

Embedded System Design Techniques™

Designing IoT Sensor Nodes using the ESP8266

Session 5: Device Management and the Automated Universe

July 14th, 2017

Jacob Beningo

Course Overview

Topics:

- The IoT Architecture
- Getting Started with the ESP8266
- Interfacing Sensors to the ESP8266
- Connecting the ESP8266 to the internet
- **Device Management and the Automated Universe**

Session Overview

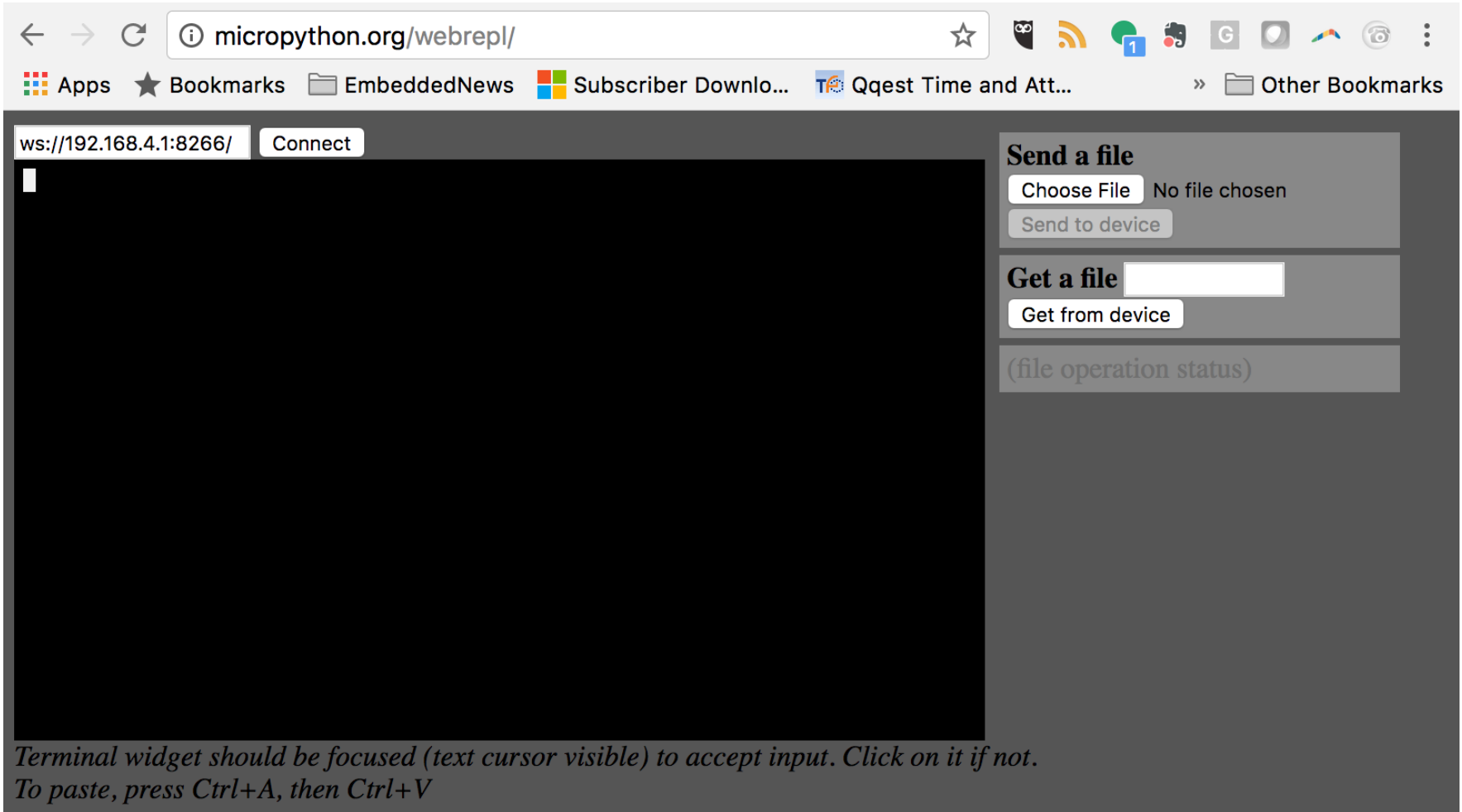
- Device Management
- Firmware Updates
- More Micro Python



Device Management & Configuration

- Firmware Updates
- Kernel Updates
- Device Configuration
 - Security
 - Application Settings
 - Discovery

