Raspberry Pi App Engine

Quick Explanation

Version 1

Dylan Westra

11-07-2019

**Index**

[General overview 1](#_Toc12734)

[Programming an app 2](#_Toc23702)

[Server app 2](#_Toc8836)

[Client app 3](#_Toc27849)

[Server 4](#_Toc11128)

[Class diagram 4](#_Toc481)

[Server events 5](#_Toc18037)

[Server library 6](#_Toc25535)

[Web client library 7](#_Toc17279)

[Apps 8](#_Toc21335)

[MediaPlayer 8](#_Toc21681)

[Server 8](#_Toc17260)

[Webclient 8](#_Toc20116)

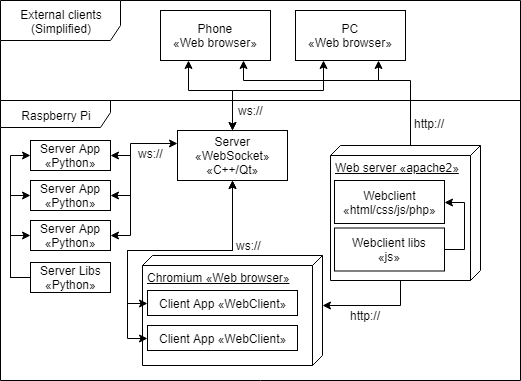
[RpaeAudio 8](#_Toc22534)

[Server 8](#_Toc26658)

[Webclient 8](#_Toc27364)

# General overview

The Raspberry Pi App Engine (from now on referred to as RPAE) will exist of a total of 5 components: Server Apps, Server Library’s, The Server, The Web client and the Web client Library’s. The web client will be called through an apache2 HTTP server, making it available to external clients. There is one issue with this setup however. Chromium browser only supports a total of 255 WebSocket connections throughout all tabs. This means a maximum of 255 Client apps could run on chromium. In a future version the Web Client should use only one WebSocket connection to manage all apps.

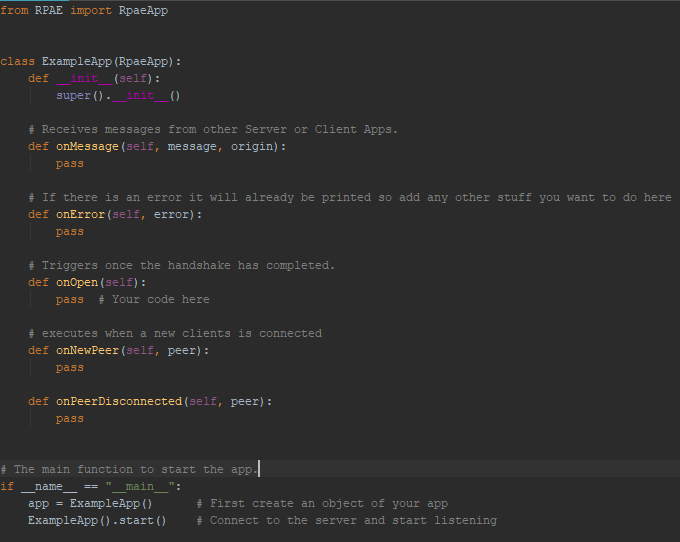


# Programming an app

When using the complete power of the RPAE one has to program 2 apps. One in Python and One in Web programming languages (HTML, CSS, JavaScript). Below is an example on how to set up an app.

Below will just be a very simplified explanation. For programming apps refer to the Dev Documents.

## Server app

An empty server app: 

onMessage: Here you can interpret messages send by other clients. Origin will contain information about the send

onError: This function will execute when the connection fails.

onOpen: This function executes once the handshake is completed and the app is connected.

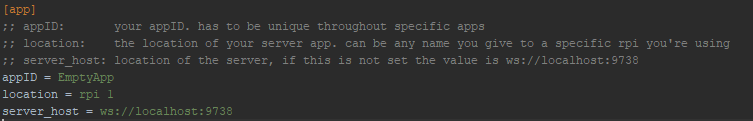
onNewPeer: This function executes when a new peer connects to the server with the same App ID

onPeerDisconnect: This function executes once a peer with the same App ID disconnects.

Below the class you should start up the app. Currently once the connection is lost the app also closes.

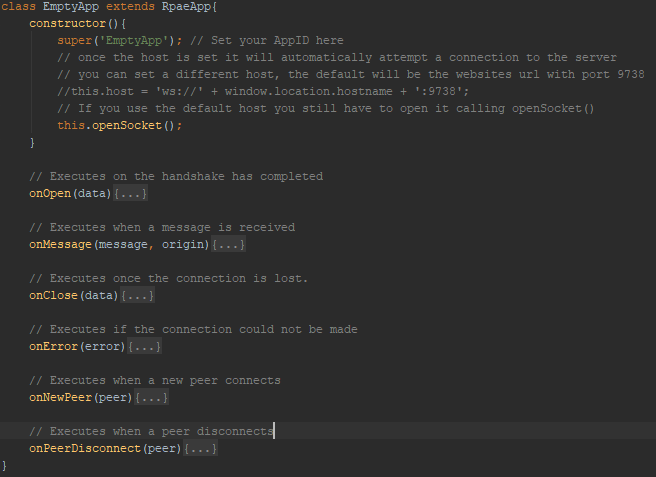
Messages to other peers can be send with the function self.sendMessage(“Your message”, target=”client, server, all or specific peers”, uid=”required for specific targets”)

The Server app also has a file called app.ini that contains the App ID. The App ID should be the same on either the Web client app and the Server app.

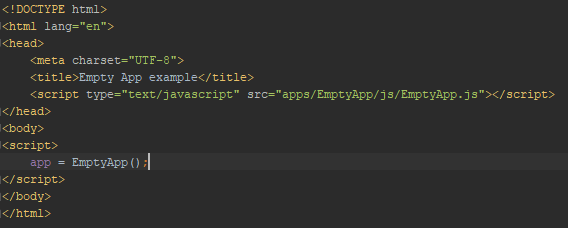


## Client app

An empty client app



As you may have noticed the Server app and Client app or quite equal on inherited functions. The difference is that the App ID isn’t set in a config file but rather in the class itself. The same sendMessage function is available to the Client apps.

