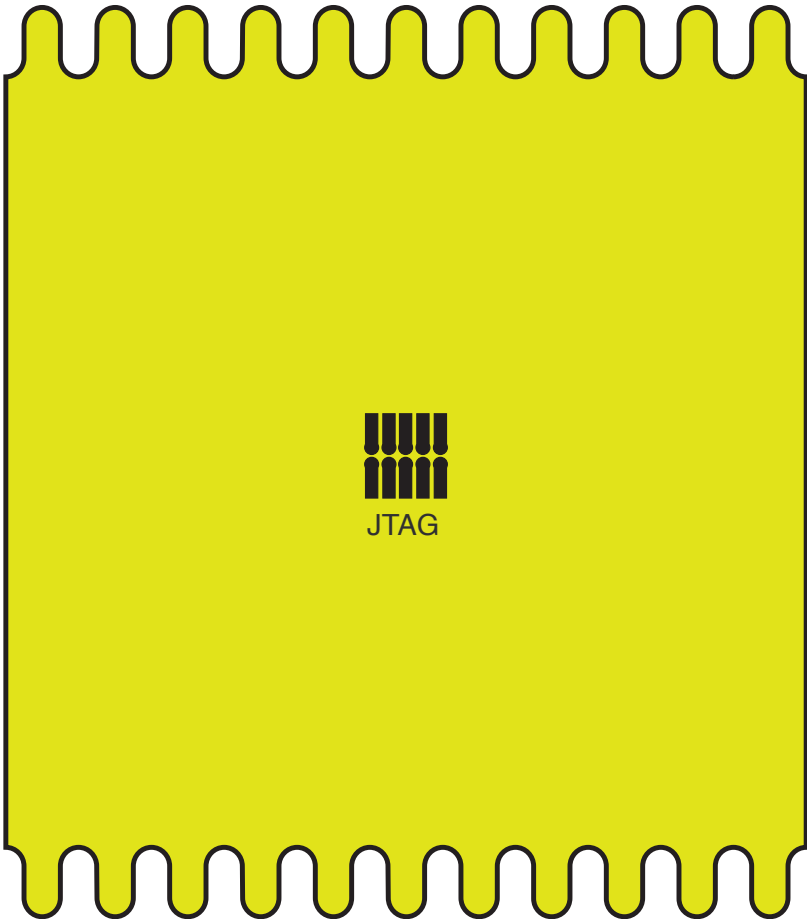
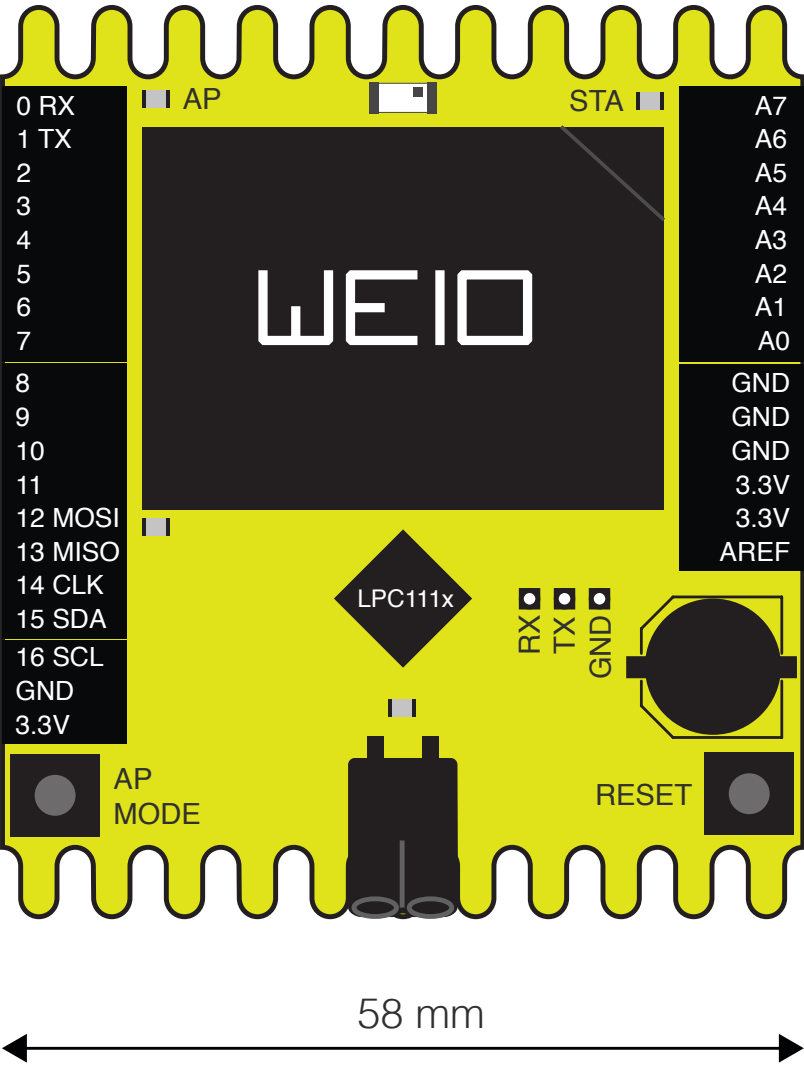
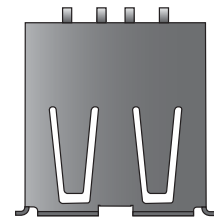


Web of Things Platform

Weio looks like this

recto

verso

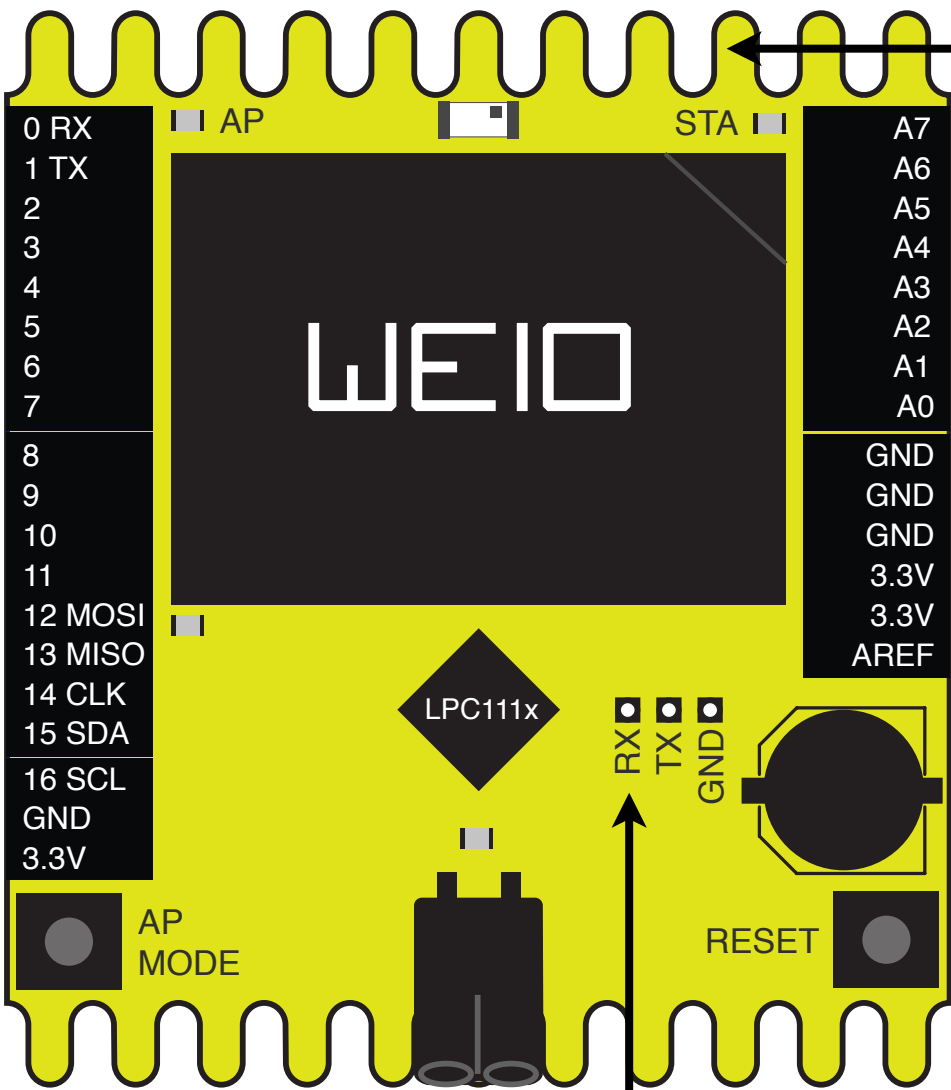


USB is too big to fit  
Finally we don't need it

JTAG for LPC & carambola2  
pogo pins or similair header

Weio looks like this

VIA LPC111x  
USER GPIO  
USER RXTX  
USER SPI  
USER i2c



This form is  
for screws

USER ADC  
From 0 to 3.3V

ADC AREF is  
optional. Only if  
possible

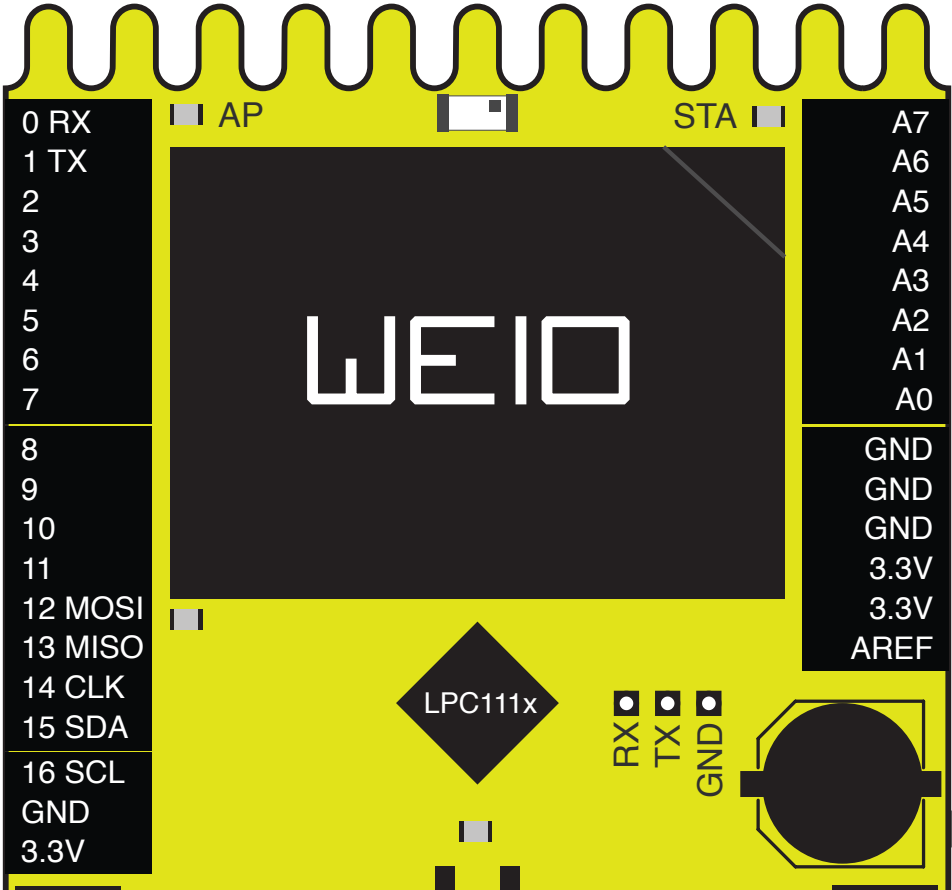
Carambola RXTX  
For serious problem debugging

