

# GR-CITRUS 搭載 Ruby フォーム v2.46～

## クラスメソッド説明

Wakayama.rb  
山本三七男(たろサ)

## Rubyファーム仕様(基本的にバインドしているmrbgem)

```
mruby-math      #->Math.sin  
mruby-numeric-ext #->10.chr 0.zero?
```

GitHubのReleasesに、mrbgemのバインド違いのファームをリリースします。

バインドして欲しいmrbgemがありましたら、Issuesにあげてください。

## カーネルクラス

```
pinMode(pin, mode)
digitalWrite(pin, value)
digitalRead(pin)
analogRead(number)
pwm(pin, value)
analogReference(mode)
initDac()
analogDac(value)
delay(value)
delayMicroseconds(us)
millis()
micros()
led([sw])
tone(pin, freq[, duration])
noTone(pin)
randomSeed(value)
random([min,] max)
puts([value])
pulseIn(pin, val[, timeout])
shiftOut(dpin, clkpin, border, value)
shiftIn(dpin, clkpin, border)

MemFile.open(number, filename[, mode])
MemFile.close(number)
MemFile.read(number)
MemFile.write(number, buf, len)
MemFile.seek(number, byte)
MemFile.cp(src, dst[, mode])
MemFile.rm(filename)
```

## シリアルクラス

```
Serial.new(number[, bps])
  bps(bps)
  print([str])
  println([str])
  available()
  read()
  write(buf, len)
  flash()
```

## I2Cクラス

```
I2c.new(num)
  write(deviceID, address, data)
  read(deviceID, addL[, addH])
  begin(deviceID)
  lwrite(data)
  end()
  request(address, count)
  lread()
  available()
```

## サーボクラス

```
Servo.attach(ch, pin[, min, max])
Servo.write(ch, angle)
Servo.us(ch, us)
Servo.read(ch)
Servo.attached(ch)
Servo.attached?(ch)
Servo.detach(ch)
```

## システムクラス

```
System.exit()  
System.setrun(filename)  
System.version([r])  
System.push(address, buf, length)  
System.pop(address, length)  
System.fileload()  
System.reset()  
System.useSD()  
System.useWiFi()  
System.useMP3(pausePin, stopPin)  
System.use(className[, options])  
System.use?(className[, options])  
System.getMrbPath()
```

## グローバル変数

```
ON      = 1  
OFF     = 0  
HIGH   = 1  
LOW    = 0  
OUTPUT = 1  
INPUT  = 0  
INPUT_PULLUP = 2  
OUTPUT_OPENDRAIN = 3  
MSBFIRST = 1  
LSBFIRST = 0
```

## リアルタイムクロッククラス

```
Rtc.getTime()  
Rtc.setTime(array)  
Rtc.deinit()  
Rtc.init([adj])
```

# WA-MIKANと使えるクラス (V2ライブラリ)

## SDカードクラス

- SD.exists(filename)
- SD.mkdir(dirname)
- SD.remove(filename)
- SD.copy(srcfilename, distfilename)
- SD.rmdir(dirname)
- SD.open(number, filename[, mode])
- SD.close(number)
- SD.read(number)
- SD.seek(number, byte)
- SD.write(number, buf, len)
- SD.flush(number)
- SD.size(number)
- SD.position(number)
- SD.cpmem(sdfilename, memfilename[, mode])

## MP3クラス

- MP3.play(filename)
- MP3.led(sw)

## WiFiクラス

```
WiFi.at(command[, mode])  
WiFi.bypass()  
WiFi.cClose(number)  
WiFi.connect(SSID, Passwd)  
WiFi.connectedIP()  
WiFi.dhcp(mode, bool)  
WiFi.disconnect()  
WiFi.httpGet(URL[, Headers])  
WiFi.httpGetSD(Filename, URL[, Headers])  
WiFi.httpPost(URL, Headers, Body)  
WiFi.httpPostSD(URL, Headers, Filename)  
WiFi.httpServer([Port])  
WiFi.httpServerSD([Port])  
WiFi.ipconfig()  
WiFi.multiConnect(mode)  
WiFi.recv(number)  
WiFi.send(number, Data[, length])  
WiFi.serialOut(mode[, serialNumber])  
WiFi.setMode(mode)  
WiFi.softAP(SSID, Passwd, Channel, Encrypt)  
WiFi.udpOpen(number, IP_Address, SendPort, ReceivePort)  
WiFi.version()  
WiFi.base64(sFile, dFile[, decode])  
WiFi.ntp(IP[, diff])
```

## メソッドの説明 (V2ライブラリ)

### グローバル定数

グローバル定数として以下の変数を使用可能です。

ON 1

OFF 0

HIGH 1

LOW 0

INPUT 0

OUTPUT 1

INPUT\_PULLUP 2

OUTPUT\_OPEN\_DRAIN 3

MSBFIRST 1

LSBFIRST 0

# メソッドの説明 (V2ライブラリ)

## カーネルクラス

### PINのモード設定 `pinMode(pin, mode)`

ピンのデジタル入力と出力を設定します。

pin: ピンの番号

mode: 0: INPUTモード

1: OUTPUTモード

デフォルトは入力 (INPUT) モードです。

### デジタルライト `digitalWrite(pin, value)`

ピンのデジタル出力のHIGH/LOWを設定します。

pin: ピンの番号

value: 0: LOW

1: HIGH

### デジタルリード `digitalRead(pin)`

ピンのデジタル入力値を取得します。

pin: ピンの番号

戻り値

0: LOW

1: HIGH



# メソッドの説明 (V2ライブラリ)

## カーネルクラス

アナログリード `analogRead(pin)`

ピンのアナログ入力値を取得します。  
pin: アナログピンの番号 (14, 15, 16, 17)

戻り値  
10ビットの値 (0~1023)

アナログDACピン初期化 `initDac()`

アナログ出力ピンを初期化します。  
初期化しないとアナログ出力しません。

アナログDAC出力 `analogDac(value)`

ピンからアナログ電圧を出力します。  
value: 10bit精度 (0~4095) で0~3.3V

LEDオンオフ `led([sw])`

基板のLEDを点灯します。

sw: 0: 消灯  
1: 点灯

swを省略した場合は、消灯している場合は点灯し、点灯している場合は消灯します。

# メソッドの説明 (V2ライブラリ)

## カーネルクラス

PWM出力 `pwm(pin, value)`

ピンのPWM出力値をセットします。

pin: ピンの番号

value: 出力PWM比率 (0~255)

PWM設定後に、他のピンのpinMode設定をしてください。一度PWMに設定したピンは、リセットするまで変更できません。ショートしているPIOはINPUTに設定しておいてください。

アナログリファレンス `analogReference(mode)`

アナログ入力で使われる基準電圧を設定します。

mode: 0:DEFAULT : 5.0V Arduino互換, 1:INTERNAL : 1.1V 内蔵電圧, 2:EXTERNAL : AVREFピン供給電圧,  
3:RAW12BIT : 3.3V

ディレイ `delay(value)`

指定の時間 (ms) 動作を止めます。

value: 時間 (msec)

※delay中に強制的にGCを行っています。

ミリ秒を取得します `millis()`

システムが稼動してから経過した時間を取得します。

戻り値

起動からのミリ秒数

# メソッドの説明 (V2ライブラリ)

## カーネルクラス

マイクロ秒を取得します `micros()`

システムが稼動してから経過した時間を取得します。

戻り値

起動してからのマイクロ秒数

トーンを出力 `tone(pin, frequency[, duration])`

トーンを出力します。

pin: ピン番号

frequency: 周波数 Hz

duration: 出力を維持する時間[ms]。省略時、0指定時は出力し続ける。

トーンを停止 `noTone(pin)`

トーンを出力を停止します。

pin: ピン番号

# メソッドの説明 (V2ライブラリ)

## カーネルクラス

乱数の設定    `randomSeed (value)`

乱数を得るための種を設定します。  
value: 種となる値

乱数の取得    `random([min, ] max)`

乱数を取得します。  
min: 乱数の取りうる最小値。省略可  
max: 乱数の取りうる(最大値 + 1)  
maxは乱数の取りうる最大値に+1したものです。

USBシリアルに出力    `puts([value])`

USBシリアルポートに変数や文字列、配列の内容を出力します。  
value: 変数やアレイ  
省略時には改行のみ出力されます。

## メソッドの説明 (V2ライブラリ)

### カーネルクラス

パルスの長さの測定 `pulseIn(pin, val[, timeout])`

パルスの長さを測ります。

pin: ピン番号

val: 測定するパルスの種類。HIGHまたはLOW

timeout: タイムアウトまでの時間(単位 マイクロ秒)。省略時は1sec

戻り値

パルスの長さ[us]。パルスがスタートする前にタイムアウトとなった場合は 0。

指定した時間の一時停止 `delayMicroseconds(us)`

指定した時間、処理を一時停止します。

us: 一時停止する時間(単位 マイクロ秒)。

## メソッドの説明 (V2ライブラリ)

### カーネルクラス

1バイト分データを1ビットずつ出力 `shiftOut(dataPin, clockPin, bitOrder, value)`

1バイト分のデータを1ビットずつ出力します。

dataPin: データ出力ピン

clockPin: クロック出力ピン

bitOrder: MSBFIRST(最上位ビットから出力) または LSBFIRST(最下位ビットから出力)

value: 送信したいデータ (8bitデータ)

1バイト分データを1ビットずつ取り込む `shiftIn(dataPin, clockPin, bitOrder)`

1バイト分のデータを1ビットずつ取り込みます。

dataPin: データ入力ピン

clockPin: クロック出力ピン

bitOrder: MSBFIRST(最上位ビットから取り込み) または LSBFIRST(最下位ビットから取り込み)

# メソッドの説明 (V2ライブラリ)

## カーネルクラス

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/Kernel\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/Kernel_Samples)

### 使用例

```
pinMode(4, INPUT)
pinMode(5, OUTPUT)
```

```
x = digitalRead(4)
digitalWrite(5, 0)
```

```
10.times do
  led
  delay(1000)
end
```

```
#!/mruby
puts 10      #=> 10
puts 1.234   #=> 1.234
puts 'ABCD'  #=> ABCD
puts true    #=> 1
puts false   #=> 0
puts nil     #=> nil
puts [1, "A", ['array', 'OK', 'です', 14], 3, 4, 5, 6, 7]
```

# メソッドの説明 (V2ライブラリ)

## システムクラス

システムのバージョン取得 `System.version([R])`

システムのバージョンを取得します。  
R: 引数があればmrubyのバージョンを返します。

プログラムの終了 `System.exit()`

プログラムを終了させます。  
`System.setRun`により次に実行するプログラムがセットされていれば、そのプログラムが実行されます。

実行するプログラムの設定 `System.setrun(filename)`

次に実行するプログラムを設定します。  
filename: mrbファイル名

コマンドモードの呼び出し `System.fileload()`

コマンドモードを呼び出します。



# メソッドの説明 (V2ライブラリ)

## システムクラス

フラッシュメモリに書き込み `System.push(address, buf, length)`

フラッシュメモリに値を書き込みます。  
address: 書き込み開始アドレス (0x0000~0x00ff)  
buf: 書き込むデータ  
length: 書き込むサイズ (MAX 32バイト)

戻り値  
1: 成功  
0: 失敗

※ここに書き込んだ値は、電源を切っても消えません。

フラッシュメモリから読み出し `System.pop(address, length)`

フラッシュメモリから値を読み出します。  
address: 読み込みアドレス (0x0000~0x00ff)  
length: 読み込みサイズ (MAX 32バイト)

戻り値  
読み込んだデータ分

システムのリセット `System.reset()`

システムをリセットします。電源ONスタート状態となります。

## メソッドの説明 (V2ライブラリ)

### システムクラス

SDカードを使えるようにします `System.useSD()`

SDカードを使えるように設定します。

戻り値

0: 使用不可, 1: 使用可能

WA-MIKANボード (WiFi) を使えるようにします `System.useWiFi()`

WA-MIKANボード (WiFi) を使えるように設定します。

戻り値

0: 使用不可, 1: 使用可能

MP3再生を使えるようにします: `System.useMP3(pausePin, stopPin)`

MP3再生を行えるように設定します。

0番ピンとGNDの間にスピーカーを接続してください。

pausePin: 再生中の一時停止に使用するピン番号です。LOWになると一時停止/再開を繰り返します。

stopPin: 再生を止めるときに使用するピン番号です。LOWになると停止します。

戻り値

0: 使用不可, 1: 使用可能

設定できるピンは、1, 3, 4, 6, 9, 10, 14, 15, 16, 17, 18番ピンの11個です。

# メソッドの説明 (V2ライブラリ)

## システムクラス

実行しているmrbファイルパスを取得します: `System.getMrbPath()`

実行しているmrbファイルパスを取得します。

戻り値

実行しているmrbファイルパス(ファイル名です)。

追加クラスを使用できるようにします: `System.use(className[, options])`

追加クラスを使用できるようにする。

className: クラス名です。'SD'、'WiFi'、'MP3' のいずれかです。

Options: オプションの配列です。

SDはオプション無し。

WiFiはオプション無し。

MP3は再生中の一時停止に使用するピン番号と、再生を止めるときに使用するピン番号の配列を指定します。例) [3, 4]