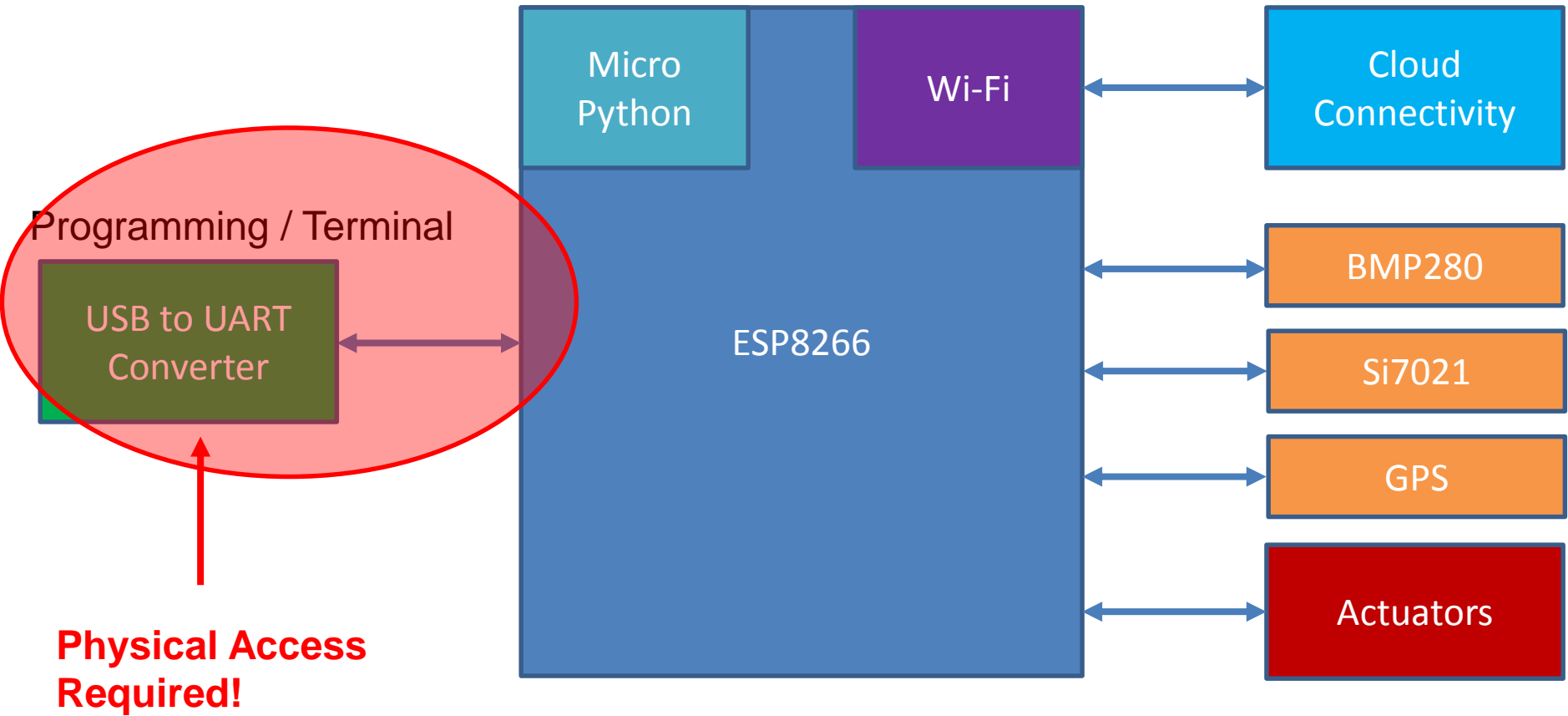
Updating the ESP8266 Remotely



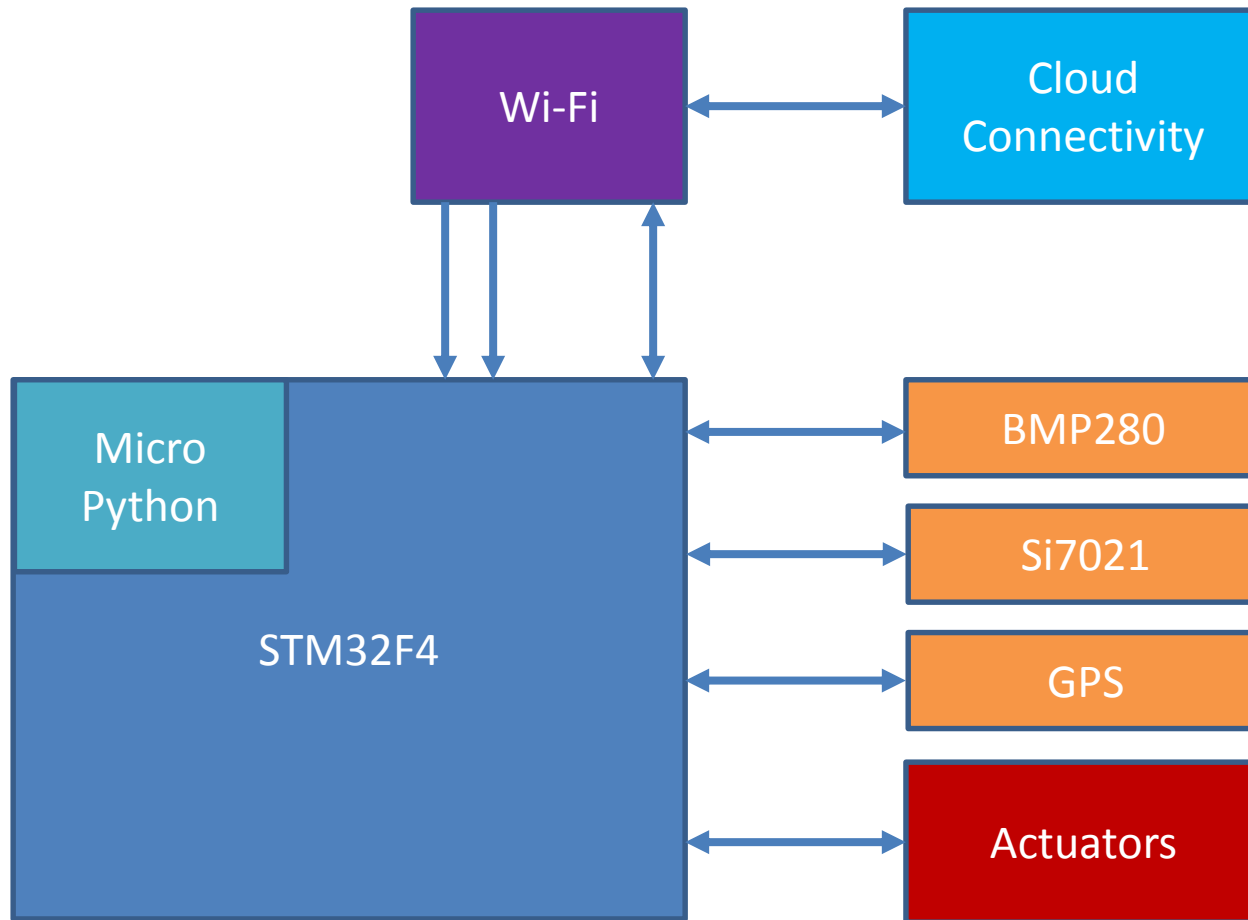
The screenshot shows a web browser window at micropython.org/webrepl/. The browser's address bar and bookmarks are visible at the top. The main content area features a terminal window on the left with a dark background and a white cursor. Above the terminal is a text input field containing 'ws://192.168.4.1:8266/' and a 'Connect' button. To the right of the terminal are two panels: 'Send a file' with a 'Choose File' button and 'Send to device' button, and 'Get a file' with a text input field and a 'Get from device' button. Below these panels is a '(file operation status)' label.

*Terminal widget should be focused (text cursor visible) to accept input. Click on it if not.
To paste, press Ctrl+A, then Ctrl+V*

