [ ]:
```

1.3 AudioStream

A `AudioStream` is similar to the `VideoStream`, but for audio only.

```

In [1]: from ipywebrtc import AudioStream

In [2]: audio = AudioStream.from_url('Big.Buck.Bunny.mp3')
        audio

AudioStream(audio=Audio(value=b'Big.Buck.Bunny.mp3', format='url'))

In [ ]: audio.playing = False

In [ ]:
```

1.4 WidgetStream

A `WidgetStream` creates a `MediaStream` out of any widget.

```
In [1]: from ipywebrtc import WidgetStream, VideoStream
```

1.4.1 Example with pythreejs: streaming of a webgl canvas

```

In [2]: from pythreejs import Mesh, SphereGeometry, MeshLambertMaterial, PerspectiveCamera, DirectionalLight
        ball = Mesh(
            geometry=SphereGeometry(radius=1),
            material=MeshLambertMaterial(color='red'),
            position=[2, 1, 0]

```

```
)

c = PerspectiveCamera(
    position=[0, 5, 5], up=[0, 1, 0],
    children=[DirectionalLight(color='white', position=[3, 5, 1], intensity=0.5)]
)

scene = Scene(children=[ball, c, AmbientLight(color='#777777')])

renderer = Renderer(
    camera=c,
    scene=scene,
    controls=[OrbitControls(controlling=c)]
)

renderer

Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color='white', intensity=0.5, position=
In [3]: # the webgl_stream will be updated after the scene has changed (so drag the above ball around
    webgl_stream = WidgetStream(widget=renderer)
    webgl_stream

WidgetStream(widget=Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color='white', inter
In [4]: # You can limit the fps
    webgl_stream2 = WidgetStream(widget=renderer, max_fps=5)
    webgl_stream2

WidgetStream(max_fps=5, widget=Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color='wh
```

1.4.2 Example with ipywidgets: streaming of a slider widget

```
In [5]: from ipywidgets import FloatSlider
    slider = FloatSlider(
        value=7.5,
        step=0.1,
        description='Test:',
        disabled=False,
        continuous_update=False,
        orientation='horizontal',
        readout=True,
        readout_format='.1f',
    )

    slider

FloatSlider(value=7.5, continuous_update=False, description='Test:', readout_format='.1f')
In [6]: widget_stream = WidgetStream(widget=slider, max_fps=1)
    widget_stream

WidgetStream(max_fps=1, widget=FloatSlider(value=7.5, continuous_update=False, description='Test:',
In [ ]: # Close the stream
    widget_stream.close()
```

1.4.3 Example with ipyleaflet: streaming of a map widget

```
In [7]: from ipyleaflet import Map
    m = Map(center=(46, 14), zoom=5)
```

```

m
Map(basemap={'url': 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', 'max_zoom': 19, 'attributi
In [8]: map_stream = WidgetStream(widget=m, max_fps=1)
       map_stream

WidgetStream(max_fps=1, widget=Map(basemap={'url': 'https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.p
In [ ]: map_stream.close()
In [ ]:

```

1.5 VideoRecorder

A *VideoRecorder* allows you to record any stream object, e.g. from:

- *VideoStream*
- *WidgetStream*
- *CameraStream*

```

In [1]: from ipywebrtc import VideoStream, VideoRecorder
In [2]: video = VideoStream.from_url('./Big.Buck.Bunny.mp4')
In [3]: video

VideoStream(video=Video(value=b'./Big.Buck.Bunny.mp4', format='url'))
In [4]: recorder = VideoRecorder(stream=video)
       recorder

VideoRecorder(stream=VideoStream(video=Video(value=b'./Big.Buck.Bunny.mp4', format='url')), video=Vi
In [ ]: video.playing = False
In [ ]: recorder.video

```

Use ‘record’ button for recording. Programatical control is available using the `MediaRecorder.record` trait.

```

In [ ]: recorder.recording = True
In [ ]: recorder.recording = False

```

Saving can be done by clicking the download button, or programmatically using the `save` method. If `autosave` is `True`, the recording will be saved directly to disk.

```

In [ ]: recorder.save('example.webm')
In [ ]: from ipywidgets import Video

       example = Video.from_file('example.webm')
       example

```

1.6 ImageRecorder

A *ImageRecorder* allows you to record a screenshot from any stream object, e.g. from:

- *VideoStream*
- *WidgetStream*
- *CameraStream*

1.6.1 Example with scikit image

We first convert the png encoded data to raw pixel values (as a numpy array).