戻り値

0: 使用不可

1: 使用可能

# メソッドの説明 (V2ライブラリ)

## システムクラス

追加クラスを使用できるようにします: `System.use?(ClassName[, Options])`

追加クラスを使用できるようにする。

ClassName: クラス名です。'SD'、'WiFi'、'MP3' のいずれかです。

Options: オプションの配列です。

SDはオプション無し。

WiFiはオプション無し。

MP3は再生中の一時停止に使用するピン番号と、再生を止めるときに使用するピン番号の配列を指定します。例) [3, 4]

戻り値

true: 使用可能

false: 使用不可

# メソッドの説明 (V2ライブラリ)

## システムクラス

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/System\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/System_Samples)

### 使用例

```
#アドレス0x0000から0x0005に{0x3a, 0x39, 0x38, 0x00, 0x36}の5バイトのデータを書き込みます
buf = 0x3a.chr+0x39.chr+0x38.chr+0x0.chr+0x36.chr
```

```
System.push( 0x0000, buf, 5 )
```

```
#アドレス0x0000から5バイトのデータを読み込みます
ans = System.pop(0x0000, 5)
```

```
System.setrun('sample.mrb') #次に実行するプログラム名をセットします
```

```
System.exit() #このプログラムを終了します。
```

```
Usb = Serial.new(0,115200)
```

```
if(System.use?("WiFi") == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end
Usb.println "WiFi Ready"
```

```
if(System.use?("MP3", [3,4]) == false)then
  Usb.println "MP3 can't use."
  System.exit()
end
Usb.println "MP3 Ready"
```

# メソッドの説明 (V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

### シリアル通信の初期化 `Serial.new(num, bps)`

シリアル通信を初期化します。シリアル通信を使用する場合は、初めに初期化を行ってください。

num: 初期化する通信番号  
0: USB  
1: 0ピン送信/1ピン受信  
2: 5ピン送信/6ピン受信  
3: 7ピン送信/8ピン受信  
4: 12ピン送信/11ピン受信

bps: ボーレート (bps) 基本的に任意の値が設定できます。

戻り値  
シリアルのインスタンス

### ボーレートの設定 `bps (baudrate)`

シリアル通信のボーレートを設定します。

baudrate: ボーレート

### シリアルポートへの出力 `print([str])`

シリアルポートに出力します。

str: 文字列。省略時は何も出力しません設定できます。

# メソッドの説明 (V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

シリアルポートへの出力 (¥r¥n付き) `println([str])`

シリアルポートに¥r¥n付きで出力します。  
str: 文字列。省略時は改行のみ

シリアル受信チェック `available()`

シリアルポートに受信データがあるかどうか調べます。

戻り値

シリアルバッファにあるデータのバイト数。0の場合はデータなし。

シリアルポートからデータ取得 `read()`

シリアルポートの受信データを取得します。

戻り値

読み込んだデータ配列

データは0x00~0xFFの値

# メソッドの説明 (V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

### シリアルポートヘデータ出力 `write(buf, len)`

シリアルポートにデータを出力します。

buf: 出力データ

len: 出力データサイズ

戻り値

出力したバイト数

### シリアルデータをフラッシュします `flash()`

シリアルデータをフラッシュします。



# メソッドの説明 (V2ライブラリ)

## シリアルクラス

このクラスはポート毎にインスタンスを生成して使います。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/Serial\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/Serial_Samples)

### 使用例

```
USB_Out = Serial.new(0, 115200)
sw = 0

while(USB_Out.available() > 0) do    #何か受信があった
  USB_Out.read()
end

50.times do
  while(USB_Out.available() > 0) do #何か受信があった
    c = USB_Out.read()              #文字取得
    USB_Out.print c                  #読み込んだ文字をprintします
  end

  #LEDを点滅させます
  led sw
  sw = 1 - sw

  delay 500
end
```

```
USB_Out = Serial.new(0, 115200)
data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr + 0x0d.chr + 0x0a.chr
USB_Out.write(data, 7)

System.exit()
```

## メソッドの説明 (V2ライブラリ)

### MemFileクラス (Flashメモリをメディアのように扱うクラス)

ファイルのオープン `MemFile.open(number, filename[, mode])`

ファイルをオープンします。  
number: ファイル番号 0 または 1  
filename: ファイル名 (8.3形式)  
mode: 0:Read, 1:Append, 2:New Create

戻り値  
成功: 番号, 失敗: -1

※同時に開けるファイルは2つまでに限定しています。

ファイルのクローズ `MemFile.close(number)`

ファイルをクローズします。  
number: クローズするファイル番号 0 または 1

ファイルの読み出し位置に移動 `MemFile.seek(number, byte)`

Openしたファイルの読み出し位置に移動します。  
number: ファイル番号 0 または 1  
byte: seekするバイト数 (-1) でファイルの最後に移動する

戻り値  
成功: 1, 失敗: 0

## メソッドの説明 (V2ライブラリ)

### MemFileクラス (Flashメモリをメディアのように扱うクラス)

Openしたファイルからの読み込み `MemFile.read(number)`

Openしたファイルから1バイト読み込みます。

number: ファイル番号 0 または 1

戻り値

0x00~0xFFが返る。ファイルの最後だったら-1が返る。

Openしたファイルにバイナリデータを書き込む `MemFile.write(number, buf, len)`

Openしたファイルにバイナリデータを書き込みます。

number: ファイル番号 0 または 1

buf: 書き込むデータ

len: 書き込むデータサイズ

戻り値

実際に書いたバイト数

ファイルをコピーします `MemFile.cp(srcFilename, dstFilename[, mode])`

ファイルをコピーします。

srcFilename: コピー元ファイル名

dstFilename: コピー先ファイル名

mode: 0:上書きしない, 1:上書きする 省略時は上書きしない。

戻り値

成功: 1, 失敗: 0

## メソッドの説明 (V2ライブラリ)

### MemFileクラス (Flashメモリをメディアのように扱うクラス)

ファイルを削除します MemFile.rm( Filename )