Our recommendation

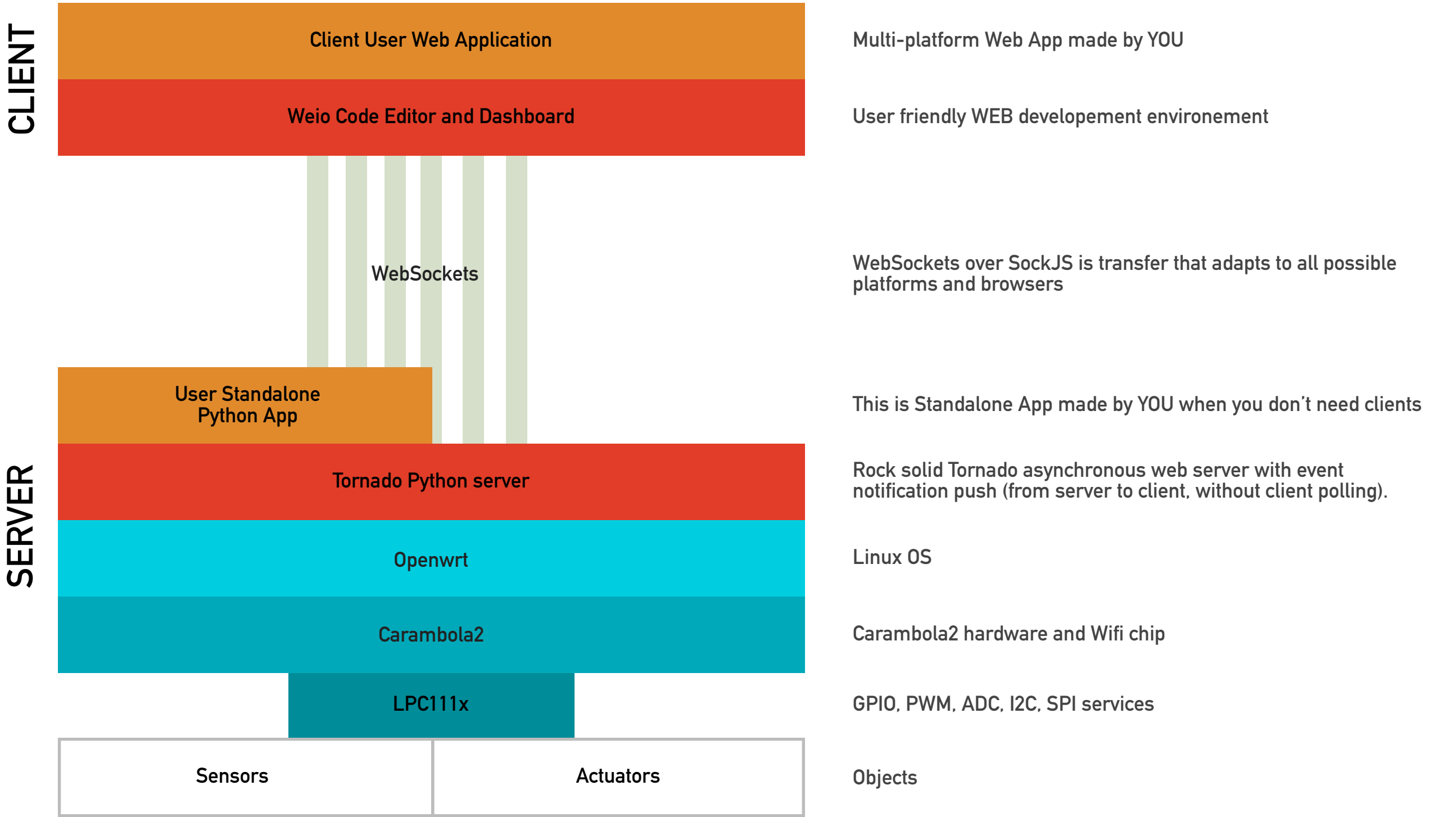
Table 2. Ordering options

Type number	Series	Flash	Total SRAM	Power profiles	UART RS-485	I <sup>2</sup> C/ Fast+	SPI	ADC channels	GPIO	Package
LPC1110										
LPC1110FD20	LPC1100L	4 kB	1 kB	yes	1	1	1	5	16	SO20
LPC1111										
LPC1111FDH20/002	LPC1100L	8 kB	2 kB	yes	1	1	1	5	16	TSSOP20
LPC1111FHN33/101	LPC1100	8 kB	2 kB	no	1	1	1	8	28	HVQFN33
LPC1111FHN33/102	LPC1100L	8 kB	2 kB	yes	1	1	1	8	28	HVQFN33
LPC1111FHN33/103	LPC1100XL	8 kB	2 kB	yes	1	1	2	8	28	HVQFN33
LPC1111FHN33/201	LPC1100	8 kB	4 kB	no	1	1	1	8	28	HVQFN33
LPC1111FHN33/202	LPC1100L	8 kB	4 kB	yes	1	1	1	8	28	HVQFN33
LPC1111FHN33/203	LPC1100XL	8 kB	4 kB	yes	1	1	2	8	28	HVQFN33

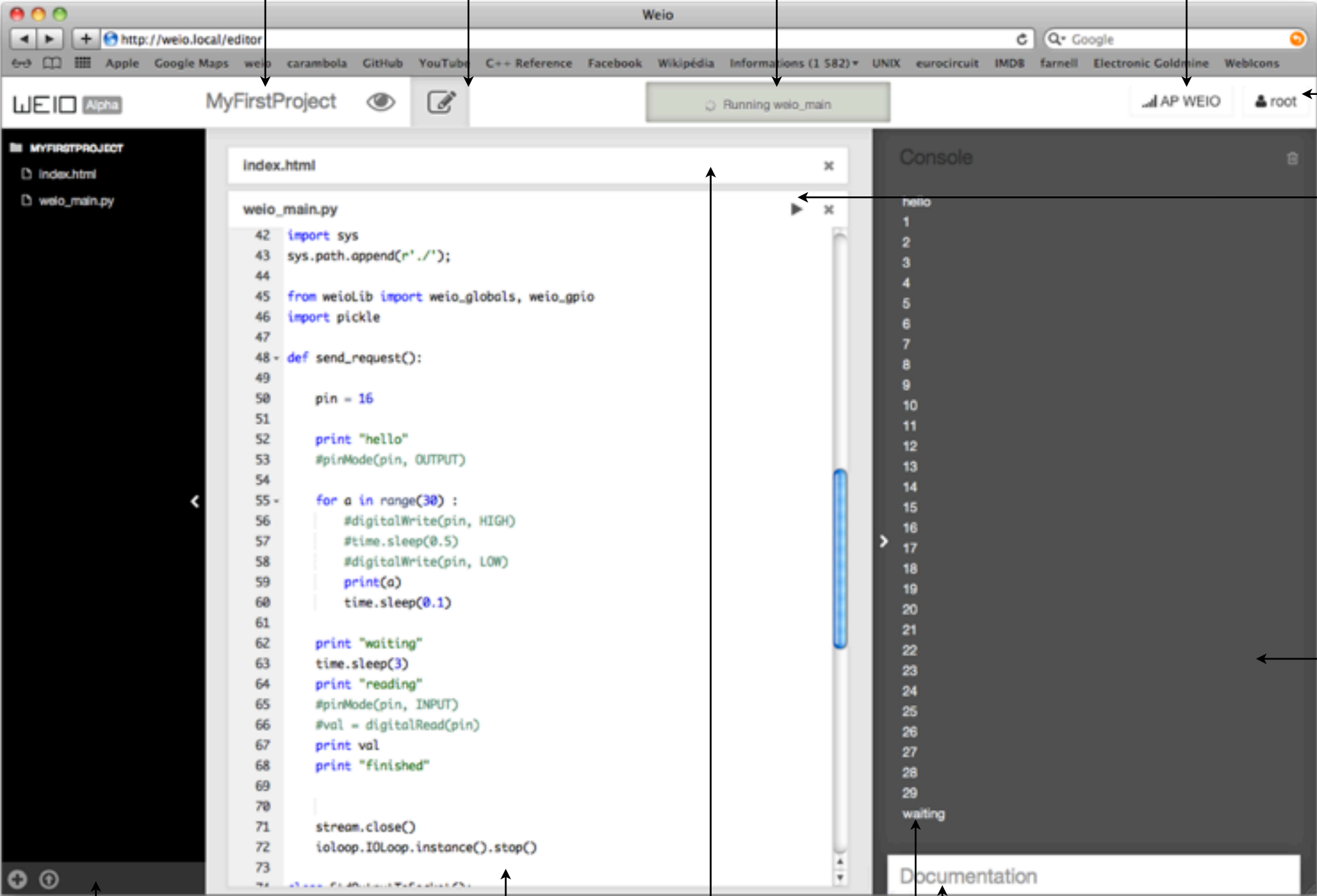
function	number of pins
User GPIO	11
User SPI	3
User i2c	2
User ADC	8
Carambola2 Reserverd <b>SPI</b>	4
Total	28



Weio architecture from Web App to LEDs



Current project      Preview & Editor mode      weio speaks to me - status monitor      List of available Wifi networks and it's current state



User global settings :  
Bonjour name, password, wifi settings, time zone, python plugins, system update, available space, processes killer

Play button runs weio\_main python program for standalone applications. Asynchronous communication manner, non blocking for Editor application or other processes.

For pure web applications it's not needed to run anything, Preview button permits to see web app in action

lateral sidebars can be minimized to gain more place for code editor

files in current project  
new files can be created or added

project can be bundle and  
downloaded to PC

Ace Ajax code editor  
python, html, css, json

Weio API has  
colored syntax

Files are in stacks  
that can be opened or  
closed

Direct console output from weio  
Documentation is called in the same window

## Sneak peak - functional version of software - Editor web app

This code has been runned directly from inside Carambola2 without external hardware or software.

All IDE, web apps and server are included inside Carambola2

No software installation needed on pc or weio, everything is included and ready to go. Internet is not needed to run this application, embedded software and AP mode is self sufficient.

Websockets establish realtime communication between server and clients

Weio has automatic wifi network detection and easy interface for it's configuration

Weio application is contained in flash memory and takes about 700Kb of space. At boot application is decompressed to RAM and runned.

This is not final design, it's draft document and just a screen shoot of one iteration in weio project



