ipywebrtc Documentation

Release 0.6.0

Maarten Breddels

Mar 29, 2021

EXAMPLES AND API DOCS:

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

ONE

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.

[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.

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

[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).

[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).

[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 intance by linking a ToggleButton to a VideoStream:

[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(value=False...
```

[ ]:
```

```

1.2 CameraStream

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

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

1.2.1 With constraints

You can pass *constraints* to the camera:

[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

[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)
```

[4]:
```
back_camera
```

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

1.2.3 Record images from the camera

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

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

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

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

[7]:

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

[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],
        ...,
```

(continues on next page)

(continued from previous page)

```
         [ 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)
```

[ ]:

1.3 AudioStream

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

[1]:
```
from ipywebrtc import AudioStream
```

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

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

[ ]:
```
audio.playing = False
```

[ ]:

1.4 WidgetStream

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

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

1.4.1 Example with pythreejs: streaming of a webgl canvas

[2]:
```
from pythreejs import Mesh, SphereGeometry, MeshLambertMaterial, PerspectiveCamera,␣
↪DirectionalLight, Scene, AmbientLight, Renderer, OrbitControls
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=(3.0, 5.0,...
```

[3]:
```
# the webgl_stream will be updated after the scene has changed (so drag the above␣
↪ball around if nothing happens)
webgl_stream = WidgetStream(widget=renderer)
webgl_stream
```

```
WidgetStream(widget=Renderer(camera=PerspectiveCamera(children=(DirectionalLight(color=
↪'white', intensity=0.5,...
```

[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='white',␣
↪int...
```

1.4.2 Example with ipywidgets: streaming of a slider widget

[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')
```

[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:', readout_fo...
```

[ ]:
```
# Close the stream
widget_stream.close()
```

1.4.3 Example with ipyleaflet: streaming of a map widget

[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, 'attribution': 'Map ...
```

[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}.png', 'max_...
```

[ ]:
```
map_stream.close()
```

[ ]:
```

```

1.5 VideoRecorder

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

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

[1]:
```
from ipywebrtc import VideoStream, VideoRecorder
```

[2]:
```
video = VideoStream.from_url('./Big.Buck.Bunny.mp4')
```

[3]:
```
video
```

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

[4]:
```
recorder = VideoRecorder(stream=video)
recorder
```

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

[ ]:
```
video.playing = False
```

[ ]:
```
recorder.video
```

Use 'record' button for recording. Programatical control is available using the MediaRecorder.record trait.

[ ]:
```
recorder.recording = True
```

[ ]:
```
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.

[ ]:
```
recorder.save('example.webm')
```

[ ]:
```
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]:
```
import ipywidgets as widgets
from ipywebrtc import ImageRecorder, VideoStream
```

[2]:
```
video = VideoStream.from_url('Big.Buck.Bunny.mp4')
video
```

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

Using the image recorder, you can take screenshot of the stream clicking the camera button

[3]:
```
image_recorder = ImageRecorder(stream=video)
image_recorder
```

```
ImageRecorder(image=Image(value=b''), stream=VideoStream(video=Video(value=b'Big.Buck.
↪Bunny.mp4', format='url'...
```

[4]:
```
image_recorder.image
```

```
Image(value=b'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x02\x80\x00\x00\x01d\x08\
↪x06\x00\x00\x00\xb0\xf0\x8a...
```

Or do it, programatically:

[ ]:
```
image_recorder.recording = True
```

[ ]:
```
image_recorder.autosave = False
```

[ ]:
```
image_recorder.download()
```

[ ]:
```
image_recorder.image.height
```

The data is PNG encoded (by default), so we show how to use PIL to read in the data

[5]:
```
import PIL.Image
import PIL.ImageFilter
import io
im = PIL.Image.open(io.BytesIO(image_recorder.image.value))
```

PIL Images display by default as image in the notebook. Calling the filter methods returns a new image which gets displayed directly.

[6]:
```
im.filter(PIL.ImageFilter.BLUR)
```

[6]:

1.6.1 Example with scikit image

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

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

[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]],
```

(continues on next page)

(continued from previous page)

```
       ...,
       [[ 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 manipulatios, such as reordering the channels (red, green, blue, alpha)

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

[8]:

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

[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', <function sob...
```

[ ]:

1.7 AudioRecorder

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

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

[1]:
```
from ipywebrtc import VideoStream, AudioStream, AudioRecorder
```

[2]:
```
video = VideoStream.from_url('./Big.Buck.Bunny.mp4')
video
```

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

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

```
AudioRecorder(audio=Audio(value=b'', format='webm'),␣
↪stream=VideoStream(video=Video(value=b'./Big.Buck.Bunny.m...
```

[4]:
```
video.playing = False
```

[5]:
```
recorder.audio
```