ファイルを削除します。

Filename: 削除するファイル名

戻り値

成功: 1, 失敗: 000~0xFFが返る。ファイルの最後だったら-1が返る。

# メソッドの説明 (V2ライブラリ)

## MemFileクラス

### 使用例

```
MemFile.open(0, 'sample.txt', 2)
  MemFile.write(0, 'Hello mruby World', 17)
  data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr
  MemFile.write(0, data, 5)
MemFile.close(0)

MemFile.cp('sample.txt', 'memfile.txt', 1)

USB = Serial.new(0, 115200)      #USBシリアル通信の初期化

MemFile.open(0, 'memfile.txt', 0)
while(true)do
  c = MemFile.read(0)
  if(c < 0)then
    break
  end
  USB.write(c.chr, 1)
end
MemFile.close(0)
System.exit()
```

# メソッドの説明 (V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### I2C通信を行うピンの初期化 I2c.new(number)

I2C通信を行うピンの初期化を行います。

num: 通信番号

1: SDA-0ピン, SCL-1ピン

2: SDA-5ピン, SCL-6ピン

3: SDA-7ピン, SCL-8ピン

4: SDA-12ピン, SCL-11ピン

戻り値

I2cのインスタンス

### アドレスからデータを読み込み: read(deviceID, addressL[, addressH])

アドレスからデータを読み込みます。

deviceID: デバイスID

addressL: 読み込み下位アドレス

addressH: 読み込み上位アドレス

戻り値

読み込んだ値

### アドレスにデータを書き込みます write(deviceID, address, data)

アドレスにデータを書き込みます。

deviceID: デバイスID

address: 書き込みアドレス

data: データ

戻り値

常に 0

# メソッドの説明 (V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

I2Cデバイスに対して送信を開始するための準備をする: `begin(deviceID)`

I2Cデバイスに対して送信を開始するための準備をします。この関数は送信バッファを初期化するだけで、実際の動作は行わない。繰り返し呼ぶと、送信バッファが先頭に戻る。

deviceID: デバイスID 0~0x7Fまでの純粋なアドレス

デバイスに対してI2Cの送信シーケンスの発行 `end()`

デバイスに対してI2Cの送信シーケンスを発行します。I2Cの送信はこの関数を実行して初めて実際に行われる。

戻り値  
常に 0

デバイスに受信シーケンスを発行しデータを読み出す `request(address, count)`

デバイスに対して受信シーケンスを発行しデータを読み出します。

address: 読み込み開始アドレス

count: 読み出す数

戻り値  
実際に受信したバイト数

# メソッドの説明 (V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

### 送信バッファの末尾に数値を追加する `lwrite(data)`

送信バッファの末尾に数値を追加します。

data: セットする値

戻り値

送信したバイト数(バッファに溜めたバイト数)を返す。

送信バッファ(260バイト)に空き容量が無ければ失敗して0を返す。

### デバイスに受信シーケンスを発行しデータを読み出す `lread()`

デバイスに対して受信シーケンスを発行しデータを読み出します。

戻り値

読み込んだ値

### 受信バッファ内にあるデータ数を調べる `available()`

デバイスに対して受信バッファ内にあるデータ数を調べます。

戻り値

データ数



# メソッドの説明 (V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/I2C\\_Samples](https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/I2C_Samples)

### 使用例

```
@APTemp = 0x5D                                # 0b01011101 圧力・温度センサのアドレス
USB = Serial.new(0, 115200)                    # USBシリアル通信の初期化

# センサ接続ピンの初期化 (12番SDA, 11番SCL)
sensor = I2c.new(3)
delay(300)

# 気圧と温度センサの初期化
@APTemp = 0x5D                                # 0b01011101
APTemp_CTRL_REG1 = 0x20 # Control register
APTemp_SAMPLING = 0xA0 # A0:7Hz, 90:1Hz
# 7Hz
sensor.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)
delay(100)

# 気圧を取得します -----
# Address 0x28, 0x29, 0x2A, 0x2B, 0x2C
v0 = sensor.read( @APTemp, 0x28, 0x29)
v1 = sensor.read( @APTemp, 0x2A)
a = v0 + v1 * 65536
a = a / 4096.0 # hPa単位に直す

# 温度を取得します -----
v2 = sensor.read( @APTemp, 0x2B, 0x2C)
if v2 > 32767
  v2 = v2 - 65536
end
t = v2 / 480.0 + 42.5
USB.println(a.to_s + ", " + t.to_s)
```

# メソッドの説明 (V2ライブラリ)

## I2cクラス

このクラスはポート毎にインスタンスを生成して使います。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/I2C\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/I2C_Samples)

### 使用例

```
USB = Serial.new(0, 115200)      #USBシリアル通信の初期化
#センサ接続ピンの初期化(12番SDA, 11番SCL)
sensor = I2c.new(3)
delay(300)
#気圧と温度センサの初期化
@APTemp = 0x5D                  # 0b01011101
APTemp_CTRL_REG1 = 0x20        # Control register
APTemp_SAMPLING = 0xA0          # A0:7Hz, 90:1Hz
sensor.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)    # 7Hz
delay(100)

#Address 0x2B, 0x2C
sensor.begin(@APTemp)
sensor.lwrite(0x2B)
sensor.end()
sensor.request(@APTemp, 1)
datL = sensor.lread()

sensor.begin(@APTemp)
sensor.lwrite(0x2C)
sensor.end()
sensor.request(@APTemp, 1)
datH = sensor.read()
v = datL + datH * 256
if v > 32767
  v = v - 65536
end
t = v / 480.0 + 42.5
USB.println(t.to_s)
```

