ipywebrtc Documentation

Release 0.3.0

Maarten Breddels

Jul 31, 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(value=b'\x00\x00\x00 ftypisom\x00\x00\x02\x00isomiso2avc1mp41\x00\x00\x00\x08free\x00\x05
```

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 [3]: # video2 = VideoStream.from_url('http://localhost:8888/path_to_your_hosted_file.mp4')
        video2 = VideoStream.from_url('./ipyvolume.mp4')
        video2
```

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

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 [4]: # 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 [5]: import ipywidgets as widgets

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

```
VBox(children=(VideoStream(format='url', value=b'./ipyvolume.mp4'), ToggleButton(value=False, descrip
```

CHAPTER 2

CameraStream

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

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

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'}})
```

CHAPTER 3

MediaRecorder

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

- *VideoStream*
- WidgetStream
- *CameraStream*

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

```
In [2]: video = VideoStream.from_url('./ipyvolume.mp4')
        video
```

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

```
In [3]: recorder = MediaRecorder(stream=video)
        recorder
```

```
MediaRecorder(stream=VideoStream(format='url', value=b'./ipyvolume.mp4'))
```

Use 'record' and 'play' button for recording and checking. Programatical control is available using the *MediaRecorder.record* trait, and the MediaRecorder.play method.

```
In [ ]: recorder.record = True
```

```
In [ ]: recorder.record = False
```

```
In [ ]: recorder.play()
```

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.mp4')
```

```
In [ ]: example = VideoStream.from_file('example.mp4')
        example
```

CHAPTER 4

MediaImageRecorder

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

- *VideoStream*
- WidgetStream
- *CameraStream*

```
In [1]: import ipywidgets as widgets
        from ipywebrtc import MediaImageRecorder, VideoStream
```

```
In [2]: video = VideoStream.from_url('ipyvolume.mp4')
        video
```

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

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

```
In [3]: image_recorder = MediaImageRecorder(stream=video)
        image_recorder
```

```
MediaImageRecorder(image=Image(value=b''), stream=VideoStream(format='url', value=b'ipyvolume.mp4'))
```

Or do it, programatically:

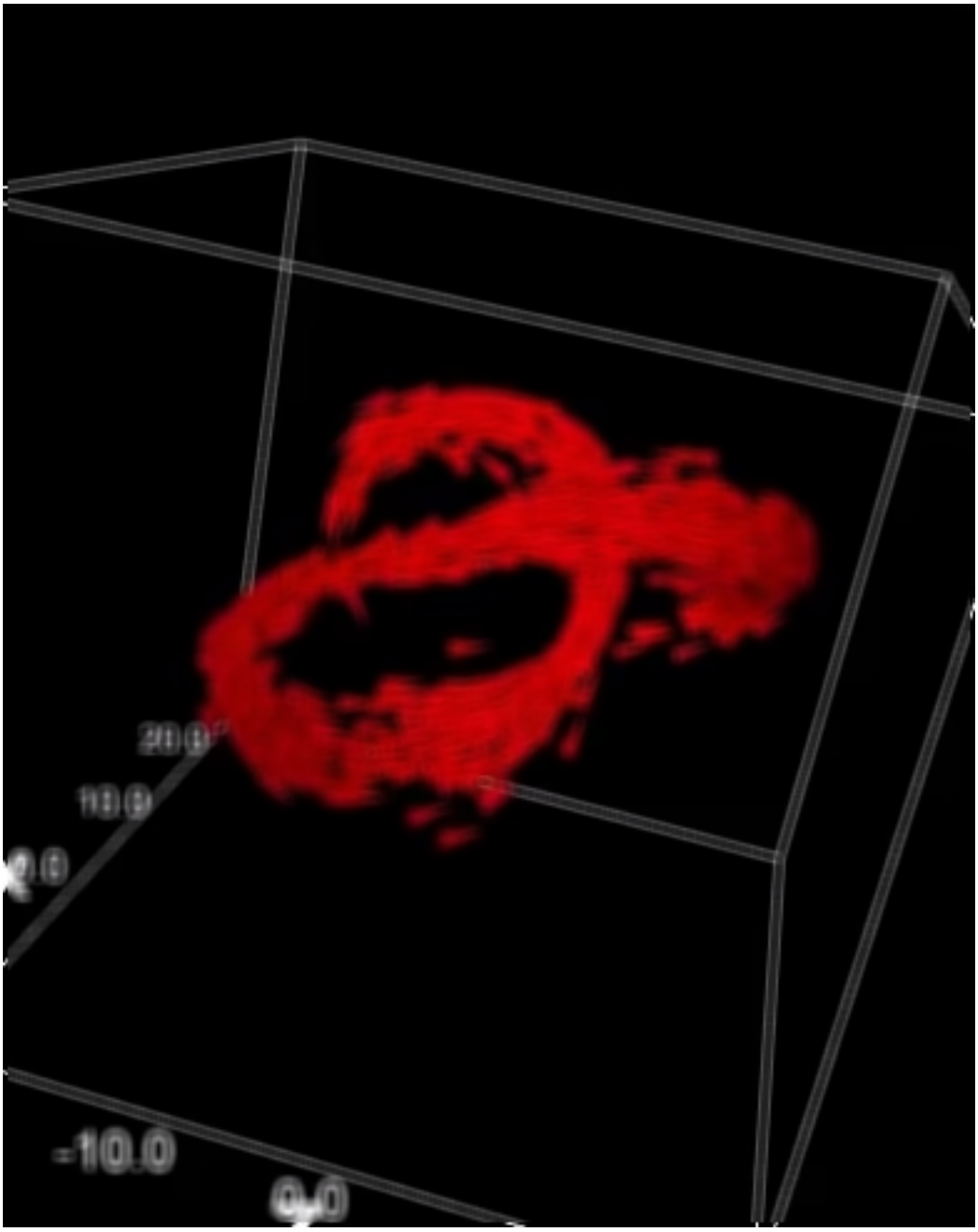
```
In [4]: image_recorder.grab()
```

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

```
In [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.

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



4.1 Example with scikit image

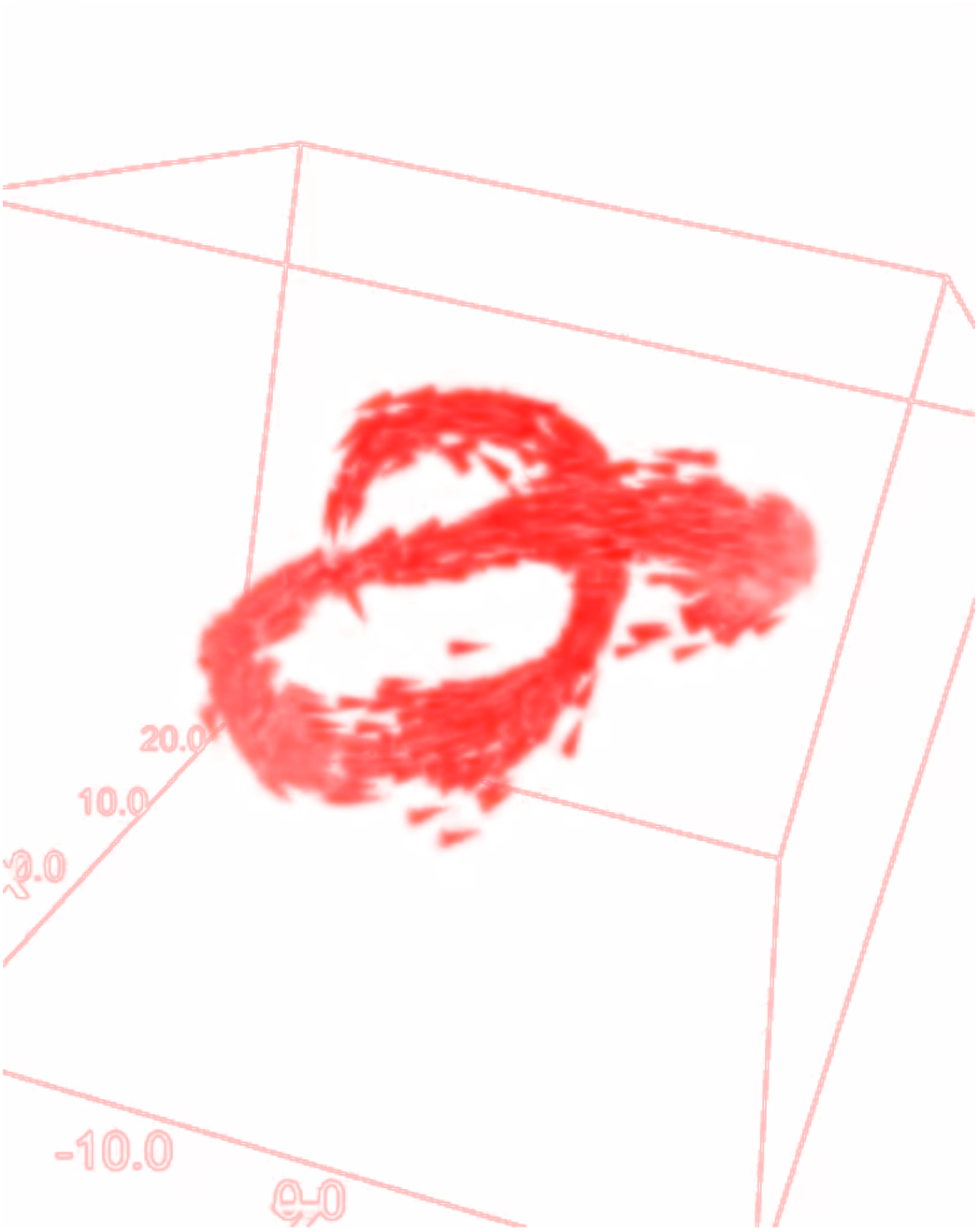
We first convert the png encoded data to raw pixel values (as a numpy array).

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

Out[7]: array([[[  2,   1,   2, 255],
                [  2,   1,   2, 255],
                [  2,   1,   2, 255],
                ...,
```

```
       [  3,   2,   3, 255],
       [  3,   2,   3, 255],
       [  3,   2,   3, 255]],

      [[  2,   1,   2, 255],
       [  2,   1,   2, 255],
       [  2,   1,   2, 255],
       ...,
       [  2,   1,   2, 255],
       [  2,   1,   2, 255],
       [  2,   1,   2, 255]],

      [[  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       ...,
       [  2,   1,   2, 255],
       [  2,   1,   2, 255],
       [  2,   1,   2, 255]],

      ...,
      [[  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       ...,
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255]],

      [[  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       ...,
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255]],

      [[  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       ...,
       [  1,   0,   1, 255],
       [  1,   0,   1, 255],
       [  1,   0,   1, 255]]], dtype=uint8)
```

Now we can do easy manipulatios, 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()

        filter_options = [('roberts', roberts), ('sobel', sobel), ('scharr', scharr), ('prewitt', pre
        filter_widget = widgets.ToggleButtons(options=filter_options)
```

```
        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.jslink((image_recorder.image, 'width'), (image, 'width'))
        widgets.jslink((image_recorder.image, 'height'), (image, 'height'))
        widgets.VBox([filter_widget, video, widgets.HBox([image_recorder, image]), ])
```

```
VBox(children=(ToggleButtons(options=(('roberts', <function roberts at 0x118b58d08>), ('sobel', <func
```

```
In [10]: image_recorder.grab()
```

CHAPTER 5

API docs

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

5.1 ipywebrtc

ipywebrtc.**chat**(*room=None*, *stream=None*, ***kwargs*)

5.2 ipywebrtc.webrtc

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

Bases: ipywidgets.widgets.domwidget.DOMWidget, ipywebrtc.webrtc.HasStream

Represents a media source.

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

Bases: *ipywebrtc.webrtc.MediaStream*

Represents a media source by a video.

data

A trait for byte strings.

filename

A trait for unicode strings.

loop

A boolean (True, False) trait.

play

A boolean (True, False) trait.

url

A trait for unicode strings.

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

Represents a media source by a camera/webcam.

audio
A boolean (True, False) trait.

close()
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

video
A boolean (True, False) trait.

class ipywebrtc.webrtc.**MediaRecorder**(***kwargs*)
Bases: ipywidgets.widgets.widget.Widget

data
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

filename
A trait for unicode strings.

mime_type
A trait for unicode strings.

record
A boolean (True, False) trait.

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

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

close()
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

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

close()
Close method.

Closes the underlying comm. When the comm is closed, all of the widget views are automatically removed from the front-end.

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*

server
A trait for unicode strings.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

i

Index