## Weio is opensource project - WEIO GPL header

index.html

weio\_main.py

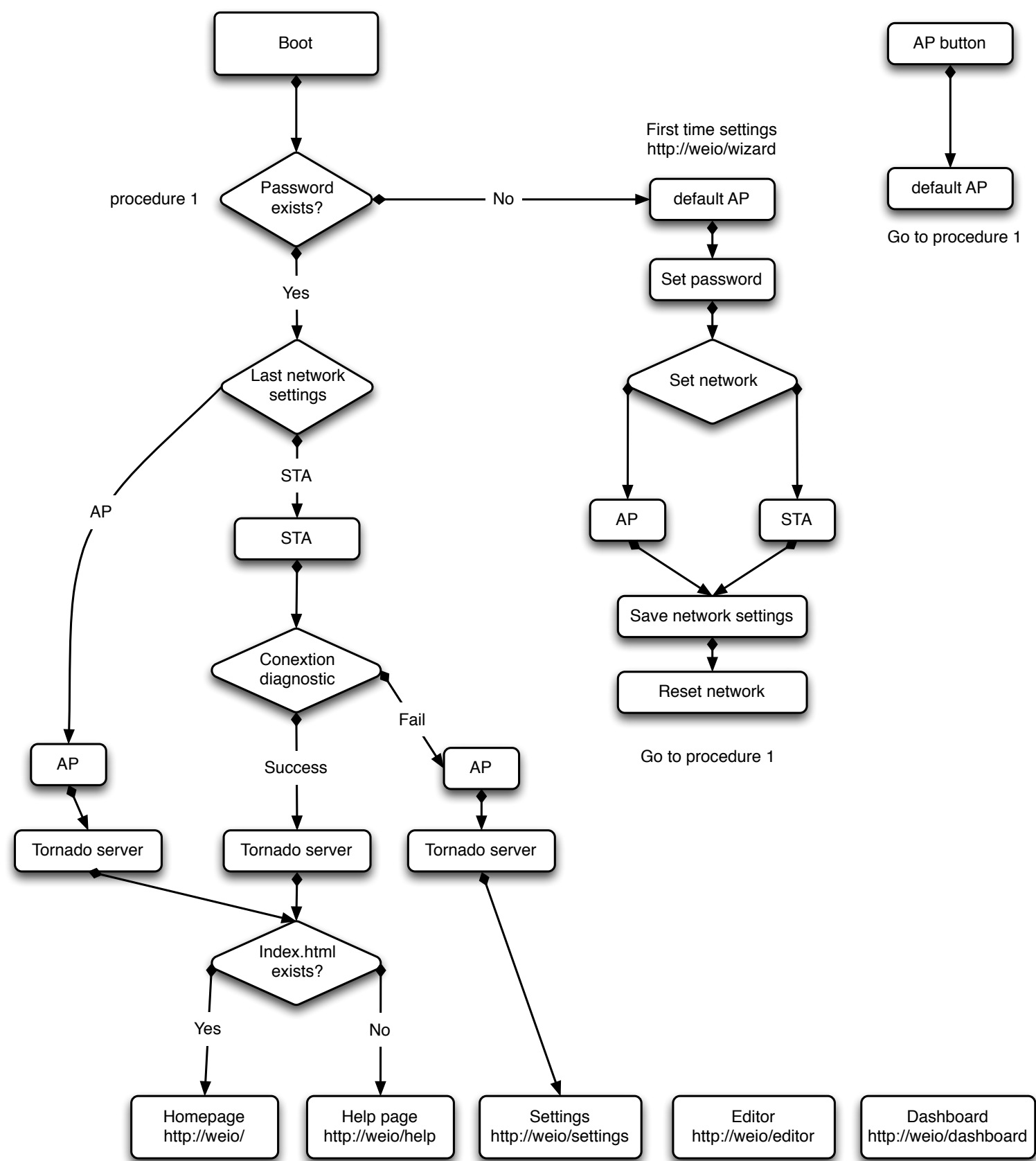
```
2 #
3 # WEIO Web Of Things Platform
4 # Copyright (C) 2013 Nodesign.net, Uros PETREVSKI, Drasko DRASKOVIC
5 # All rights reserved
6 #
7 #          ##          ## ##### #####
8 #          ## ## ## ##          ## ## ##
9 #          ## ## ## ##          ## ## ##
10 #          ## ## ## #####          ## ## ##
11 #          ## ## ## ##          ## ## ##
12 #          ## ## ## ##          ## ## ##
13 #          ### ### ##### #####
14 #
15 #          Web Of Things Platform
16 #
17 # This file is part of WEIO
18 # WEIO is free software: you can redistribute it and/or modify
19 # it under the terms of the GNU General Public License as published by
20 # the Free Software Foundation, either version 3 of the License, or
21 # (at your option) any later version.
22 #
23 # WEIO is distributed in the hope that it will be useful,
24 # but WITHOUT ANY WARRANTY; without even the implied warranty of
25 # MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
26 # GNU General Public License for more details.
27 #
28 # You should have received a copy of the GNU General Public License
29 # along with this program. If not, see <http://www.gnu.org/licenses/>.
30 #
31 # Authors :
32 # Uros PETREVSKI <uros@nodesign.net>
33 # Drasko DRASKOVIC <drasko.draskovic@gmail.com>
```

Console

hello  
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13  
14  
15  
16  
17  
18  
19  
20  
21  
22  
23  
24  
25  
26  
27  
28  
29  
waiting  
reading  
reading



Configurator diagram & first time boot



Production and editing

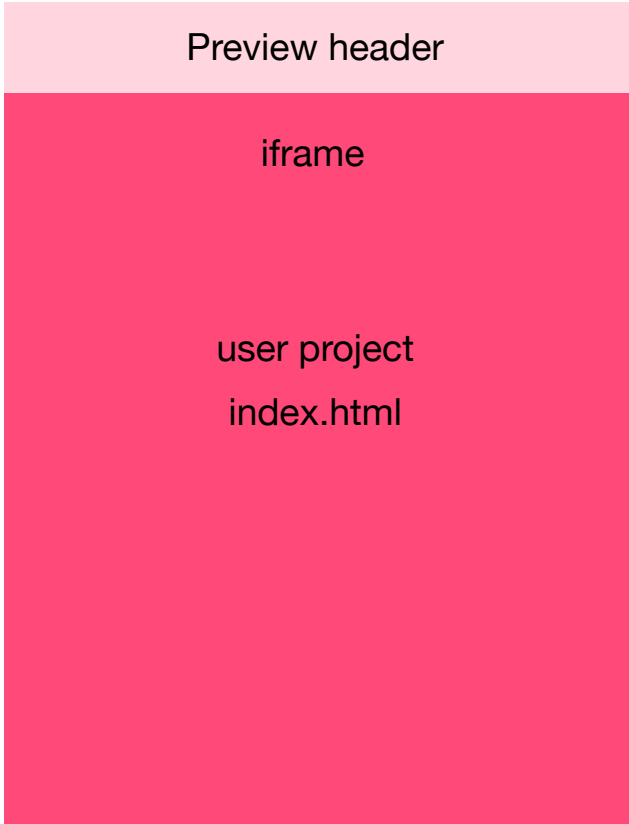


address

<http://weio.local>

<http://weio.local/preview>

<http://weio.local/editor>



internal path










static/user\_weio/index.html

static/user\_weio/preview.html










editor/index.html






# WEIO Directory structure

## Root overview

 clientDependencies	Libraries & images necessary for client webApp
 editor	Editor HTML code & Editor Python server side code
 preview	Preview HTML code
 productionScripts	Automatisation for bundle creation and install script on hardware
 README.md	Project overview
 server.py	Tornado server, Weio starts here
 uds_weio_main	Unix domain socket created by Tornado server and standalone client
 userProjects	Projects made by users and embedded examples
 weioLib	Weio API standalone library

## Weio standalone API

 .DS_Store	
 .git	
 .gitignore	
 clientDependencies	
 editor	
 preview	
 productionScripts	
 README.md	
 server.py	
 uds_weio_main	
 userProjects	
 weioLib	

 __init__.py	
 UnixSocketClient.py	Not used at this moment, work in progress
 weio_client.py	Tornado client for communication between Tornado server and weio_main.py user standalone
 weio_globals.py	Weio global words, low level oriented
 weio_gpio.py	Driving GPIO from user space
 WeioAPIbridge.py	Websocket bridge between standalone webApp and Tornado server
 WeioFiles.py	Work with files, open, save, list directories, sort extensions,...

# WEIO Communication Protocol

Ask something server :

**‘request ‘ : ‘something’**

**‘request ‘ : ‘something’, ‘data’, ‘blabla’**

Server responds :

**‘requested’ : ‘something’, ‘data’ : ‘justforyou’,  
‘status’ : ‘i m bored, ask me more’**

Server has something to say :

**‘serverPush ‘ : ‘doltRight’, ‘data’:’ buy a milk’,  
‘status’ : ‘i m thirsty’**

## WEIO Client to server communication

editor.html

making one request

```
var askForFileListRequest = { "request": "getFileList" }  
baseFiles.send(JSON.stringify(askForFileListRequest));
```

pack instruction in 'request'

editor.py

receiving on the server

```
def serve(self, request) :  
    global weio_main  
  
    # parsing strings from browser  
    rq = ast.literal_eval(request)  
  
    # answer dictionary object  
    data = {}  
  
    if 'getFileList' in rq['request'] :  
        # echo given request  
        data['requested'] = rq['request']  
  
        # read all files paths from user directories  
        data['data'] = WeioFiles.scanFolders()  
        # notify what is happening at this moment  
        data['status'] = "I'm ready, gimme some awesome cod  
        fileList = data  
  
    #sending  
    self.send(json.dumps(data))
```

finding message

echo request with results

pack answer in 'data'

dump and send



## WEIO Client to server communication + data

editor.html

making one request + additional data

```
var askForFileListRequest = { "request": "getFile", "data": newData};  
baseFiles.send(JSON.stringify(askForFileListRequest));
```

editor.py

receiving on the server

```
elif 'getFile' in rq['request'] :  
  
    # echo given request  
    data['requested'] = rq['request']  
  
    # echo given data  
    data['requestedData'] = rq['data']  
  
    fileInfo = rq['data']  
  
    pathname = fileInfo['path']  
  
    rawFile = WeioFiles.getRawContentFromFile(pathname)  
  
    fileInfo['data'] = rawFile  
  
    data['data'] = fileInfo  
  
    self.send(json.dumps(data))
```

## WEIO Server to client communication - response to instruction

editor.html

Server is responding to instruction

```
if 'getFileList' in rq['request'] :-  
    # echo given request  
    data['requested'] = rq['request']  
    # read all files paths from user directories  
    data['data'] = WeioFiles.scanFolders()  
    # notify what is happening at this moment  
    data['status'] = "I'm ready, gimme some awesome code!"  
    fileList = data  
    #sending  
    self.send(json.dumps(data))
```

editor.py

Server push is caught

```
baseFiles.onmessage = function(e) {-  
    //console.log('Received: ' + e.data);  
    // JSON data is parsed into object  
    data = JSON.parse(e.data);  
    // switch  
    if ("requested" in data) {-  
        // this is instruction that was echoed from server + data as response  
        instruction = data.requested;  
        if (instruction == "getFileList") {-  
            fileList = data.data;  
            console.log(fileList.allFiles);  
  
            editorData.tree = fileList.allFiles;  
            initEditor();  
        }  
        // install first index.html  
        addNewStrip("index.html");  
    }  
}
```

