

Pyspeak Documentation

- **Author** : Nikolaos Bampaliaris
- **Developer:** Nikolaos Bampaliaris

Requirements:

- Python 3
- Tested in python 3.6.1
- This is an open source project, you can do it whatever you want except selling it.
- **You must have in mind that pyspeak is not meant for large server-client applications with millions of connections because it will be very slow. Pyspeak has been designed to help beginners with no network experience create easily basic network applications.**

Contents

New Project.....	4
How Does Pyspeak Work.....	5
Server_main.py.....	6
Exceptions in server_main.py:.....	8
Client_main.py.....	9
Exceptions in client_main.py:.....	11
Chat Box Example.....	14
Run It:.....	17

New Project

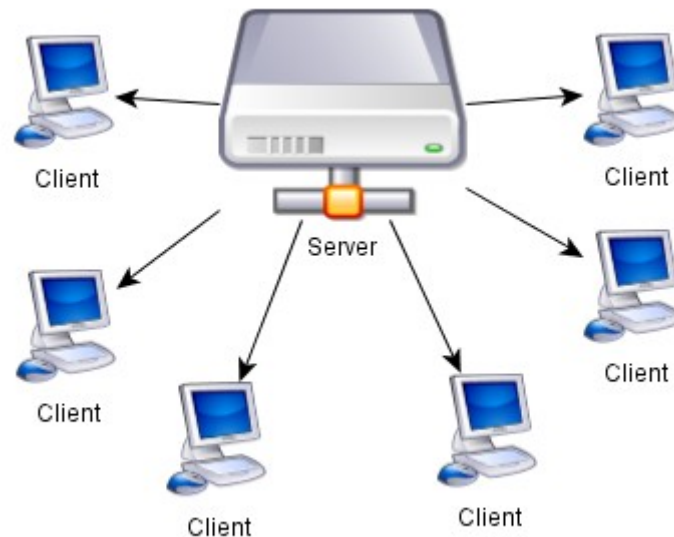
To start a new project is very easy. Open the command prompt on windows or the terminal on linux-osMac and change the directory inside the **pyspeak** folder. Then run:

- **python pyspeak.py project_directory**

This will create a folder with the name **project_directory** and **pyspeak** will copy all the required files.

That's it! You created your first project on **pyspeak!!!**

How Does Pyspeak Work



Every **pyspeak** application needs at least one server and one client in order to work. The server is getting requests from the clients, processing them and sending answers back to the clients.

After creating a project you will notice two main modules. **Server_main.py** and **client_main.py** . These are the modules that you must implement in order to handle the communication between the server and the clients.

That's it! You learn how **pyspeak** works. Now you must learn how to implement those modules.

Server_main.py

You must implement this module in order to write the logic of what your server must do.

You will notice a class with the name Server which inherits pyspeak.Server class. This class has three default methods which you must implement.

```
import pyspeak

class Server(pyspeak.Server):

    def accepted(self, client):
        '''The server is calling this method every time a new client
        connects.'''
        pass

    def connectionClosed(self, client):
        '''The server is calling this method every time a client
        is closing the connection.'''
        pass

    def received(self, client, data):
        '''The server is calling this method every time a client
        sends requests.'''
        pass
```

The accepted(self, client) method:

Pyspeak.Server is calling this method automatically every time a new client connects to the server. The parameter client is a python socket object which you can use in order to identify this client. This method is commonly used to display messages as “A new client has been connected to the server!”

The connectionClosed(self, client) method:

Pyspeak.Server is calling this method every time a client disconnects from the server. The parameter client is a python socket object. This method is commonly used to display messages as “A client has been disconnected from the server.”.

The received(self, client, data) method:

Pyspeak.Server is calling this method every time a client sends data to the server. The parameter client is a python socket object which you can use to identify the client and the data is a tuple with exactly 4 elements. Data[0] is an integer number, data[1] is a floating number, data[2] is a boolean value and finally data[3] is a string with max size 1000 characters. You must implement this method in order to catch data, process them and then send answers back to the clients.

The getClients(self) method:

This is a hidden method inside pyspeak.Server class. You can call this method through a Server object. This method will return a list with all the client sockets which are currently connected on the server.

The send(self, client, integer = 0, real = 0, boolean = False, string = '') method:

This is a hidden method inside pyspeak.Server class. You can call this method through a Server object. This method sends a list with four elements to the client specified. The string has a length limit to 1000 characters.