```
In [7]: import numpy as np
        im_array = np.array(im)
        im_array

Out[7]: array([[141, 127, 142, 255],
               [134, 120, 134, 255],
               [119, 105, 117, 255],
               ...,
               [205, 226, 255, 255],
               [205, 226, 255, 255],
               [205, 226, 255, 255]],

               [[139, 124, 138, 255],
               [131, 116, 129, 255],
               [115, 100, 111, 255],
               ...,
               [205, 226, 255, 255],
               [205, 226, 255, 255],
               [205, 226, 255, 255]],

               [[130, 115, 127, 255],
               [123, 109, 119, 255],
               [107, 93, 101, 255],
               ...,
               [205, 226, 255, 255],
               [205, 226, 255, 255],
               [205, 226, 255, 255]],

               ...,
               [[ 99, 99, 48, 255],
               [ 99, 100, 45, 255],
               [100, 102, 41, 255],
               ...,
               [171, 200, 71, 255],
               [149, 178, 51, 255],
               [156, 184, 60, 255]],

               [[ 99, 98, 52, 255],
               [ 98, 98, 48, 255],
               [100, 101, 44, 255],
               ...,
               [172, 202, 72, 255],
               [156, 185, 58, 255],
               [155, 183, 59, 255]],

               [[ 94, 93, 49, 255],
               [ 95, 95, 47, 255],
               [100, 101, 46, 255],
               ...,
               [174, 203, 73, 255],
               [160, 188, 62, 255],
               [154, 182, 58, 255]]], dtype=uint8)
```

Now we can do easy manipulations, such as reordering the channels (red, green, blue, alpha)

```
In [8]: PIL.Image.fromarray(im_array[...,::-1])
```



Or build a slightly more sophisticated example using scikit-image (run this notebook with a live kernel, such as mybinder for this to work)

```
In [9]: from skimage.filters import roberts, sobel, scharr, prewitt
        from skimage.color import rgb2gray
        from skimage.color.adapt_rgb import adapt_rgb, each_channel, hsv_value
        from skimage import filters

image = widgets.Image()
output = widgets.Output()
filter_options = [('roberts', roberts), ('sobel', sobel), ('scharr', scharr), ('prewitt', prewitt)]
filter_widget = widgets.ToggleButtons(options=filter_options)

@output.capture()
def update_image(change):
    # turn into nparray
    im_in = PIL.Image.open(io.BytesIO(image_recorder.image.value))
    im_array = np.array(im_in)[..., :3] # no alpha

    # filter
    filter_function = filter_widget.value
    im_array_edges = adapt_rgb(each_channel)(filter_function)(im_array)
    im_array_edges = ((1-im_array_edges) * 255).astype(np.uint8)
    im_out = PIL.Image.fromarray(im_array_edges)

    # store in image widget
    f = io.BytesIO()
    im_out.save(f, format='png')
    image.value = f.getvalue()

image_recorder.image.observe(update_image, 'value')
filter_widget.observe(update_image, 'value')
widgets.VBox([filter_widget, video, widgets.HBox([image_recorder, image]), output])
```



```
VBox(children=(ToggleButtons(options=({'roberts', <function roberts at 0x1c1089dae8>), ('sobel', <fun
In [ ]:
```

1.7 AudioRecorder

A *AudioRecorder* allows you to record audio from almost any stream object, e.g. from:

- *VideoStream*
- *AudioStream*
- *WidgetStream*
- *CameraStream*

```
In [1]: from ipywebrtc import VideoStream, AudioStream, AudioRecorder
In [2]: video = VideoStream.from_url('./Big.Buck.Bunny.mp4')
        video

VideoStream(video=Video(value=b'./Big.Buck.Bunny.mp4', format='url'))

In [3]: recorder = AudioRecorder(stream=video)
        recorder

AudioRecorder(audio=Audio(value=b'', format='webm'), stream=VideoStream(video=Video(value=b'./Big.Bu
In [4]: video.playing = False
In [5]: recorder.audio

Audio(value=b'\x1aE\xdf\xa3\xa3B\x86\x81\x01B\xf7\x81\x01B\xf2\x81\x04B\xf3\x81\x08B\x82\x88matroska
In [ ]: recorder.save('example.webm')
In [ ]: from ipywidgets import Audio

        example = Audio.from_file('example.webm')
        example

In [ ]: audio_stream = AudioStream.from_file('example.webm')
        audio_stream

In [ ]: recorder2 = AudioRecorder(stream=audio_stream)
        recorder2

In [ ]: audio_stream.playing = False
```

1.8 API docs

Note that *ipywebrtc.webrtc* is imported in the ipywebrtc namespace, so you can access ipywebrtc.CameraStream instead of *ipywebrtc.webrtc.CameraStream*.

1.8.1 ipywebrtc

`ipywebrtc.chat` (*room=None, stream=None, **kwargs*)
Quick setup for a chatroom.

Parameters

- **room** (*str*) – Roomname, if not given, a random sequence is generated and printed.
- **stream** (*MediaStream*) – The media stream to share, if not given a *CameraStream* will be created.

Return type *WebRTCRoom*

1.8.2 ipywebrtc.webrtc

class ipywebrtc.webrtc.**MediaStream** (***kwargs*)
Bases: *ipywidgets.widgets.domwidget.DOMWidget*

Represents a media source.

See <https://developer.mozilla.org/en/docs/Web/API/MediaStream> for details In practice this can a stream coming from an *HTMLVideoElement*, *HTMLCanvasElement* (could be a WebGL canvas) or a camera/webcam/microphone using *getUserMedia*.

The currently supported *MediaStream* (subclasses) are:

- *VideoStream*: A video file/data as media stream.
- *CameraStream*: Webcam/camera as media stream.
- *ImageStream*: An image as a static stream.
- *WidgetStream*: Arbitrary *DOMWidget* as stream.

A *MediaStream* can be used with:

- *VideoRecorder*: To record a movie
- *ImageRecorder*: To create images/snapshots.
- *AudioRecorder*: To record audio.
- *WebRTCRoom* (or rather *WebRTCRoomMqtt*): To stream a media stream to a (set of) peers.

class ipywebrtc.webrtc.**VideoStream** (***kwargs*)
Bases: *ipywebrtc.webrtc.MediaStream*

Represent a stream of a video element

classmethod **from_download** (*url*, ***kwargs*)

Create a *VideoStream* from a url by downloading Parameters ——— url: str

The url of the file that will be downloaded and its bytes assigned to the value trait of the video trait.

****kwargs:** Extra keyword arguments for *VideoStream*

Returns an *VideoStream* with the value set from the content of a url.

classmethod **from_file** (*filename*, ***kwargs*)

Create a *VideoStream* from a local file.

filename: str The location of a file to read into the value from disk.

****kwargs:** Extra keyword arguments for *VideoStream*

Returns an *VideoStream*.

classmethod **from_url** (*url*, ***kwargs*)

Create a *VideoStream* from a url. This will create a *VideoStream* from a *Video* using its url

url: str The url of the file that will be used for the .video trait.

****kwargs:** Extra keyword arguments for *VideoStream*

Returns an *VideoStream*.

playing

Plays the videostream or pauses it.

video

An ipywidgets.Video instance that will be the source of the media stream.

class ipywebrtc.webrtc.CameraStream (**kwargs)

Bases: *ipywebrtc.webrtc.MediaStream*

Represents a media source by a camera/webcam/microphone using getUserMedia. See <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia> for more detail. The constraints trait can be set to specify constraints for the camera or microphone, which is described in the documentation of getUserMedia, such as in the link above. Two convenience methods are available to easily get access to the 'front' and 'back' camera, when present

