ACADEMY

```python
    # Get 6dof settings from qtm
    xml_string = await connection.get_parameters(parameters=["6d"])
    body_index = create_body_index(xml_string)

    wanted_body = "Ball"

    def on_packet(packet):
        info, bodies = packet.get_6d()
        print(
            "Framenumber: {} - Body count: {}".format(
                packet.framenumber, info.body_count
            )
        )

        if wanted_body is not None:
            # Extract one specific body
            wanted_index = body_index[wanted_body]
            position, rotation = bodies[wanted_index]
            print("{} - Pos: {} - Rot: {}".format(wanted_body, position,
        else:
            # Print all bodies
            for position, rotation in bodies:
                print("Pos: {} - Rot: {}".format(position, rotation))
```

HOW TO USE THE PYTHON SDK FOR QTM

QUALISYS AB
Kvarnbergsgatan 2 · 411 05 Gothenburg · SWEDEN
Tel. +46 31 336 94 00 · Fax +46 31 336 94 20
sales@qualisys.com · www.qualisys.com

Introduction

This manual introduces the Python implementation of the real-time protocol for QTM. The SDK allows you to stream 3D, 6DoF, and other types of data, such as analog and eye tracker data, from QTM. You can stream the data in real time or stream processed data by playing a recorded file. Beyond just streaming data, the SDK also allows Python scripts to take control of QTM and do things like start and stop capture, which can be useful for creating applications that interface with QTM. This manual demonstrates how to connect Python to QTM and how to get started developing.

Please note that the SDK requires Python version 3.5 or newer.

For more comprehensive details, please refer to the Qualisys Python Github page: https://github.com/qualisys/qualisys_python_sdk.

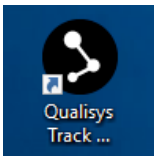
Contents

Streaming from QTM **2**
Python script for streaming data **5**
Running the program **10**

Streaming from QTM

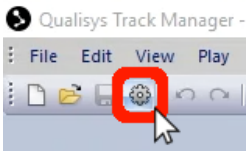
> You can stream data from QTM to Python on the same computer or stream from a computer running QTM to a different computer running Python as long as they are connected to the same network.

For the Python SDK to stream data and control QTM, QTM needs to be open.

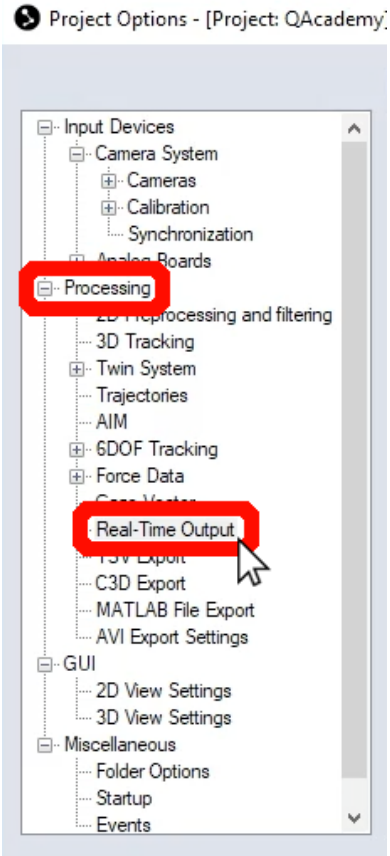


QTM also needs to be set up to allow client control:

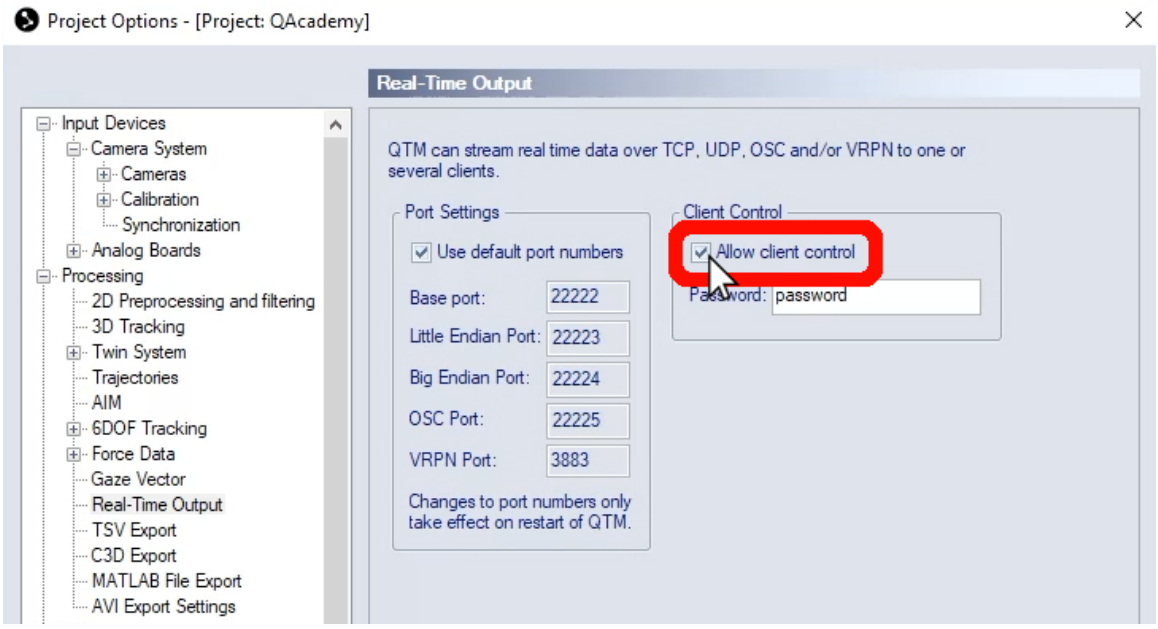
1. Open Project Options by clicking the gear icon near the top left of your screen or by typing the keyboard shortcut **Ctrl+W**.



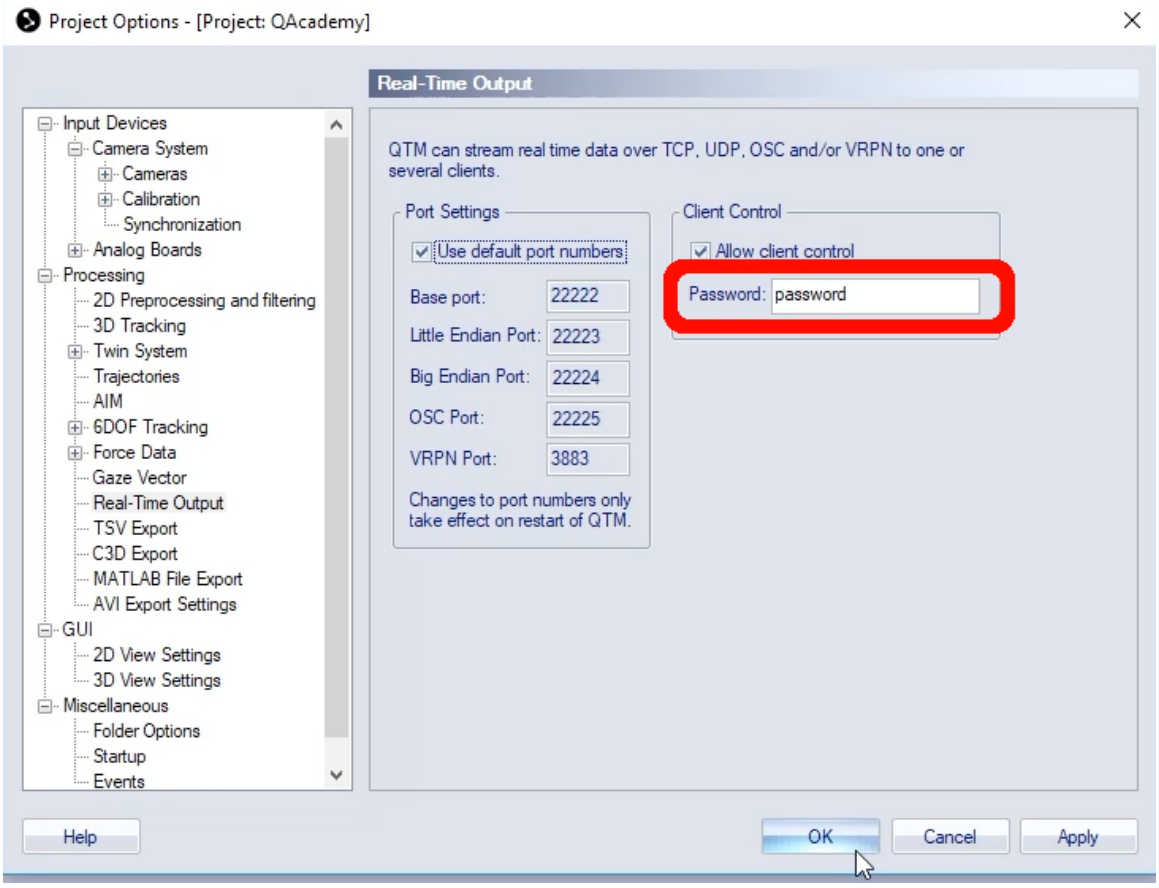
2. In the list to the left, navigate to “Processing” → “Real-Time Output.”



3. Under “Client Control,” click the checkbox next to “Allow client control.”



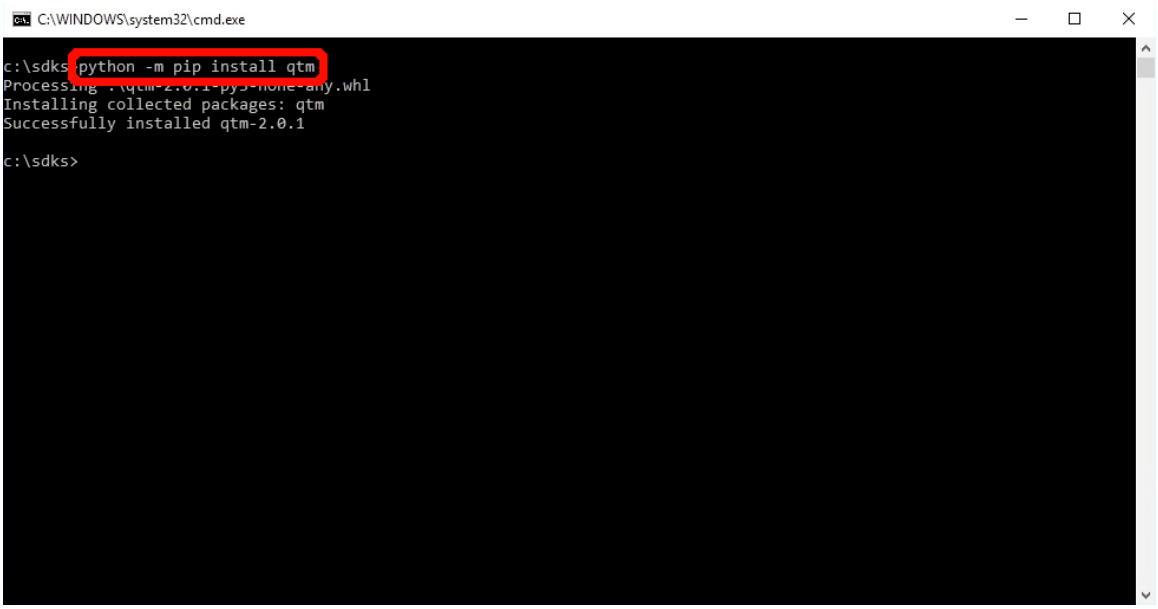
4. Set the password as desired.



5. Click “OK” to save the settings.

You will also need to install the QTM library for Python:

1. Open Command Prompt (keyboard shortcut **Win+R**, then type "cmd").
2. Enter the following: "`python -m pip install qtm`" to install the library.



Python script for streaming data

The following example script is written to take control of QTM, open a file, start playing the file with real-time output, fetch the streaming data of interest, and print out the streaming data.

It begins by importing the asynchronous library and the XML parsing library, both of which are built in. It also imports the QTM library installed earlier:

stream_6dof_example.py

```
import asyncio
import xml.etree.ElementTree as ET

import qtm

def create_body_index(xml_string):
    """ Extract a name to index dictionary from 6dof settings xml """
    xml = ET.fromstring(xml_string)

    body_to_index = {}
    for index, body in enumerate(xml.findall("*/Body/Name")):
        body_to_index[body.text.strip()] = index

    return body_to_index


async def main():

    # Connect to qtm
    connection = await qtm.connect("127.0.0.1")

    # Connection failed?
    if connection is None:
        print("Failed to connect")
        return

    # Take control of qtm, context manager will automatically release control after scope end
    async with qtm.take_control(connection, "password"):

```

The `create_body_index` function indexes the rigid bodies from QTM with their names and the order in which they will stream:

```
import qtm

def create_body_index(xml_string):
    """ Extract a name to index dictionary from 6dof settings xml """
    xml = ET.fromstring(xml_string)

    body_to_index = {}
    for index, body in enumerate(xml.findall("*/Body/Name")):
        body_to_index[body.text.strip()] = index

    return body_to_index


async def main():

    # Connect to qtm
    connection = await qtm.connect("127.0.0.1")
```

The `main` function starts by connecting to QTM:

```
    return body_to_index


async def main():

    # Connect to qtm
    connection = await qtm.connect("127.0.0.1")

    # Connection failed?
    if connection is None:
        print("Failed to connect")
        return
```

The IP address refers to the computer running QTM. If it is on the same computer, the localhost IP address of 127.0.0.1 is used.

Next, the script takes control of QTM. Note that the password needs to match the one input in QTM:

```
    # Connection failed?
    if connection is None:
        print("Failed to connect")
        return

    # Take control of qtm, context manager will automatically release control after scope end
    async with qtm.take_control(connection, "password"):

        realtime = False

        if realtime:
            # Start new realtime
            await connection.new()
        else:
            # Load qtm file
            await connection.load("C:/Users/labbuser/Documents/QAcademy/Data/Ball0002.qtm")
```

The script tells QTM to start a new real-time stream or to load a file and stream it. In this example, `realtime` is set to `False`, so QTM will load a file:

```
    # Take control of qtm, context manager will automatically release control after scope end
    async with qtm.take_control(connection, "password"):

        realtime = False

        if realtime:
            # Start new realtime
            await connection.new()
        else:
            # Load qtm file
            await connection.load("C:/Users/labbuser/Documents/QAcademy/Data/Ball0002.qtm")

            # start rtfromfile
            await connection.start(rtfromfile=True)


    # Get 6dof settings from qtm
    xml_string = await connection.get_parameters(parameters=["6d"])
    body_index = create_body_index(xml_string)
```

The next section of script gets the 6DoF settings data from QTM and uses the rigid body index created earlier:

```
39            # start rtfromfile
40            await connection.start(rtfromfile=True)
41
42
43    # Get 6dof settings from qtm
44    xml_string = await connection.get_parameters(parameters=["6d"])
45    body_index = create_body_index(xml_string)
46
47    wanted_body = "Ball"
48
49    def on_packet(packet):
50        info, bodies = packet.get_6d()
51        print(
52            "Framenumber: {} - Body count: {}".format(
53                packet.framenumber, info.body_count
54            )
55        )
```

In this example, the string variable `wanted_body` specifies the rigid body from which data will be streamed:

```
43    # Get 6dof settings from qtm
44    xml_string = await connection.get_parameters(parameters=["6d"])
45    body_index = create_body_index(xml_string)
46
47    wanted_body = "Ball"
48
49    def on_packet(packet):
50        info, bodies = packet.get_6d()
51        print(
52            "Framenumber: {} - Body count: {}".format(
53                packet.framenumber, info.body_count
54            )
55        )
```

The `on_packet` function receives the data from QTM, with `get_6d` extracting the 6DoF data from the received packet:

```
47    wanted_body = "Ball"
48
49    def on_packet(packet):
50        info, bodies = packet.get_6d()
51        print(
52            "Framenumber: {} - Body count: {}".format(
53                packet.framenumber, info.body_count
54            )
55        )
```

Then, `print` is set to output the frame number and rigid body count:

```
49    def on_packet(packet):
50        info, bodies = packet.get_6d()
51        print(
52            "Framenumber: {} - Body count: {}".format(
53                packet.framenumber, info.body_count
54            )
55        )
56
```

The data for the wanted body is extracted using the rigid body index, and the rigid body's name, position, and rotation are printed:

```
47    wanted_body = "Ball"
48
49    def on_packet(packet):
50        info, bodies = packet.get_6d()
51        print(
52            "Framenumber: {} - Body count: {}".format(
53                packet.framenumber, info.body_count
54            )
55        )
56
57        if wanted_body is not None:
58            # Extract one specific body
59            wanted_index = body_index[wanted_body]
60            position, rotation = bodies[wanted_index]
61            print("{} - Pos: {} - Rot: {}".format(wanted_body, position, rotation))
62        else:
63            # Print all bodies
64            for position, rotation in bodies:
65                print("Pos: {} - Rot: {}".format(position, rotation))
66
```

If a wanted body was not specified, the script will output the data for all rigid bodies:

```
56
57        if wanted_body is not None:
58            # Extract one specific body
59            wanted_index = body_index[wanted_body]
60            position, rotation = bodies[wanted_index]
61            print("{} - Pos: {} - Rot: {}".format(wanted_body, position, rotation))
62        else:
63            # Print all bodies
64            for position, rotation in bodies:
65                print("Pos: {} - Rot: {}".format(position, rotation))
66
67    # Start streaming frames
68    await connection.stream_frames(components=["6d"], on_packet=on_packet)
69
```

The `stream_frames` function gets data from the QTM library, specifying which components of the stream to acquire. The previously defined `on_packet` function is used when a packet arrives:

```
57        if wanted_body is not None:
58            # Extract one specific body
59            wanted_index = body_index[wanted_body]
60            position, rotation = bodies[wanted_index]
61            print("{} - Pos: {} - Rot: {}".format(wanted_body, position, rotation))
62        else:
63            # Print all bodies
64            for position, rotation in bodies:
65                print("Pos: {} - Rot: {}".format(position, rotation))
66
67    # Start streaming frames
68    await connection.stream_frames(components=["6d"], on_packet=on_packet)
69
70    # Wait asynchronously 5 seconds
71    await asyncio.sleep(5)
72
```

In this example, `asyncio.sleep(5)` tells the script to keep running for 5 seconds:

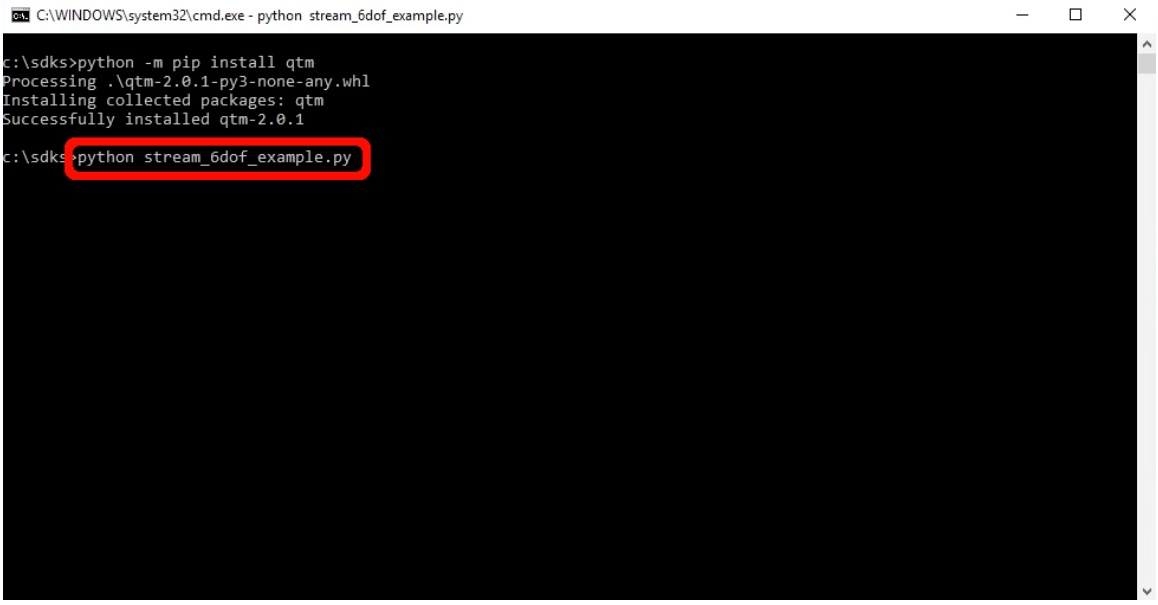
```
67    # Start streaming frames
68    await connection.stream_frames(components=["6d"], on_packet=on_packet)
69
70    # Wait asynchronously 5 seconds
71    await asyncio.sleep(5)
72
73    # Stop streaming
74    await connection.stream_frames_stop()
```

Finally, the data stream is stopped:

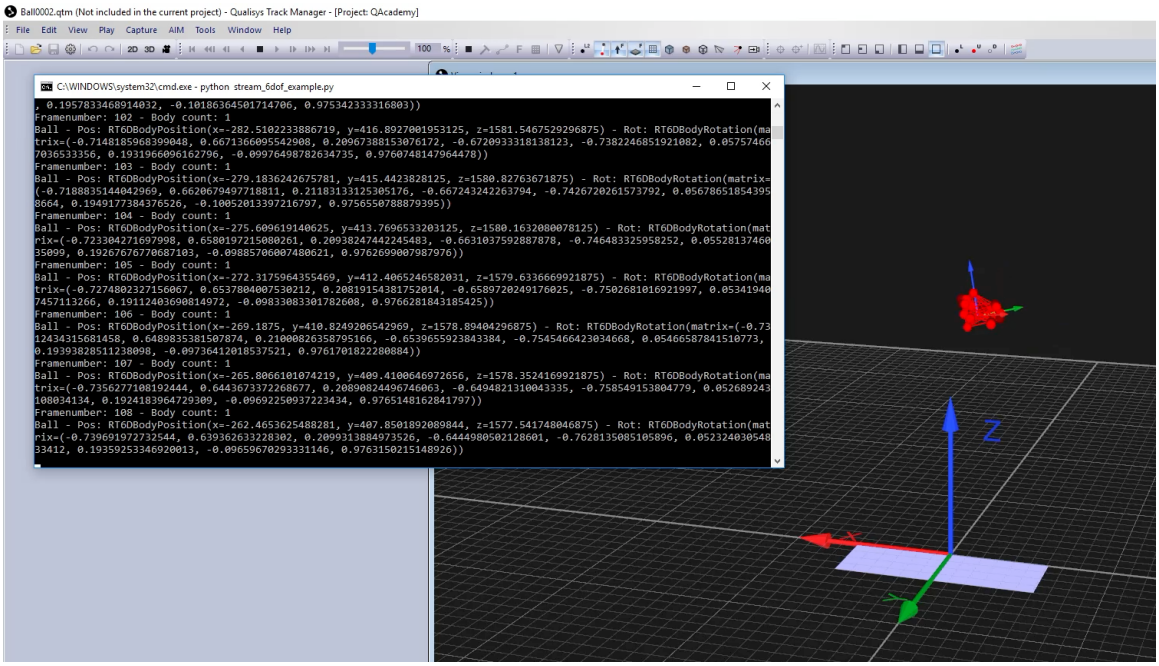
```
67    # Start streaming frames
68    await connection.stream_frames(components=["6d"], on_packet=on_packet)
69
70    # Wait asynchronously 5 seconds
71    await asyncio.sleep(5)
72
73    # Stop streaming
74    await connection.stream_frames_stop()
```

Running the program

The Python script is run from Command Prompt.



The example script automatically opens the file in QTM and starts playing it with real-time output, gets the specified real-time data, and outputs it to the console.



For more training resources, visit QAcademy at www.qualisys.com.

This manual was produced by Qualisys AB and Bassett Biomechanics.