Exceptions in server_main.py:

```
#Try to run the server.
try:

    #Create a server object.
    server = Server("192.168.1.4", 1996)

    #Run the server.
    server.run()
    pass

#Server could not run, because cpu is full of threads.
except pyspeak.ServerRunError:
    print("Server can't run because cpu is full of threads.")
    pass

#Server could not been established.
except pyspeak.ServerIpError:
    print("Server could not been established. Please check ip and port.")
    pass

#Port is already in use.
except pyspeak.ServerExistsError:
    print("Server could not been establised because the port is already in use.")

#-----Write your code below-----#
```

As you can see, when you try to create and run the server three exceptions can occur.

Pyspeak.ServerRunError:

This exception is been thrown if the computer's CPU is already running the maximum amount of threads it can run.

Pyspeak.Server need's 5 threads in order to run. This exception it's very rare to be thrown but there is always a possibility.

Pyspeak.ServerIpError:

This exception is been thrown if the ip where the server is trying to run is not valid. If this happens, make sure that the ip you provided is true.

Pyspeak.ServerExistsError:

This exception is been thrown if the port where the server is trying to run, is already in use by another application.

Write Your Code Below:

You will notice that there is a comment which says “Write your code below”. In `server_main.py`, the `Server` class and it's methods run in different threads than the module itself. So you can actually write extra code inside the module which will run independent at the same time with `pyspeak.Server`.

That was everything about `server_main.py`! Now you are ready to move to the `client_main.py` module.

Client_main.py

You must implement this module in order to write the logic of what your client must do.

You will notice a class with the name `Client` which inherits `pyspeak.Client` class. This class has three default methods which you must implement.

```

import pyspeak

class Client(pyspeak.Client):

    def connected(self, client):
        '''The client is calling this method when the server
           accepts him (only once).'''
        pass

    def connectionClosed(self, client):
        '''The client is calling this method when the server closes
           the connection (only once).'''

    def received(self, client, data):
        '''The client is calling this method every time the server
           sends requests.'''
        pass

```

The connected(self, client) method:

Pyspeak.Client is calling this method automatically only once when the client connects to the server. The parameter client is a python socket object which you can use in order to identify the server. This method is commonly used to display messages as “I connected to the server!”

The connectionClosed(self, client) method:

Pyspeak.Client is calling this method only once when the server is closing the connection. The parameter client is a python socket object (Its the server itself). This method is commonly used to display messages as “The connection closed by the server.”.

The received(self, client, data) method:

Pyspeak.Client is calling this method every time the server sends data to the client. The parameter client is a python socket object

which you can use to identify the server and the data is a tuple with exactly 4 elements. Data[0] is an integer number, data[1] is a floating number, data[2] is a boolean value and finally data[3] is a string with max size 1000 characters. You must implement this method in order to catch data, process them and then send requests back to the server.

The send(self, integer = 0, real = 0, boolean = False, string = '') method:

This is a hidden method inside pyspeak.Client class. You can call this method through a Client object. This method sends a list with four elements to the server. The string has a length limit to 1000 characters.

Exceptions in client_main.py:

As you can see, when you try to create and run the client, two exceptions can occur.

```

#Try to run the server.
try:

    #Create a server object.
    client = Client("192.168.1.4", 1996)

    #Run the server.
    client.run()
    pass

#Client could not run, because cpu is full of threads.
except pyspeak.ClientRunError:
    print("Client can't run because cpu is full of threads.")
    pass

#Client could not connect.
except pyspeak.ClientConnectError:
    print("Client could not connect to the server. Please check ip and port.")

#-----Write your code below-----#

```

Pyspeak.ClientRunError:

This exception is been thrown if the computer's CPU is already running the maximum amount of threads it can run.

Pyspeak.Client need's 3 threads in order to run. This exception it's very rare to be thrown but there is always a possibility.

Pyspeak.ClientConnectError:

This exception is been thrown if the ip or the port where the client is trying to connect are not valid. If this happens, make sure that the ip and the port you provided are true.

Write Your Code Below:

You will notice that there is a comment which says “Write your code below”. In client_main.py, the Client class and it's methods run in different threads than the module itself. So you can actually write extra code inside the module which will run independent at the same time with pyspeak.Client.

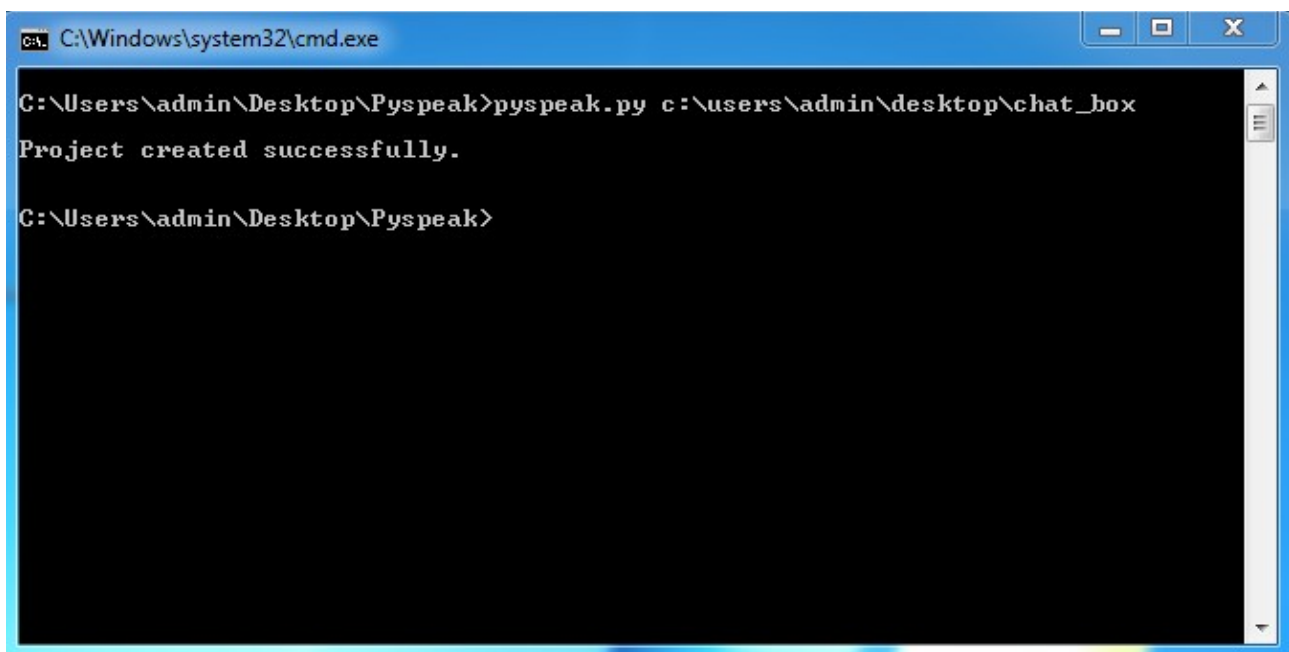
That was everything about client_main.py!

Chat Box Example

Lets first say how the application is going to work.

The client must send a request to the server and tell him to send a message to all the other clients except the client who send the request. Now lets see how we can accomplish that.

First create a pyspeak project:



```
C:\Windows\system32\cmd.exe

C:\Users\admin\Desktop\Pyspeak>pyspeak.py c:\users\admin\desktop\chat_box
Project created successfully.

C:\Users\admin\Desktop\Pyspeak>
```

Implement server_main.py:

```
import pyspeak

class Server(pyspeak.Server):

    def accepted(self, client):
        '''The server is calling this method every time a new client
        connects.'''
        pass

    def connectionClosed(self, client):
        '''The server is calling this method every time a client
        is closing the connection.'''
        pass

    def received(self, client, data):
        '''The server is calling this method every time a client
        sends requests.'''

        #Go through all connected clients.
        for connected_client in self.getClients():

            #Send the message to all clients except the
            #one who send the request.
            if connected_client != client:

                #Send the string.
                self.send( connected_client, string = data[3] )
                pass
            pass
        pass
```

First we are looping through all the clients who are connected to the server (We can use the `getClients()` method to get a list with all the clients). Then we check if the current client is not the client who send the request (message) and if this is True, we send the string which we received (`data[3]`) to the current client.

That's all! The server just receives messages from the client and then sends the messages to all the other clients except the one who actually send the message.

Now lets implement the client_main.py module.

Implement client_main.py:

```
import pyspeak

class Client(pyspeak.Client):

    def connected(self, client):
        '''The client is calling this method when the server
           accepts him (only once).'''
        pass

    def connectionClosed(self, client):
        '''The client is calling this method when the server closes
           the connection (only once).'''

    def received(self, client, data):
        '''The client is calling this method every time the server
           sends requests.'''
        print( data[3] )
        pass
```

The only thing we must do in the received() method is to print the string that the server is sending to us. That's all!

No let's see the code we must write in order to send a message to the other clients.

```
#-----Write your code below-----#

while True:

    #Get the message.
    message = input()

    #Stop the program.
    if message == "/stop":
        client.stop()
        break

    #Else send the message.
    client.send(string = message)
```

Under the comment “Write your code below” go and code this. This is just a while loop where every time you must type something and press enter to continue. If this message == “/stop” then we are closing the connection with the server and ending the

program, else we send the message to the server. That's all! You just created a chat box really easily!

Run It:

First make sure to change the ip and port in the `server_main.py` and `client_main.py` .

After that, run the `server_main.py` module in order to start the server and wait for clients.

Now run two times `client_main.py` in order to start two different clients.