```
>>> CameraStream.facing_user(audio=False)
>>> CameraStream.facing_environment(audio=False)
```

constraints

Constraints for the camera, see <https://developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia> for details.

classmethod facing_environment (audio=True, **kwargs)

Convenience method to get the camera facing the environment (often the back)

audio: bool Capture audio or not

kwargs: Extra keyword arguments passed to the *CameraStream*

classmethod facing_user (audio=True, **kwargs)

Convenience method to get the camera facing the user (often front)

audio: bool Capture audio or not

kwargs: Extra keyword arguments passed to the *CameraStream*

class ipywebrtc.webrtc.WidgetStream (**kwargs)

Bases: *ipywebrtc.webrtc.MediaStream*

Represents a widget media source.

max_fps

(int, default None) The maximum amount of frames per second to capture, or only on new data when the value is None.

widget

An instance of ipywidgets.DOMWidget that will be the source of the MediaStream.

class ipywebrtc.webrtc.ImageStream (**kwargs)

Bases: *ipywebrtc.webrtc.MediaStream*

Represent a media stream by a static image

classmethod from_download (url, **kwargs)

Create a *ImageStream* from a url by downloading Parameters ——— url: str

The url of the file that will be downloaded and its bytes assigned to the value trait of the video trait.

****kwargs:** Extra keyword arguments for *ImageStream*

Returns an *ImageStream* with the value set from the content of a url.

classmethod from_file (*filename*, ****kwargs**)

Create a *ImageStream* from a local file.

filename: **str** The location of a file to read into the value from disk.

****kwargs:** Extra keyword arguments for *ImageStream*

Returns an *ImageStream*.

classmethod from_url (*url*, ****kwargs**)

Create a *ImageStream* from a url. This will create a *ImageStream* from an *Image* using its url

url: **str** The url of the file that will be used for the .image trait.

****kwargs:** Extra keyword arguments for *ImageStream*

Returns an *ImageStream*.

image

An ipywidgets.Image instance that will be the source of the media stream.

class ipywebrtc.webrtc.VideoRecorder (*format='webm', filename=traitlets.Undefined, recording=False, autosave=False, **kwargs*)

Bases: ipywebrtc.webrtc.Recorder

Creates a recorder which allows to record a *MediaStream* widget, play the record in the Notebook, and download it or turn it into a *Video* widget.

For help on supported values for the “codecs” attribute, see <https://stackoverflow.com/questions/41739837/all-mime-types-supported-by-mediarecorder-in-firefox-and-chrome>

codecs

Optional codecs for the recording, e.g. “vp8” or “vp9, opus”.

save (*filename=None*)

Save the video to a file, if no filename is given it is based on the filename trait and the format.

```
>>> recorder = VideoRecorder(filename='test', format='mp4')
>>> ...
>>> recorder.save() # will save to test.mp4
>>> recorder.save('foo') # will save to foo.mp4
>>> recorder.save('foo.dat') # will save to foo.dat
```

video

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the klass attribute

class ipywebrtc.webrtc.ImageRecorder (*format='png', filename=traitlets.Undefined, recording=False, autosave=False, **kwargs*)

Bases: ipywebrtc.webrtc.Recorder

Creates a recorder which allows to grab an *Image* from a *MediaStream* widget.

format

The format of the image.

image

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the class attribute

save (*filename=None*)

Save the image to a file, if no filename is given it is based on the filename trait and the format.

```
>>> recorder = ImageRecorder(filename='test', format='png')
>>> ...
>>> recorder.save() # will save to test.png
>>> recorder.save('foo') # will save to foo.png
>>> recorder.save('foo.dat') # will save to foo.dat
```

class ipywebrtc.webrtc.**AudioRecorder** (*format='webm', filename=traitlets.Undefined, recording=False, autosave=False, **kwargs*)

Bases: ipywebrtc.webrtc.Recorder

Creates a recorder which allows to record the Audio of a MediaStream widget, play the record in the Notebook, and download it or turn it into an Audio widget.

For help on supported values for the “codecs” attribute, see <https://stackoverflow.com/questions/41739837/all-mime-types-supported-by-mediarecorder-in-firefox-and-chrome>

audio

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the class attribute

codecs

Optional codecs for the recording, e.g. “opus”.

save (*filename=None*)

Save the audio to a file, if no filename is given it is based on the filename trait and the format.