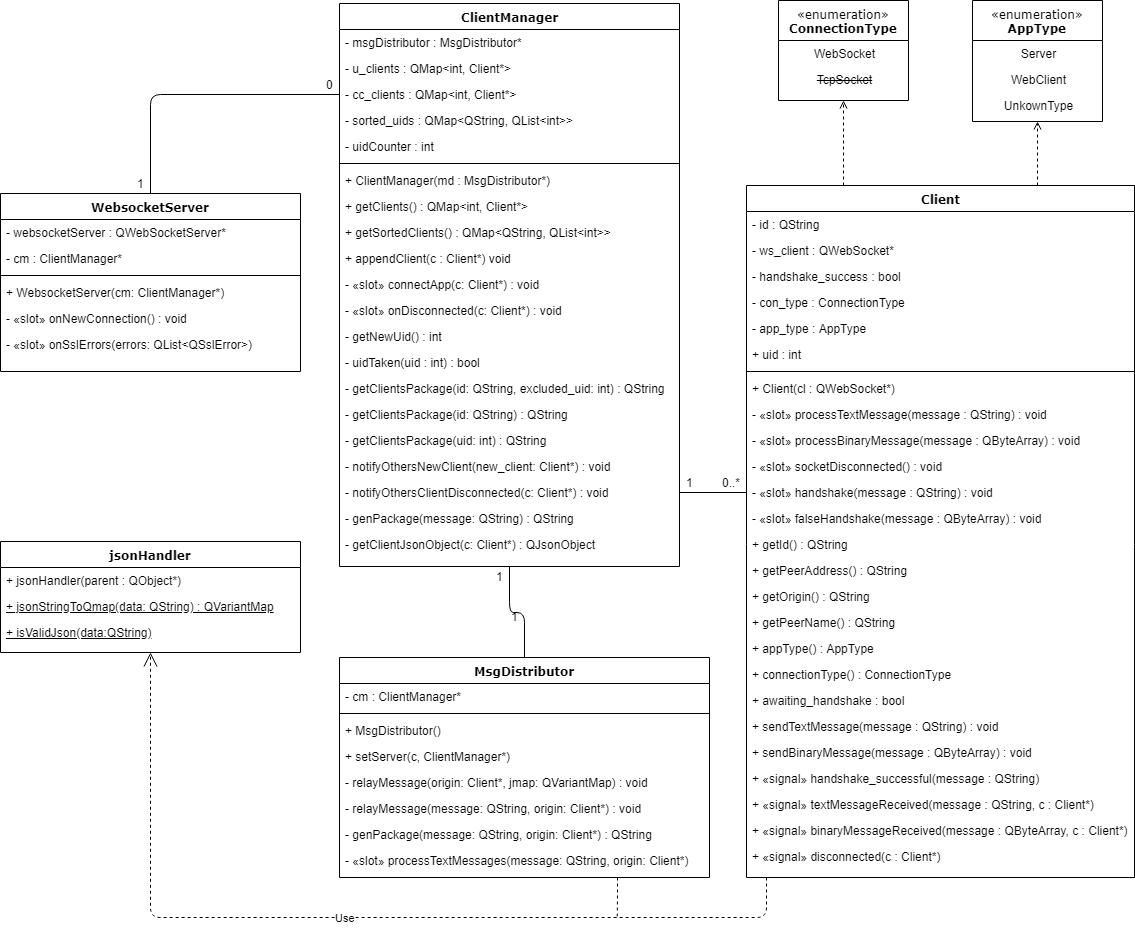


To start up an app one should declare it in one of the HTML files.

# Server

## Class diagram

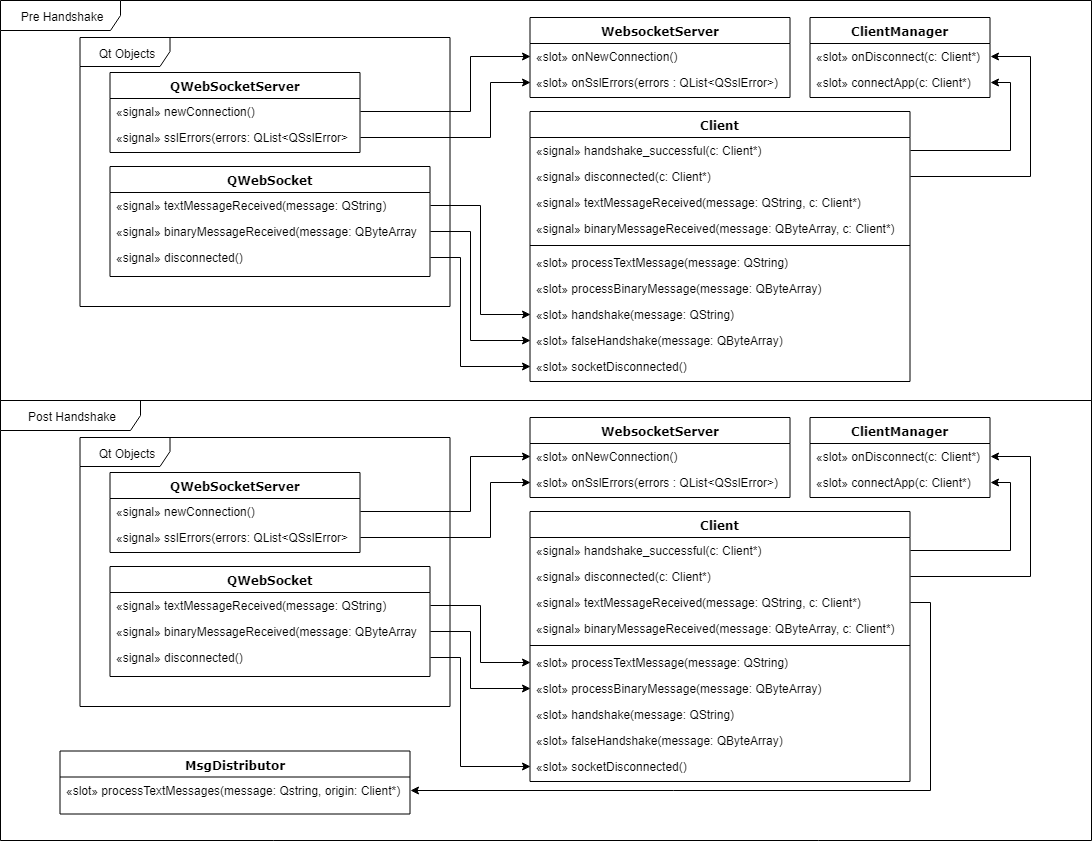
The server is written in C++ with the Qt framework. Because Qt supports signals and slots I decided to also add the signals and slots in this diagram. There wasn’t much information in how to implement this into class diagrams. Thus the result is below.