Kernel Updates

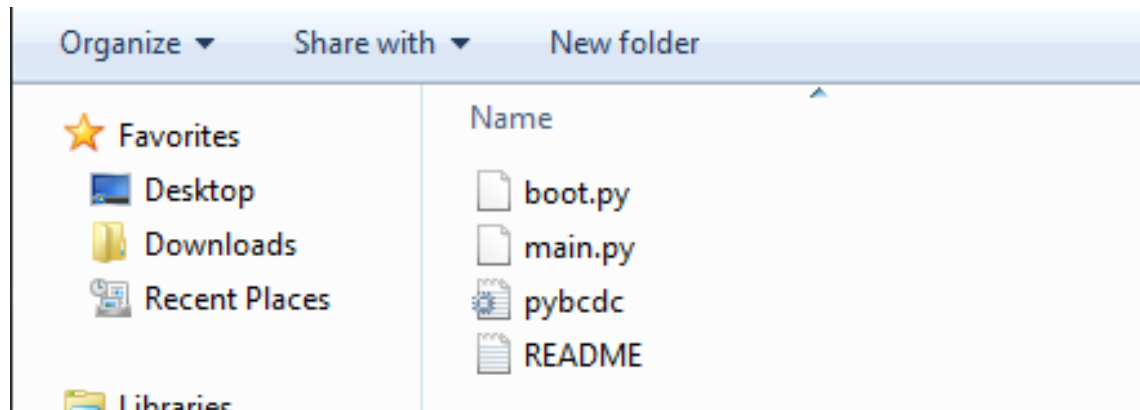


Kernel Updates - Alternative



Firmware Updates

- Two different methods
 - Copy and paste updated scripts
 - Use rshell to copy



Security



More Micro Python – Managing Time

```
import time
```

```
time.sleep(1)           # sleep for 1 second
```

```
time.sleep_ms(500)     # sleep for 500 milliseconds
```

```
time.sleep_us(10)      # sleep for 10 microseconds
```

```
start = time.ticks_ms() # get millisecond counter
```

```
delta = time.ticks_diff(time.ticks_ms(), start)
```

```
# compute time difference
```

More Micro Python – Managing Time

```
from machine import Timer

tim = Timer(-1)
tim.init(period=5000,
         mode=Timer.ONE_SHOT,
         callback=lambda t:print(1))
tim.init(period=2000,
         mode=Timer.PERIODIC,
         callback=lambda t:print(2))
```

More Micro Python - Debugging

```
3898
3898
3898
Traceback (most recent call last):
  File "main.py", line 62, in <module>
  File "scheduler.py", line 61, in Run
  File "tasks.py", line 176, in PressureSample
OSError: 5
Micro Python v1.5.8 on 2014-12-29; PYBv1.0 with STM32F405RG
Type "help()" for more information.
>>> █
```

Sensor Failed!

Execution halted

tasks.py

```
def PressureSample():
```

```
    # Read the previous conversion
```

```
    RxData = bytearray(4)
```

try:

```
    # Read the last conversion from the sensor
```

```
    RxData = I2C2.mem_read(2, int(BMP180_Address[0]), 0xF6)
```

```
    # Start next conversion
```

```
    I2C2.mem_write(0xE0, int(BMP180_Address[0]),0xF4)
```

```
except OSError as er:
```

```
    print("Received Exception OSError: " + str(er))
```

Presented by:

Customize the Kernel

```
beningo@ubuntu:~/MicroPython/micropython$ ls
ACKNOWLEDGEMENTS  docs      lib       pic16bit  teensy
bare-arm          drivers  LICENSE  py        tests
cc3200            esp8266  logo     qemu-arm  tools
CODECONVENTIONS.md examples minimal  README.md unix
CONTRIBUTING.md  extmod   mpy-cross stmhal    windows
```

Folder	Purpose
Bare-arm	Minimal version for ARM MCU's
Teensy	uP version for Teensy 3.1
Pic16bit	uP for 16 bit Microchip parts
Cc3200	uP for CC3200 from TI
Esp8266	uP for esp8266 wifi module
Py	Core Python implementation, compiler, runtime, etc
Stmhal	uP for STM32F405RG using St's HAL

Frozen Modules

- Frozen modules are pre-compiled Python libraries that can be built into the kernel

The image shows a file explorer view of a directory. The 'modules' folder is circled in red. A red arrow points from this folder to a terminal window. The terminal window shows the following commands and output:

```
CC ../py/./lib/embed/abort_.c
CC ../py/./lib/utils/printf.c
MPY modules/lcd160cr_test.py
MPY modules/lcd160cr.py
Creating build-PYBV4/frozen_mpy.c
CC build-PYBV4/frozen_mpy.c
```


Frozen Modules

```
>>> import lcd160cr  
>>> lcd160cr.myfunction()
```

- Any extra functionality can be built into the kernel using Frozen modules

Conclusions

- ESP8266 is an interesting option for creating an IoT device
- The module with Micro Python is marginally stable and operates inconsistently
- Firmware update mechanism has many potential security vulnerabilities
- Great for rapid prototyping and proof of concept

Additional Resources

- Download Course Material for
 - Python Doxygen Templates
 - Example source code
 - Blog
 - YouTube Videos
- Embedded Bytes Newsletter
 - <http://bit.ly/1BAHYXm>



From www.beningo.com under

- Blog > CEC – Designing IoT Sensor Nodes using the ESP8266

The Lecturer – Jacob Beningo



Jacob Beningo

Principal Consultant



Social Media / Contact

E : jacob@beningo.com

T : 810-844-1522

Twitter : Jacob_Beningo

f : Beningo Engineering

in : JacobBeningo

EDN : Embedded Basics

ARM Connected Community

Consulting

- Advising
- Coaching
- Content
- Consulting
- Training

www.beningo.com

Presented by: