

Blender Basic Network setup - Boil it down to the basics by “Old Jim” Coulter

Making games and playing them is a lot of fun, but it is even more fun if you can make games to play together with your friends. I will show you how to “cook” a basic network setup in blender. Since Blenders game engine has no build in network support we have to do it in Python, for this you need a matching Python installation for Blender. Take a close look at the terminal or dos-box output when Blender starts, it will tell you which version it needs and if the import worked.

Ingredients:

IP: Unique Computer name

 LAN IP: Only accessible inside the local network

 Internet IP: Accessible form all over the world (when routers and firewalls allows it)

Port: Access line to a specific communication

Socket: Transmitting and receiving station.

Libraries: Importing cookbooks (modules) from the Python Library gives blender info on how to prepare a special meal

Logic Bricks: central switching station of the Blender game engine.

Objects: We will use 2 cubes which we can move around and these movements we will send over the Network.

If you load the file `WSAG-BasicNetwork.blend` you can find 2 scripts in there which are relevant to our network setup. (Server.py and Client.py). These scripts show what you get if you boil a network down to its basics. So what do they do?

Server.py:

```
#-----SETUP-----#
1.from GameLogic import *
2.from socket import *
3.from cPickle import *
4.cont = GameLogic.getCurrentController()
5.obj = cont.getOwner()

6.if obj.OneTime == 0:
7. Host = ''
8. ServerPort = 10000
9. GameLogic.sServer = socket(AF_INET,SOCK_DGRAM)
10.     GameLogic.sServer.bind((Host,ServerPort))
11.     GameLogic.sServer.setblocking(0)
12.     obj.OneTime = 1

13.PosYou = obj.getPosition()

14. scene = getCurrentScene()
15.Client = scene.getObjectList()["OBClient"]

16.PosClient = [0,0,0]
```

```

#-----RECEIVE/SEND-----#
17.try:
18.    Data, CLIP = GameLogic.sServer.recvfrom(1024)
19.    UPData = loads(Data)
20.    PosClient = [UPData[0],UPData[1],UPData[2]]
21.    Client.setPosition(PosClient)
22.    Data = dumps((PosYou))

23.    GameLogic.sServer.sendto(Data,CLIP)
24.except:
25.    pass

```

```
1.from GameLogic import *
```

From the library we `import` the book (module) called `GameLogic`. By importing this book we have access to all the information about the blender game engine.

```
2.from socket import *
```

Gives us all the information about network socket setups

```
3.from cPickle import *
```

Gives us information on how to pack data in to a container (similar to a zip program)

```
4.cont = getCurrentController()
```

This command gets the `Controller` for the script and saves it to the variable `cont`. The logic brick we just made accessible, is the `Controller` that gave the order to execute the python script.

```
5.Obj = Cont.getOwner()
```

Gets us the owner of this script. That would be the object that owns the logic brick.

```
6.if Obj.OneTime == 0:
```

```
12.    Obj.OneTime = 1
```

Line 6 calls up the property `OneTime` and if its value is equal to `0` it will execute the lines below that are indented. Line 12 will change the value of the property `OneTime` to `1`. By changing the property we make sure that the indented lines below line 5 only run one time.

```
7. host = ''
```

Saves a empty text holder in to the variable `host`. The purpose of this empty text holder is to make a place where later the computer name can be written in to. So you may ask; why not write the computer name in to it right away? Well you could do that but then this script would only run on your own computer. So that's why we leave it empty and let the socket fill it out later all by it self. You also would have the option to write localhost between the `,` and `'`, that would say the script right a way to enter the name of the local computer in there... But why write more then necessary?

```
8. ServerPort = 10000
```

Saves the value 10000 in to the variable `ServerPort`. This value will be the port number over witch the socket will communicate. That's why you will have to make sure that your firewall and your router won't block this and the other ports that will be added later. Information on how you can setup your firewall and router to forward information over this port, can be found in the manuals of your firewall and router. If any questions or problems come up then you can ask them in the gameblender.org forum.

```
9. GameLogic.sServer = socket(AF_INET, SOCK_DGRAM)
```

This code will create a socket that uses the protocol that is defined inside of the brackets and saves it to the global variable `GameLogic.sServer`.

`AF_INET, SOCK_DGRAM` is the definition for the UDP (User Datagram Protocol) protocol. We will use this protocol because it is very fast and will keep on working even if it once can't send or receive a packet.

```
10. GameLogic.sServer.bind( (Host, ServerPort) )
```

The code `.bind((host, Serverport))` is used to bind the information that we have saved in to variable `host` and `Serverport` to the socket. Now the socket knows on what computer and over which port he should communicate.

```
11. GameLogic.sServer.setblocking(0)
```

The socket will already work with the lines above. But as I already mentioned by line 8 we would like to make sure that the script will keep on running even if the socket did not send or receive a packet. This can happen very easy, if the connection having a problem, the client and server have different streaming rates the connection is temporary used by a other program... That's why we will use the code `.setblocking(0)`. The `0` in the brackets will tell the script that it should not block even if it did not send or receive anything. If you would put a `1` there, the script would try to receives/send a packet until it has had success. If you would do this everything would be as slow as the slowest client that is connected with the server.

```
13.PosYou = obj.getPosition()
```

Gets the position of servers cube (the server controls the blue cube)

```
14. scene = getCurrentScene()
```

```
15.Client = scene.getObjectList() ["OBClient"]
```

The command `getCurrenScene()` will get all the elements of the current scene in to the script. These elements will be saved in the variable `scene`. The variable `scene` contains all the elements of the current scene, now we would like to have access to all the Objects in the scene, this will happen by using the command `.getObjectList()`. In the square brackets we write the name of the object that we would like to save to the variable `objPump1`. Its important that you put the letters "OB" in front of the objects name.

```
16.PosClient = [0,0,0]
```

In the variable `PosClient` we provide a list with 3 elements. This list will be later filed with the coordinates of the player. The 3 elements `[0,0,0]` stand for the X, Y and Z coordinates.

```
17.try:
```

Line 17 starts to try to execute the indented lines below if one of the lines fail lines 24 and 25 come in to action telling the script to just pass on. The only line below that actually can fail is the Line 18. It will fail when it tries to receive and there is no data there to receive. This can for example happen if no Client is sending any data.

```
18. Data, CLIP = GameLogic.sServer.recvfrom(1024)
```

The variable `GameLogic.sServer` contains the information about the socket. The code `.recvfrom` commands the socket to receive data. `(1024)` defines the maximal size, that the buffer can receive at once.

Every time we receive data we get 2 blocks. The first block contains the data and is saved in the variable `Data`. The second block contains sender address and is saved in the variable `CLIP` (= Client IP)

```
19.      UPData = loads(Data)
```

Now we will use the methods from the `cPickle` module for the first time. The received data will be unpacked and saved to the variable `UPData`, As you will see in line 22, data that is send over the network is first packed in to a container.

```
20.      PosClient = [UPData[0],UPData[1],UPData[2]]
```

The received und now unpacked data is saved in to the list we made in line 16.

```
UPData[0] = Position on the world axis X
```

```
UPData[1] = Position on the world axis Y
```

```
UPData[2] = Position on the world axis Z
```

```
21.      Client.setPosition(PosClient)
```

The list we just have filed out is now used to set the position of the clients player.

```
22.      Data = dumps((PosYou))
```

The command `dumps` will pack any data inside the double brackets in to a container and save it to the variable `Data`. Important: You have to use double brackets here “`((PosYou))`” and not single brackets “`(PosYou)`”. The Variable `PosYou` that we are packing here contains the information of the coordinates of the Server cube (blue) see line 13.

```
23.      GameLogic.sServer.sendto(Data,CLIP)
```

To send `Data` we use `GameLogic.sServer` to call up the socket and give it the command `.sendto`. To send data do it just as in line 18, you need 2 things: The data and the address that this packed should be send to. This information you put inside the brackets.

```
24.except:
```

```
25.      pass
```

These lines work together with line 17 and already where explained.

The client `script.py` is very similar to the `server.py` script, the only real difference is that it sends data to the server first, so it dose not have an already received package that contains the servers address; so this will have to be entered manually. We do this in the script `setup.py` line 10. By entering the servers IP address there behind the global variable `GameLogic.IP`. If you now study the script below you may say he way not just write it in to line 7 instead of putting the global variable `GameLogic.IP` there. You are right, you can do so but by having the IP entering in the `setup.py` script it makes it easier to find it. Also the `setup.py` script contains other informations that help a user to start this program in any network. See lines that are behind a comment mark “`#`”. They are not part of the script, but give you information on how to use this script. So that’s why we will have the 3D window on the left side and the `setup.py` script on the right side when we open the `BasicNetwork.blend` file.

Client.py:

```
#-----SETUP-----#
1.from GameLogic import *
2.from socket import *
3.from cPickle import *
4.cont = GameLogic.getCurrentController()
5.obj = cont.getOwner()

6.if obj.OneTime == 0:
7.  ServerIP = GameLogic.IP
8.  Serverport = 10000
```

```

9. Clientname = ''
10.     ClientPort = 10001
11.     GameLogic.sClient = socket(AF_INET,SOCK_DGRAM)
12.     GameLogic.sClient.bind((Clientname,ClientPort))
13.     GameLogic.host = (ServerIP,Serverport)
14.     GameLogic.sClient.setblocking(0)
15.     obj.OneTime = 1

16.PosYou = obj.getPosition()

17.scene = getCurrentScene()
18.Server = scene.getObjectList()["OBServer"]

19.PosServer = [0,0,0]

#-----RECEIVE/SEND-----#
20.Data = dumps((PosYou))
21.GameLogic.sClient.sendto(Data,GameLogic.host)

22.try:
23.     Data, SRIP = GameLogic.sClient.recvfrom(1024)
24.     UPData = loads(Data)
25.     PosServer = [UPData[0],UPData[1],UPData[2]]
26.     Server.setPosition(PosServer)
27.     Server.setOrientation(OriServer)
28. except:
29.     pass

```

Since all the commands in this script were explained in the server.py script I will only give a short summary on these lines of code:

Lines 1 – 5: Basic setup to access the Object, its properties and logic bricks

Lines 6 + 15: Make sure that the intended lines below Line 6 only run once.

Line 7: Saves the global Variable that contains the IP to a local Variable.

Lines 8 – 12: Setup the Socket

Line 13: Saves the complete Server address to a variable.

Line 14: Tells the script to just keep running even if no data was send/received.

Line 16: Gets the position of the red cube.

Line 17 + 18: Make the blue cube accessible by this script.

Line 19: Makes a empty list, that will be uses to set the servers cube (blue cube)

Line 20 + 21: Packs your position in a container and sends it to the server.

Line 22, 28,29: Will try to run the intended lines below line 22, if they fail the script will just pass on.

Appetite for more

I hope this little starter whetted your appetite for more.

If you take a closer look at:

-WSAG-PumpkinRun-Tutorial.pdf

-WSAG-PumpkinRun.blend

These files will give you more detailed Information about the WSAG Network.

Also a lot of information about more complex network „meals“ can be found at www.wsag.ch.vu.

If any questions come up you can ask them at the gameblender.org forum.