## Server events

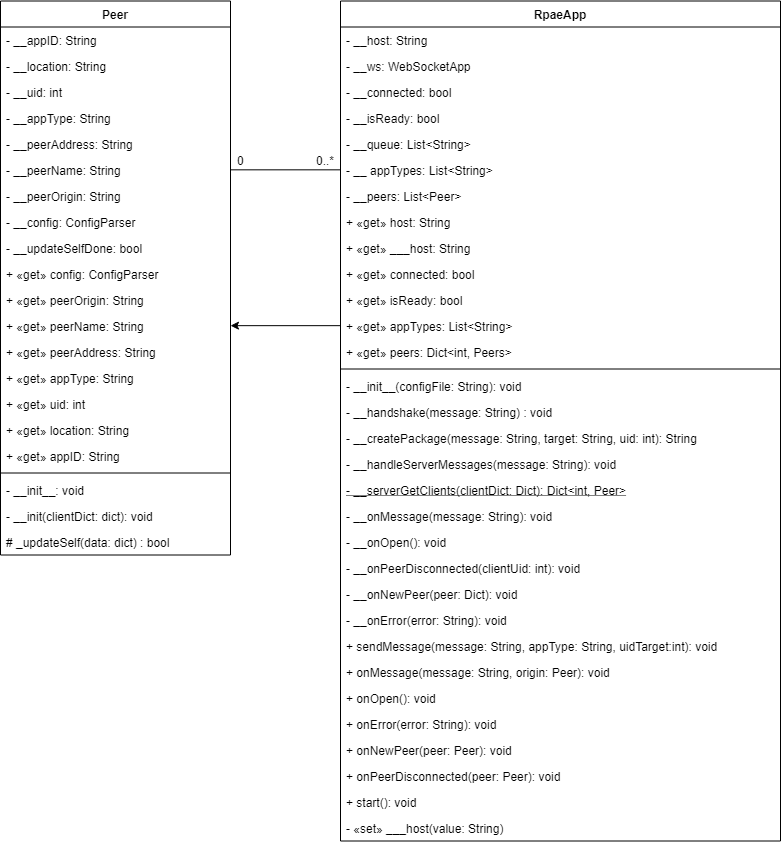
This paragraph describes the events taking place inside the server and how they are all connected.

In Qt there is a system with signals and slots. A signal is as it says, a signal which can be connected in Qt by connect() with a receiver called a slot. In the diagram below you’ll see where all signals end up.



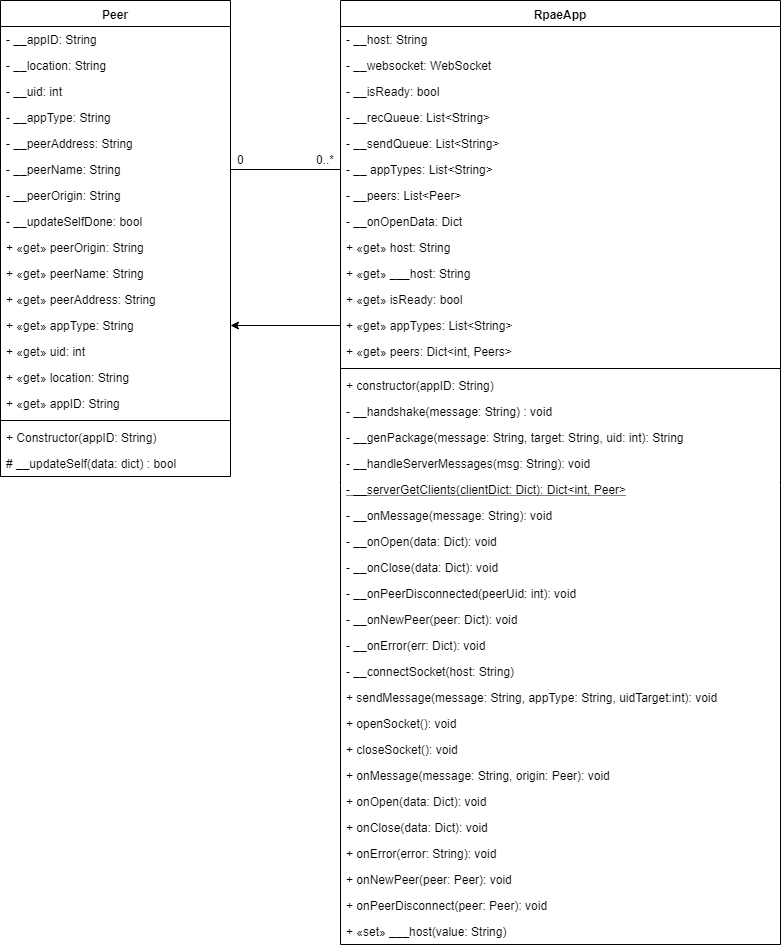
# Server library

The server library is written in python and contains a class that should be inherited by the Server Apps.



# Web client library

The web client library is almost identical to the server app library. This is because we use the same way to connect to the server.



# Apps

## MediaPlayer

### Server

### Webclient

## RpaeAudio

### Server

### Webclient