```
Audio(value=b'\x1aE\xdf\xa3\xa3B\x86\x81\x01B\xf7\x81\x01B\xf2\x81\x04B\xf3\x81\x08B\
↪x82\x88matroskaB\x87\x81\...
```

[ ]:
```
recorder.save('example.webm')
```

[ ]:
```
from ipywidgets import Audio

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

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

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

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

1.8 API docs

Note that *ipywebrtc.webrtc* is imported in the ipywebrtc namespace, to 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.**AudioRecorder**(**args*, ***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 klass 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.**AudioStream**(**args*, ***kwargs*)
Bases: *ipywebrtc.webrtc.MediaStream*

Represent a stream of an audio element

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

classmethod from_download(*url*, ***kwargs*)
Create a *AudioStream* from a url by downloading

Parameters

- **url** (*str*) – The url of the file that will be downloadeded and its bytes assigned to the value trait of the video trait.
- ****kwargs** – Extra keyword arguments for *AudioStream*

classmethod from_file(*filename*, ***kwargs*)
Create a *AudioStream* from a local file.

Parameters

- **filename** (*str*) – The location of a file to read into the audio value from disk.
- ****kwargs** – Extra keyword arguments for *AudioStream*

classmethod from_url(*url*, ***kwargs*)
Create a *AudioStream* from a url.

This will create a *AudioStream* from an Audio using its url

Parameters

- **url** (*str*) – The url of the file that will be used for the .audio trait.

- ****kwargs** – Extra keyword arguments for *AudioStream*

playing
Plays the audiostream or pauses it.

class ipywebrtc.webrtc.**CameraStream**(**args*, ***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 avaiable to easily get access to the 'front' and 'back' camera, when present

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

constraints
//developer.mozilla.org/en-US/docs/Web/API/MediaDevices/getUserMedia for details.

Type Constraints for the camera, see https

classmethod facing_environment(*audio=True*, ***kwargs*)
Convenience method to get the camera facing the environment (often the back)

Parameters

- **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)

Parameters

- **audio** (*bool*) – Capture audio or not
- ****kwargs** – Extra keyword arguments passed to the *CameraStream*

class ipywebrtc.webrtc.**ImageRecorder**(**args*, ***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 klass 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.**ImageStream**(**args*, ***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 downloadeded and its bytes assigned to the value trait of the video trait.
- ****kwargs** – Extra keyword arguments for *ImageStream*

classmethod from_file(*filename*, ***kwargs*)
Create a *ImageStream* from a local file.

Parameters

- **filename** (*str*) – The location of a file to read into the value from disk.
- ****kwargs** – Extra keyword arguments for *ImageStream*

classmethod from_url(*url*, ***kwargs*)
Create a *ImageStream* from a url.

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

Parameters

- **url** (*str*) – The url of the file that will be used for the .image trait.
- ****kwargs** – Extra keyword arguments for *ImageStream*

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

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

Represents a media source.

See https://developer.mozilla.org/nl/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.**Recorder**(**args*, ***kwargs*)
Bases: ipywidgets.widgets.domwidget.DOMWidget

autosave
If true, will save the data to a file once the recording is finished (based on filename and format)

download()
Download the recording (usually a popup appears in the browser)

filename
The filename used for downloading or auto saving.

format
The format of the recording.

recording
(boolean) Indicator and controller of the recorder state, i.e. putting the value to True will start recording.

stream
An instance of *MediaStream* that is the source for recording.

class ipywebrtc.webrtc.**VideoRecorder**(**args*, ***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.**VideoStream**(**args*, ***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 downloadeded and its bytes assigned to the value trait of the video trait.
- ****kwargs** – Extra keyword arguments for *VideoStream*

classmethod from_file(*filename*, ***kwargs*)
Create a *VideoStream* from a local file.

Parameters

- **filename** (*str*) – The location of a file to read into the value from disk.
- ****kwargs** – Extra keyword arguments for *VideoStream*

classmethod from_url(*url*, ***kwargs*)
Create a *VideoStream* from a url.

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

Parameters

- **url** (*str*) – The url of the file that will be used for the .video trait.
- ****kwargs** – Extra keyword arguments for *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.**WebRTCPeer**(**args*, ***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 klass 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 klass attribute

class ipywebrtc.webrtc.**WebRTCRoom**(**args*, ***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 klass attribute

streams
An instance of a Python list.

class ipywebrtc.webrtc.**WebRTCRoomLocal**(**args*, ***kwargs*)
Bases: *ipywebrtc.webrtc.WebRTCRoom*

class ipywebrtc.webrtc.**WebRTCRoomMqtt**(**args*, ***kwargs*)
Bases: *ipywebrtc.webrtc.WebRTCRoom*

Use a mqtt server to connect to other peers

server
A trait for unicode strings.

class ipywebrtc.webrtc.**WidgetStream**(**args*, ***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 valeus is None.

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

CHAPTER

TWO

DEMOS

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

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

i

INDEX

S

V

W