```
>>> recorder = AudioRecorder(filename='test', format='mp3')
>>> ...
>>> recorder.save() # will save to test.mp3
>>> recorder.save('foo') # will save to foo.mp3
>>> recorder.save('foo.dat') # will save to foo.dat
```

class ipywebrtc.webrtc.**WebRTCPeer** (***kwargs*)

Bases: ipywidgets.widgets.domwidget.DOMWidget

A peer-to-peer webrtc connection

connect ()**connected**

A boolean (True, False) trait.

failed

A boolean (True, False) trait.

id_local

A trait for unicode strings.

id_remote

A trait for unicode strings.

stream_local

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the class attribute

stream_remote

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the class attribute

class ipywebrtc.webrtc.**WebRTCRoom** (**kwargs)

Bases: `ipywidgets.widgets.domwidget.DOMWidget`

A ‘chatroom’, which consists of a list of *:WebRTCPeer* connections

nickname

A trait for unicode strings.

peers

An instance of a Python list.

room

A trait for unicode strings.

room_id

A trait for unicode strings.

stream

A trait whose value must be an instance of a specified class.

The value can also be an instance of a subclass of the specified class.

Subclasses can declare default classes by overriding the class attribute

streams

An instance of a Python list.

class ipywebrtc.webrtc.**WebRTCRoomLocal** (**kwargs)

Bases: `ipywebrtc.webrtc.WebRTCRoom`

class ipywebrtc.webrtc.**WebRTCRoomMqtt** (**kwargs)

Bases: `ipywebrtc.webrtc.WebRTCRoom`

Use a mqtt server to connect to other peers

server

A trait for unicode strings.

2.1 WebRTC and ipyvolume

Use remote MediaStreams and show them in 3d using `ipyvolume`.

Fig. 1: webrtc

2.2 ImageRecorder

Record and image from *any* stream for postprocessing.

Fig. 2: recorder

2.3 WidgetStream

Turn *any* widget into a MediaStream.

Fig. 3: widget-stream

CHAPTER 3

Indices and tables

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

i

`ipywebrtc`, [13](#)

`ipywebrtc.webrtc`, [14](#)

A

audio (*ipywebrtc.webrtc.AudioRecorder attribute*), 17
 AudioRecorder (*class in ipywebrtc.webrtc*), 17

C

CameraStream (*class in ipywebrtc.webrtc*), 15
 chat () (*in module ipywebrtc*), 13
 codecs (*ipywebrtc.webrtc.AudioRecorder attribute*), 17
 codecs (*ipywebrtc.webrtc.VideoRecorder attribute*), 16
 connect () (*ipywebrtc.webrtc.WebRTCPeer method*), 17
 connected (*ipywebrtc.webrtc.WebRTCPeer attribute*), 17
 constraints (*ipywebrtc.webrtc.CameraStream attribute*), 15

F

facing_environment () (*ipywebrtc.webrtc.CameraStream class method*), 15
 facing_user () (*ipywebrtc.webrtc.CameraStream class method*), 15
 failed (*ipywebrtc.webrtc.WebRTCPeer attribute*), 17
 format (*ipywebrtc.webrtc.ImageRecorder attribute*), 16
 from_download () (*ipywebrtc.webrtc.ImageStream class method*), 15
 from_download () (*ipywebrtc.webrtc.VideoStream class method*), 14
 from_file () (*ipywebrtc.webrtc.ImageStream class method*), 16
 from_file () (*ipywebrtc.webrtc.VideoStream class method*), 14
 from_url () (*ipywebrtc.webrtc.ImageStream class method*), 16
 from_url () (*ipywebrtc.webrtc.VideoStream class method*), 14

I

id_local (*ipywebrtc.webrtc.WebRTCPeer attribute*), 17

id_remote (*ipywebrtc.webrtc.WebRTCPeer attribute*), 17

image (*ipywebrtc.webrtc.ImageRecorder attribute*), 16
 image (*ipywebrtc.webrtc.ImageStream attribute*), 16
 ImageRecorder (*class in ipywebrtc.webrtc*), 16
 ImageStream (*class in ipywebrtc.webrtc*), 15
 ipywebrtc (*module*), 13
 ipywebrtc.webrtc (*module*), 14

M

max_fps (*ipywebrtc.webrtc.WidgetStream attribute*), 15
 MediaStream (*class in ipywebrtc.webrtc*), 14

N

nickname (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18

P

peers (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18
 playing (*ipywebrtc.webrtc.VideoStream attribute*), 15

R

room (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18
 room_id (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18

S

save () (*ipywebrtc.webrtc.AudioRecorder method*), 17
 save () (*ipywebrtc.webrtc.ImageRecorder method*), 17
 save () (*ipywebrtc.webrtc.VideoRecorder method*), 16
 server (*ipywebrtc.webrtc.WebRTCRoomMqtt attribute*), 18
 stream (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18
 stream_local (*ipywebrtc.webrtc.WebRTCPeer attribute*), 17
 stream_remote (*ipywebrtc.webrtc.WebRTCPeer attribute*), 18
 streams (*ipywebrtc.webrtc.WebRTCRoom attribute*), 18

V

video (*ipywebrtc.webrtc.VideoRecorder attribute*), 16

video (*ipywebrtc.webrtc.VideoStream attribute*), 15

VideoRecorder (*class in ipywebrtc.webrtc*), 16

VideoStream (*class in ipywebrtc.webrtc*), 14

W

WebRTCPeer (*class in ipywebrtc.webrtc*), 17

WebRTCRoom (*class in ipywebrtc.webrtc*), 18

WebRTCRoomLocal (*class in ipywebrtc.webrtc*), 18

WebRTCRoomMqtt (*class in ipywebrtc.webrtc*), 18

widget (*ipywebrtc.webrtc.WidgetStream attribute*), 15

WidgetStream (*class in ipywebrtc.webrtc*), 15