## メソッドの説明 (V2ライブラリ)

### サーボクラス

サーボ出力を任意のピンに割り当てます `Servo.attach(ch, pin[, min, max])`

ch: サーボのチャンネル 0~9まで指定できます  
pin: 割り当てるピン番号  
min: サーボの角度が0度のときのパルス幅(マイクロ秒)。デフォルトは544  
max: サーボの角度が180度のときのパルス幅(マイクロ秒)。デフォルトは2400

サーボの角度をセットします: `Servo.write(ch, angle)`

ch: サーボのチャンネル 0~9まで指定できます  
angle: 角度 0~180バイスに対して受信シーケンスを発行しデータを読み出します。

サーボモータにus単位で角度を指定します: `Servo.us(ch, us)`

ch: サーボのチャンネル 0~9まで指定できます  
us: 出力したいパルスの幅 1~19999, 0で出力 OFF  
サーボモータに与えられるパルスは20ms周期で、1周期中のHighの時間を直接指定する。  
実質的にPWM出力。連続回転タイプのサーボでは、回転のスピードが設定することができる。

最後に設定された角度を読み出します: `Servo.read(ch)`

ch: サーボのチャンネル 0~9まで指定できます

戻り値  
マイクロ秒単位。ただし `us(ch)` で与えた値は読みとれません。

## メソッドの説明 (V2ライブラリ)

### サーボクラス

ピンにサーボが割り当てられているかを確認します: `Servo.attached(ch)`

ch: サーボのチャンネル 0~9まで指定できます

戻り値

1: 割り当てられている  
0: 割り当てはない

ピンにサーボが割り当てられているかを確認します: `Servo.attached?(ch)`

ch: サーボのチャンネル 0~9まで指定できます

戻り値

true: 割り当てられている  
false: 割り当てはない

サーボの動作を止め、割り込みを禁止します: `Servo.detach(ch)`

ch: サーボのチャンネル 0~9まで指定できます

# メソッドの説明 (V2ライブラリ)

## サーボクラス

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCode-Rubic/Servo\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCode-Rubic/Servo_Samples)

### 使用例

```
g_pos = 0
g_inc = 10

USB = Serial.new(0, 115200)      #USBシリアル通信の初期化

#8番ピンをサーボ用ピンに割り当てる。
Servo.attach(0, 8)
Servo.write(0, g_pos) #サーボの角度設定

#サーボを10度ずつ50回動かす
50.times do
  delay(100)
  g_pos = g_pos + g_inc
  Servo.write(0, g_pos)
  if(g_pos >= 180 || g_pos <= 0) then
    g_inc = -g_inc
  end
end

Servo.detach(0)
```

# メソッドの説明 (V2ライブラリ)

## リアルタイムクロッククラス

RTCを起動します: `Rtc.init([adj])`

RTCを起動します。

`adj`: 10秒毎の1/32768単位の補正カウント値。最大6bit(±63)、0の場合は補正を停止。

戻り値

0: 起動失敗

1: 起動成功

2: RTCは既に起動していた(成功)

`init()` を実行すると日時がリセットされます。

RTCを停止します: `Rtc.deinit()`

RTCを停止します。

戻り値

0: 失敗

1: 成功

# メソッドの説明 (V2ライブラリ)

## リアルタイムクロッククラス

RTCの日時をセットします: `Rtc.setTime(array)`

RTCの日時をセットします。

array: 年 (0000-9999), 月 (1-12), 日 (1-31), 時 (0-23), 分 (0-59), 秒 (0-59) の配列

戻り値

0: 失敗

1: 成功

RTCの日時を取得します: `Rtc.getTime()`

RTCの日時を取得します。

戻り値は以下の値が配列で返ります

year: 年 (2000-2099)

month: 月 (1-12)

day: 日 (1-31)

hour: 時 (0-23)

minute: 分 (0-59)

second: 秒 (0-59)

weekday: 曜日 (0-6) 0: 日, 1: 月, 2: 火, 3: 水, 4: 木, 5: 金, 6: 土

# メソッドの説明 (V2ライブラリ)

## リアルタイムクロッククラス

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/RTC\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/RTC_Samples)

### 使用例

```
tm = Rtc.getTime
if(tm[0] < 2010) then
  puts 'RTC Initialized'
  Rtc.init
end

Rtc.setTime([2016, 4, 16, 17, 0, 0])

15.times do|i|
  led(i % 2)
  year, mon, da, ho, min, sec = Rtc.getTime()
  puts year.to_s + "/" + mon.to_s + "/" + da.to_s + " " + ho.to_s + ":" + min.to_s + ":" +
sec.to_s
  delay(500)
end
```



# メソッドの説明 (V2ライブラリ)

## SDカードクラス

System.useSD() を呼んでおく必要があります。

ファイルのオープン SD.open(number, filename[, mode])

ファイルをオープンします。

number: ファイル番号 0 または 1

filename: ファイル名 (8.3形式)

mode: 0:Read, 1:Append, 2:New Create

戻り値

成功: 番号, 失敗: -1

※同時に開けるファイルは2つまでに限定しています。

ファイルのクローズ SD.close(number)

ファイルをクローズします。

number: クローズするファイル番号 0 または 1

ファイルの読み出し位置に移動 SD.seek(number, byte)

Openしたファイルの読み出し位置に移動します。

number: ファイル番号 0 または 1

byte: seekするバイト数 (-1) でファイルの最後に移動する

戻り値

成功: 1, 失敗: 0

# メソッドの説明 (V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または `System.use('SD')` しておく必要があります。

Openしたファイルからの読み込み `SD.read(number)`

Openしたファイルから1バイト読み込みます。

number: ファイル番号 0 または 1

戻り値

0x00~0xFFが返る。ファイルの最後だったら-1が返る。

Openしたファイルにバイナリデータを書き込む `SD.write(number, buf, len)`

Openしたファイルにバイナリデータを書き込みます。

number: ファイル番号 0 または 1

buf: 書き込むデータ

len: 書き込むデータサイズ

戻り値

実際に書いたバイト数

Openしたファイルの書き込みをフラッシュします: `SD.flush(number)`

Openしたファイルの書き込みをフラッシュします。

number: ファイル番号 0 または 1

## メソッドの説明 (V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または `System.use('SD')` しておく必要があります。

Openしたファイルのサイズを取得します: `SD.size(number)`

Openしたファイルのサイズを取得します。  
number: ファイル番号 0 または 1

戻り値  
ファイルサイズ

Openしたファイルのseek位置を取得します: `SD.position(number)`

Openしたファイルのseek位置を取得します。  
number: ファイル番号 0 または 1

戻り値  
シーク位置

# メソッドの説明 (V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または `System.use('SD')` しておく必要があります。

ディレクトリを作成する: `SD.mkdir(dirname)`

ディレクトリを作成する。

`dirname`: 作成するディレクトリ名

戻り値

成功: 1, 失敗: 0

ファイルを削除します: `SD.remove(filename)`

ファイルを削除します。

`filename`: 削除するファイル名

戻り値

成功: 1, 失敗: 0

ファイルをコピーする: `SD.copy(srcfilename, distfilename)`

ファイルをコピーする。

`srcfilename`: コピー元ファイル名

`distfilename`: コピー先ファイル名

戻り値

成功: 1, 失敗: 0

## メソッドの説明 (V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または `System.use('SD')` しておく必要があります。

ディレクトリを削除します: `SD.rmdir(dirname)`

ディレクトリを削除します。  
`dirname`: 削除するディレクトリ名

戻り値  
成功: 1, 失敗: 0

ファイルが存在するかどうか調べる: `SD.exists(filename)`

ファイルが存在するかどうか調べます。  
`filename`: 調べるファイル名

戻り値  
存在する: 1, 存在しない: 0

ファイルをフラッシュメモリにコピーします: `SD.cpmem(SDFile, MemFile[, mode])`

SDカードのファイルをフラッシュメモリにコピーします。  
`SDFile`: SDカードのファイル名  
`MemFile`: フラッシュメモリのコピー先ファイル名  
`mode`: 0:上書きしない, 1:上書きする

戻り値  
成功: 1, 失敗: 0

# メソッドの説明 (V2ライブラリ)

**SDカードクラス** `System.useSD()` を呼んでおく、または `System.use('SD')` しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/SD\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/SD_Samples)

## 使用例

```
if(System.use?('SD') == false) then
  System.exit
end

SD.open(0, 'sample.txt', 2)
SD.write(0, 'Hello mruby World', 17)
data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr
Serial.write(0, data, 5)
SD.close(0)

USB = Serial.new(0, 115200)      #USBシリアル通信の初期化

SD.open(0, 'sample.txt', 0)
while(true) do
  c = SD.read(0)
  if(c < 0) then
    break
  end
  USB.write(c.chr, 1)
end

SD.close(0)
System.exit()
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

ステーションモードを設定する: WiFi.setMode(mode)

ステーションモードを設定します。

mode: 1:Station, 2:SoftAP, 3:Station + SoftAP1

戻り値

ESP8266の戻り値

応答のシリアル出力設定: WiFi.serialOut(mode[, serialNumber])

ESP8266に送信したコマンドの応答をシリアル出力するときに設定します。

mode: 0:出力しない, 1:出力する

serialNumber: 出力先のシリアル番号

戻り値

無し

ATコマンドを送信する: WiFi.at(command[, mode])

ATコマンドを送信します。

commnad: ATコマンド文字列

mode: 0:'AT+' を自動追加する、1:'AT+' を自動追加しない

戻り値

ESP8266の戻り値

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

WiFi接続する: WiFi.connect(SSID, Passwd)

WiFiアクセスポイントの接続します。

SSID: WiFiのSSID

Passwd: パスワード

戻り値

ESP8266の戻り値

IPアドレスとMACアドレスの表示: WiFi.ipconfig()

IPアドレスとMACアドレスを表示します

戻り値

ESP8266の戻り値

USBポートとESP8266をシリアルで直結します: WiFi.bypass()

USBポートとESP8266をシリアルで直結します。

リセットするまで、処理は戻りません。



# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

ESP8266のソフトのバージョンを取得する: `WiFi.version()`

ESP8266のソフトのバージョンを取得します。

戻り値  
ESP8266の戻り値

WiFiを切断します: `WiFi.disconnect()`

WiFiを切断します。

戻り値  
ESP8266の戻り値

複数接続可能モードの設定: `WiFi.multiConnect(mode)`

複数接続可能モードの設定をします。  
mode: 0:1接続のみ, 1:4接続まで可能

戻り値  
ESP8266の戻り値

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

http GET結果をSDカードに保存する: WiFi.httpGetSD(Filename, URL[, Headers])

http GET結果をSDカードに保存します。

Filename: 保存するファイル名

URL: URL

Headers: ヘッダに追記する文字列の配列

戻り値

0: 失敗

1: 成功

2: SDカードが使えない

3: 送信データファイルをオープンできなかった

4: 送信データサイズを読み込めなかった

5: 送信データファイルを読み込めなかった

6: 受信用ファイルをオープンできなかった

7: 受信したファイルの生成に失敗した

http GETプロトコルを送信する: WiFi.httpGet(URL[, Headers])

http GETプロトコルを送信します。送信のみで、結果の受信しません。

URL: URL

Headers: ヘッダに追記する文字列の配列

戻り値

0: 失敗

1: 成功

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

### TCP/UDPの接続を閉じる: WiFi.cClose(number)

TCP/UDPの接続を閉じます。  
number: 接続番号 (0~3)

戻り値  
ESP8266の戻り値

### UDP接続を開始する: WiFi.udpOpen(number, IP\_Address, SendPort, ReceivePort)

UDP接続を開始します。  
number: 接続番号 (0~3)  
IP\_Address: 通信相手アドレス  
SendPort: 送信ポート番号  
ReceivePort: 受信ポート番号

戻り値  
ESP8266の戻り値

### 指定接続番号にデータを送信する: WiFi.send(number, Data[, length])

指定接続番号にデータを送信します。  
number: 接続番号 (0~3)  
Data: 送信するデータ  
length: 送信データサイズ

戻り値  
送信データサイズ

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

指定接続番号からデータを受信する: `WiFi.recv(number)`

指定接続番号からデータを受信します。

number: 接続番号 (0~3)

戻り値

受信したデータの配列   ただし、256以下

SDカードのファイルをhttpPOSTする: `WiFi.httpPostSD(URL, Headers, Filename)`

SDカードのファイルをhttp POSTします。

URL: URL

Headers: ヘッダに追記する文字列の配列

Filename: POSTするファイル名

戻り値

0: 失敗

1: 成功

2: SDカードが使えない

2: SDカードが使えない

3: 送信データファイルサイズを読み込めなかった

4: 送信ヘッダファイルを書き込みオープンできなかった

5: 送信ヘッダファイルサイズを読み込めなかった

6: 送信ヘッダファイルを読み込みオープンできなかった

7: 送信データファイルを読み込みオープンできなかった

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

http POSTする: WiFi.httpPost(URL, Headers, data)

http POSTします。送信のみで結果は受信しません。

URL: URL

Headers: ヘッダに追記する文字列の配列

Data: POSTデータ

戻り値

0: 失敗

1: 成功

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

httpサーバを開始します: WiFi.httpServer([Port])

httpサーバを開始します。アクセスの有無で返り値が変わります。  
通信データの受信は200バイト程度です。ヘッダ情報が全て受信できることはありません。  
ポート番号を省略したときはアクセス確認します。

Port: 待ち受けポート番号  
-1: サーバ停止