## WEIO Server to client communication - spontaneous message from server

editor.html

Server is sending spontaneous message

```
data = {}  
  
if 'stdout' in rcvd :  
    data['serverPush'] = 'stdout'  
    data['data'] = rcvd['stdout']  
    data['status'] = "Check output console"  
elif 'stderr' in rcvd :  
    data['serverPush'] = 'stderr'  
    data['data'] = rcvd['stderr']  
  
self.send(json.dumps(data))  
-
```

editor.py

Client is catching spontaneous messages

```
} else if ("serverPush" in data) {  
    demand = data.serverPush;  
    if (demand == "stdout") {  
  
        stdout = data.data;  
  
        consoleData.push(stdout);  
  
        if (consoleData.length > MAX_LINES_IN_CONSOLE) {  
            consoleData.shift();  
        }  
  
        consoleOutput = "";  
  
        for (var i=0; i<consoleData.length; i++) {  
            consoleData[i] = consoleData[i].replace("\n", "<BR>");  
            consoleOutput+=consoleData[i];  
        }  
        // this function outputs console to screen  
        $('#consoleOutput').html(consoleOutput);  
    }  
}
```



## WEIO Messages notification system

editor.html  
server push

```
data['status'] = "weio_main.py is running!"  
self.send(json.dumps(data))
```

editor.py  
client on\_message

I'm ready, gimme some awesome code!

```
get Status message from server  
if ("status" in data) {  
    setStatus(null, data.status);  
}
```

render function in javascript

```
// This function sets corresponding icon and message inside statusBar in the middle of header. Icon is string format defined in font-  
// awesome library, message is string format  
// If icon is not desired you can pass null as argument : setStatus(null, "hello world");  
  
// Icons are only used when synchronization is active or weio_main is running  
// set status is always activated from server push messages, never from client, except when closed socket is detected!  
  
function setStatus(icon, message) {  
  
    if (icon!=null) {  
        $( "#statusBar" ).html('<p id="statusBarText"><i id="statusIcon" class="" + icon + ""></i> + message + '</p>');  
    }  
    else {  
        $( "#statusBar" ).html('<p id="statusBarText"> + message + '</p>');  
    }  
}
```

# WEIO Configuration file

**conf file is not yet  
deployed everywhere is  
needed**

## WEIO Configuration file source code

```
# configuration file
import json
```

```
def getConfiguration():
    inputFile = open("config.weio", 'r')
    rawData = inputFile.read()
    inputFile.close()
    return json.loads(rawData)
```

get config file

```
def saveConfiguration(conf):
    inputFile = open("config.weio", 'w')
    print(inputFile)
    ret = inputFile.write(json.dumps(conf))
    inputFile.close()
```

write config file

```
#example & test configuration
# weio_config = {}
# weio_config['user_projects_path'] = 'userProjects/'
# weio_config['last_opened_project'] = 'myFirstProject/'
# weio_config['last_opened_files'] = ['index.html', 'weio_main.py']
# weio_config['editor_html_path'] = 'editor/editor.html'
# weio_config['preview_html_path'] = 'preview/preview.html'
# weio_config['dependencies_path'] = 'clientDependencies'
# weio_config['weio_lib_path'] = 'weioLib'
# weio_config['absolut_root_path'] = '/tmp/weio'
# weio_config['port'] = 8081
# weio_config['ip'] = '0.0.0.0'
#
# #
# saveConfiguration(weio_config)
# a = getConfiguration()
# print a['user_projects_path']
```

config file keys



WEIO Configuration file is everywhere - python or javascript just call it

server.py

from python

```
# IMPORT BASIC CONFIGURATION FILE ALL PATHS ARE DEFINED INSIDE
from weioLib import weio_config

# Take configuration from conf file and use it to define parameters
global configFile
configFile = weio_config.getConfiguration()

# put absolut path in conf, needed for local testing on PC
configFile['absolut_root_path'] = os.path.abspath(".")
weio_config.saveConfiguration(configFile)
```

preview.html

from javascript

config.json is symlinked to config.weio

```
<script>
var configFile = "";

$.getJSON('config.json', function(data) {
    configFile = data;

    $(".iframeContainer").attr("src", "userProjects/" + configFile.last_opened_project + "index.html");
    // console.log(configFile.weio_lib_path);
});

</script>
```