# ipywebrtc Documentation

*Release 0.4.0*

**Maarten Breddels**

**Aug 10, 2018**

# Contents:

IPyWebRTC gives you WebRTC IPython widgets in the Jupyter notebook.

Making a stream out of a video ipyvolume.mp4 (can be a same origin file for firefox only)

```
from ipywebrtc import VideoStream
video = VideoStream.from_file('ipyvolume.mp4', play=True)
video
```

[ widget ] Since video is a widget, we can control the play property using a toggle button.

```
from ipywebrtc import VideoStream
import ipywidgets as widgets
video = VideoStream.from_file('ipyvolume.mp4', play=True)
play_button = widgets.ToggleButton(description="Play")
widgets.jslink((play_button, 'value'), (video, 'play'))
widgets.VBox(children=[video, play_button])
```

[ widget ] Media recorder:

```
from ipywebrtc import VideoStream, MediaRecorder
video = VideoStream.from_file('ipyvolume.mp4', play=True)
recorder = MediaRecorder(source=video)
recorder
```

[ widget ] Camera stream (we can use camera facing user or facing environment):

```
from ipywebrtc import CameraStream
CameraStream.facing_user()
```

[ widget ] Making a 'chat room'

```
import ipywebrtc
import ipywidgets as widgets
camera = ipywebrtc.CameraStream()
room = ipywebrtc.WebRTCRoomMqtt(stream=camera, room='readthedocs')
box = widgets.HBox(children=[])
widgets.jslink((room, 'streams'), (box, 'children'))
box
```

[ widget ] Using a video as source stream instead of the camera (joining the same room)

```
import ipywebrtc
import ipywidgets as widgets
video = ipywebrtc.VideoStream.from_file('ipyvolume.mp4', play=True)
room = ipywebrtc.WebRTCRoomMqtt(stream=video, room='readthedocs')
box = widgets.HBox(children=[])
widgets.jslink((room, 'streams'), (box, 'children'))
box
```

[ widget ]

# CHAPTER 1

# VideoStream

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

## 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]: video = VideoStream.from_file('ipyvolume.mp4')
        video
```

```
VideoStream(video=Video(value=b'\x00\x00\x00 ftypisom\x00\x00\x02\x00isomiso2avc1mp41\x00\x00\x00\x08
```

```
In [3]: video
```

```
VideoStream(video=Video(value=b'\x00\x00\x00 ftypisom\x00\x00\x02\x00isomiso2avc1mp41\x00\x00\x00\x08
```

## 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('./ipyvolume.mp4')
        video2
```

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

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

## 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
```

## 1.4 Controlling

You can control a video for intance 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'./ipyvolume.mp4', format='url')), ToggleButton(value=F
```

CHAPTER 2

# CameraStream

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

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

## 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': 48
```

## 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'}})
```

## 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
```

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-6-5288f20d8ec1> in <module>()
      2 import PIL.ImageFilter
      3 import io
----> 4 im = PIL.Image.open(io.BytesIO(image_recorder.get_record().value))

AttributeError: 'ImageRecorder' object has no attribute 'get_record'
```

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-7-886fedf7a729> in <module>()
----> 1 im.filter(PIL.ImageFilter.BLUR)

NameError: name 'im' is not defined
```

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

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-8-11fff624d0a1> in <module>()
----> 1 import numpy as np
      2 im_array = np.array(im)
      3 im_array

ModuleNotFoundError: No module named 'numpy'
```

# CHAPTER 3

# API docs

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

## 3.1 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*

## 3.2 ipywebrtc.webrtc

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

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

**id**
A trait for unicode strings.

**nickname**
A trait for unicode strings.

**peers**
An instance of a Python list.

**room**
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**(**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.

CHAPTER 4

# Indices and tables

- genindex
- modindex
- search

# Python Module Index

## i

# Index