```
revData, conNum = WiFi.httpServer()
```

戻り値  
revData:  
0: アクセスはありません

クライアントからアクセスがあるとき、通信内容と接続番号の2つが返ります。

GET: パスが返ります。

GET以外、受信したすべての通信内容が返ります。

conNum: 接続番号が返ります。

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

httpサーバを開始します: WiFi.httpServerSD([Port])

httpサーバを開始します。アクセスの有無で返り値が変わります。  
SDカードが必須となります。受信ヘッダ内容はSDカードのheader.txtファイルに格納されます。  
ポート番号を省略したときはアクセス確認します。

Port: 待ち受けポート番号  
-1: サーバ停止

revData, conNum = WiFi.httpServer()

戻り値

revData:

- 0: アクセスはありません
- 2: SDカードが使いません
- 3: ファイルのアクセスに失敗しました

クライアントからアクセスがあるとき、通信内容と接続番号の2つが返ります。

GET: パスが返ります。

GET以外、ヘッダの1行目が返ります。

conNum: 接続番号が返ります。

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

WiFiアクセスポイントになる: WiFi.softAP(SSID, Passwd, Channel, Encrypt)

WiFiアクセスポイントになります。

SSID: WiFiのSSID

Passwd: パスワード

Channel: チャンネル

Encrypt: 暗号タイプ 0:Open, 1:WEP, 2:WPA\_PSK, 3:WPA2\_PSK, 4:WPA\_WPA2\_PSK

戻り値

ESP8266の戻り値

アクセスポイントに接続されている端末情報の取得: WiFi.connectedIP()

アクセスポイントに接続されている端末のIPアドレスとMACアドレスを取得します。

戻り値

ESP8266の戻り値

DHCPサーバ機能の切り替え: WiFi.dhcp(mode, bool)

DHCPサーバ機能を有効にするか無効にするかを設定します。

mode: 0:SoftAP, 1:Station, 2:Both softAP + Station のどのモードで有効にするかを設定します。

bool: 0:disable, 1:enable

戻り値

ESP8266の戻り値



# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

BASE64変換を行う: WiFi.base64( SoFile, DeFile[, decode] )

指定ファイルのBASE64変換を行います。SDカードが必要です。

SoFile: 入力ファイル名

DeFile: 出力ファイル名

decode: 0:エンコード, 1:デコード  
省略時はエンコードします。

戻り値

0: 成功しました

1: SDカードがありません

2: ファイルのオープンに失敗しました。

ntpサーバからTransfer Timeの取得: WiFi.ntp( IP[, diff] )

ntpサーバからTransfer Timeを取得します。

IP: ntpサーバのIPアドレス、またはドメイン名

diff: 時差です。JST(日本)時間の場合は、9をいれます。

戻り値は以下の値が配列で返ります。

year: 年(1900-)

mon: 月(1-12)

day: 日(1-31)

hour: 時(0-23)

min: 分(0-59)

second: 秒(0-59)

weekday: 曜日(0-6) 0:日, 1:月, 2:火, 3:水, 4:木, 5:金, 6:土

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)  # LOW:Disable
delay 500
digitalWrite(5, 1)  # LOW:Disable

Usb = Serial.new(0, 115200)

if( System.use?('WiFi') == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.print WiFi.version
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
```

```
pinMode(5, 1)
```

```
digitalWrite(5, 0)    # LOW:Disable
```

```
delay 500
```

```
digitalWrite(5, 1)    # LOW:Disable
```

```
Usb = Serial.new(0, 115200)
```

```
if( System.use?('WiFi') == false)then
```

```
  Usb.println "WiFi Card can't use."
```

```
  System.exit()
```

```
end
```

```
Usb.println "GR-CITRUS[bypass]"
```

```
WiFi.bypass()
```

```
#GR-CITRUSはESP8266とUSBシリアル通信が接続したままとなります。
```

```
#ターミナルを使って、ESP8266との通信テストをすることができます。
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)  # LOW:Disable
delay 500
digitalWrite(5, 1)  # LOW:Disable

Usb = Serial.new(0, 115200)

if( System.use?('WiFi') == false)then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.println WiFi.disconnect
Usb.println WiFi.setMode 3  #Station-Mode & SoftAPI-Mode
Usb.println WiFi.connect("TAROSAY", "****")
Usb.println WiFi.ipconfig
Usb.println WiFi.multiConnect 1

for value in 1..10
  Usb.println WiFi.httpGet("192.168.1.58:3000/?value1=" + value.to_s + "&value2=" +
    (value*value).to_s).to_s
end

heds=["User-Agent: curl"]
Usb.println WiFi.httpGetSD("wether1.htm", "wttr.in/wakayama").to_s
Usb.println WiFi.httpGetSD("wether2.htm", "wttr.in/wakayama", heds).to_s
Usb.println WiFi.httpGetSD("yahoo.htm", "www.yahoo.co.jp").to_s
Usb.println WiFi.httpGetSD("google.htm", "www.google.co.jp").to_s
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#UDP通信, WA-MIKAN 受信:5555, 送信: 5556
Usb.println WiFi.udpOpen(4, "192.168.1.44", 5555, 5556)

Usb.println "UDP受信した分がarray配列で返ります"
1000.times do
  array = WiFi.recv 4  #受信データがない場合は array[0]に -1 が返ります
  if(array[0] >= 0) then
    for var in array do
      Usb.println var.to_s
    end
  end
  delay 10
end
```

```
#UDP通信, WA-MIKAN 受信:5555, 送信: 5556
Usb.println WiFi.udpOpen(4, "192.168.1.44", 5555, 5556)

100.times do
  WiFi.send 4, "hoho01111122222¥r¥n"
  delay 25
end
Usb.println WiFi.send(4, 0x02.chr + "bcdefghijklmn" + 0x03.chr + "ddd¥r¥n").to_s

Usb.println WiFi.cClose 4
Usb.println WiFi.disconnect
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#http POST
```

```
header=["User-Agent: gr-citrus", "Accept: application/json", "Content-type: application/json"]  
body = ' { "name" : "tarosay" } '
```

```
WiFi.httpPost("192.168.1.52:3000", header, body)
```

```
# body.json ファイルをPOSTします
```

```
WiFi.httpPostSD("192.168.1.52:3000", header, "body.json")
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
header0 = "HTTP/1.1 200 OK\r\nServer: GR-CITRUS\r\nContent-Type: text/html\r\n"
header0 += "Date: Sun, 13 Nov 2016 12:00:00 GMT\r\nConnection: close\r\n"
body0 = '<html><head><meta http-equiv="Content-Type" content="text/html; charset=utf-8">'
body0 += '<title>RAMUNE SHOOTER</title></head>'

Usb.println WiFi.httpServer(80).to_s #-> 80ポートでhttp受信します
while(true) do
  res,num = WiFi.httpServer #-> アクセス確認しています。
  Usb.println res.to_s #-> 0のときはアクセスなし、GETのときはパスをそれ以外はヘッダ先頭行が返る

  if(res == "/exit")
    body1 = '<body><h1 align="center">終了します。</h1></body>' + "\r\n\r\n"
    header1 = "Content-Length: " + (body0 + body1).length.to_s + "\r\n\r\n"
    WiFi.send(num, header0)
    WiFi.send(num, header1)
    WiFi.send(num, body0)
    WiFi.send(num, body1)
    break
  elsif(res != 0)
    Usb.println "Else:" + res.to_s
    body1 = '<body><h1 align="center">エラーです。</h1></body>' + "\r\n\r\n"
    header1 = "Content-Length: " + (body0 + body1).length.to_s + "\r\n\r\n"
    WiFi.send(num, header0)
    WiFi.send(num, header1)
    WiFi.send(num, body0)
    WiFi.send(num, body1)
  end
  delay 100
end
WiFi.httpServer(-1) #-> サーバを停止します
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi')しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, 1)
digitalWrite(5, 0)  # LOW:Disable
delay 500
digitalWrite(5, 1)  # LOW:Disable
delay 500

Usb = Serial.new(0, 115200)

if(!System.use?('WiFi'))then
  Usb.println "WiFi Card can't use."
  System.exit()
end

Usb.println WiFi.setMode 3  #Station-Mode & SoftAPI-Mode
Usb.println WiFi.softAP "GR-CITRUS", "37003700", 2, 3
Usb.println WiFi.dhcp 0, 1
Usb.println WiFi.multiConnect 1
30.times do
  Usb.println "Connected IP= " + WiFi.connectedIP
  delay 1000
end
```



# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbb-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, OUTPUT)
digitalWrite(5, LOW)    # LOW:Disable
delay 500
digitalWrite(5, HIGH)   # HIGH:Enable
delay 500

if(!System.use?('WiFi'))then
  puts "WiFi can't use."
  System.exit()
end

puts "BASE64 Encoding"

puts WiFi.base64('photo.jpg', 'photo.b64')
puts "Encode Finish"

puts "BASE64 Decoding"
puts WiFi.base64('photo.b64', 'photo1.jpg', 1)
puts "Decode Finish"
```

# メソッドの説明 (V2ライブラリ)

## WiFiクラス

System.WiFi() を呼んでおく、またはSystem.use('WiFi') しておく必要があります。

詳細なサンプルはここを参照

[https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi\\_Samples](https://github.com/wakayamarb/wrbv-v2lib-firm/tree/master/samples/VSCoDe-Rubic/WiFi_Samples)

### 使用例

```
#ESP8266を一度停止させる(リセットと同じ)
pinMode(5, OUTPUT)
digitalWrite(5, LOW)    # LOW:Disable
delay 500
digitalWrite(5, HIGH)   # HIGH:Enable
delay 500

if(!System.use?("WiFi"))
  puts "WiFi Card can't use."
  System.exit()
end

puts WiFi.connect("TAROSAY", "*****")

tm = WiFi.ntp("ntp.jst.mfeed.ad.jp", +9)

System.exit if(!tm.instance_of?(Array))

puts tm
```

# メソッドの説明 (V2ライブラリ)

## MP3クラス

System.useMP3(pausePin, stopPin) を呼んでおく、または  
System.use('MP3', [pausePin, stopPin]) しておく必要があります。

### MP3ファイルを再生する: MP3.play(filename)

MP3ファイルまたはwavファイルを再生します。  
0番ピンとGNDの間にスピーカーを接続してください。

filename: 再生するMP3ファイル名またはwavファイル名

戻り値

エラーが出たときは、その内容が返ります。エラーが無いときは何も返りません。

System.useMP3(pausePin, stopPin) で設定したピン番号の入力をLOWにすることによって、曲の一時停止や終了を行うことができます。

設定できるピンは、1, 3, 4, 6, 9, 10, 14, 15, 16, 17, 18番ピンの11個です。

### MP3再生中にLEDを点滅させる: MP3.led(sw)

MP3再生中にLEDを点滅させます。

sw: 0:何もしない、1:点滅させる

ポーズ中はLEDは点灯した状態となります。

# メソッドの説明 (V2ライブラリ)

## MP3クラス

System.useMP3(pausePin, stopPin) を呼んでおく、または  
System.use('MP3', [pausePin, stopPin]) しておく必要があります。

### 使用例

```
Usb = Serial.new(0, 115200)

#3番ピンを一時停止に、4番ピンを再生停止ボタンに設定します。

if(System.use?('MP3', [3, 4]) == false) then
  Usb.println "MP3 can't use."
  System.exit()
end
Usb.println "MP3 Ready"

MP3.led 1
Usb.println "/koidance.mp3"
Usb.print MP3.play "/koidance.mp3"

MP3.led 0
Usb.println "/decrain.mp3"
Usb.print MP3.play "/decrain.mp3"
```