

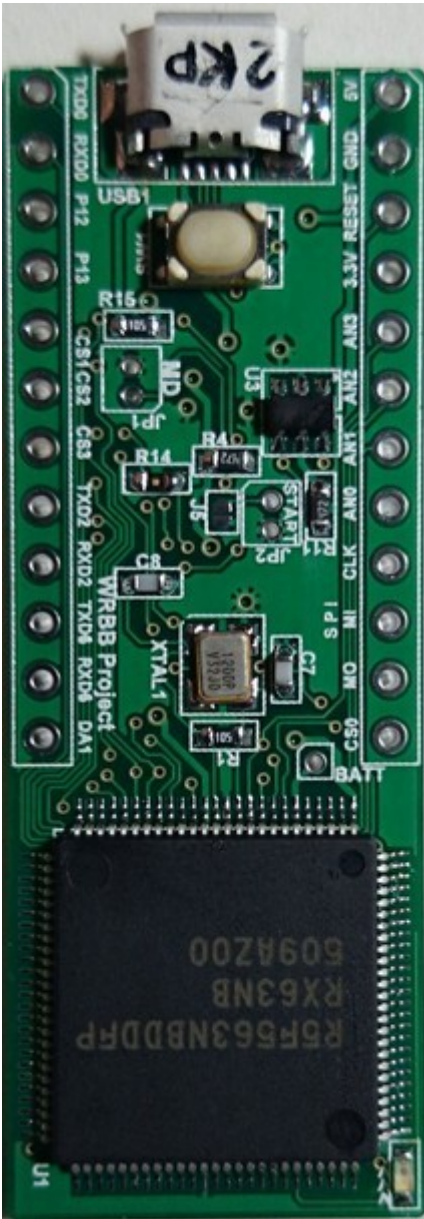
Wakayama. rbボード  
Ver. ARIDA 4  
説明資料 ver1.2

Wakayama. rb  
山本三七男(たろサ)

# Wakayama.rbボード Ver. ARIDA 4

## 特 徴

- ・mrubyを実装したRubyボードです。オブジェクト指向スクリプト言語Rubyを用いてプログラミングできます。作成したプログラムはシリアル経由で書き換えることができます。
- ・頭脳にGR-SAKURA搭載のRX63Nを持ち、ピン配置はGR-KURUMI(ほぼ)互換というガジェルスネ大好きな作者の趣向がもろに出た一品です。



## ハード仕様

### MCU

32ビットCPU RX63N(100ピン)

96MHz

FlashROM : 1Mバイト

RAM : 128Kバイト

データ用Flash : 32Kバイト

### ボード機能

USBファンクション端子 (micro-B)

LED 1個

I/Oピン 20ピン

シリアル 3個(+1個可能)

SPI 1個

A/D 4個

RTC

I2C、PWM、Servoは自由割当てです。

リセットボタン

### 電 源

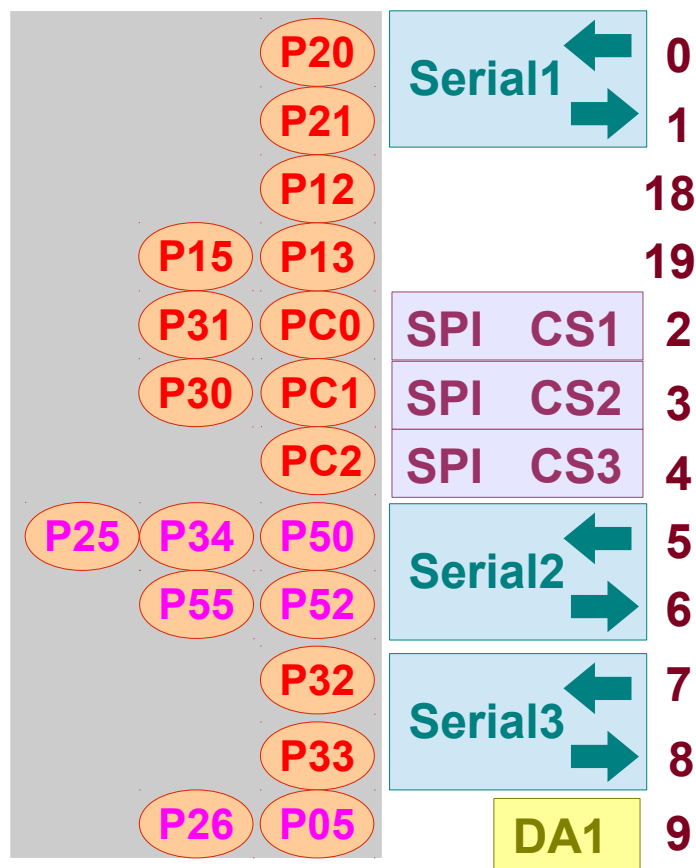
5V (USBバスパワー)

### サイズ

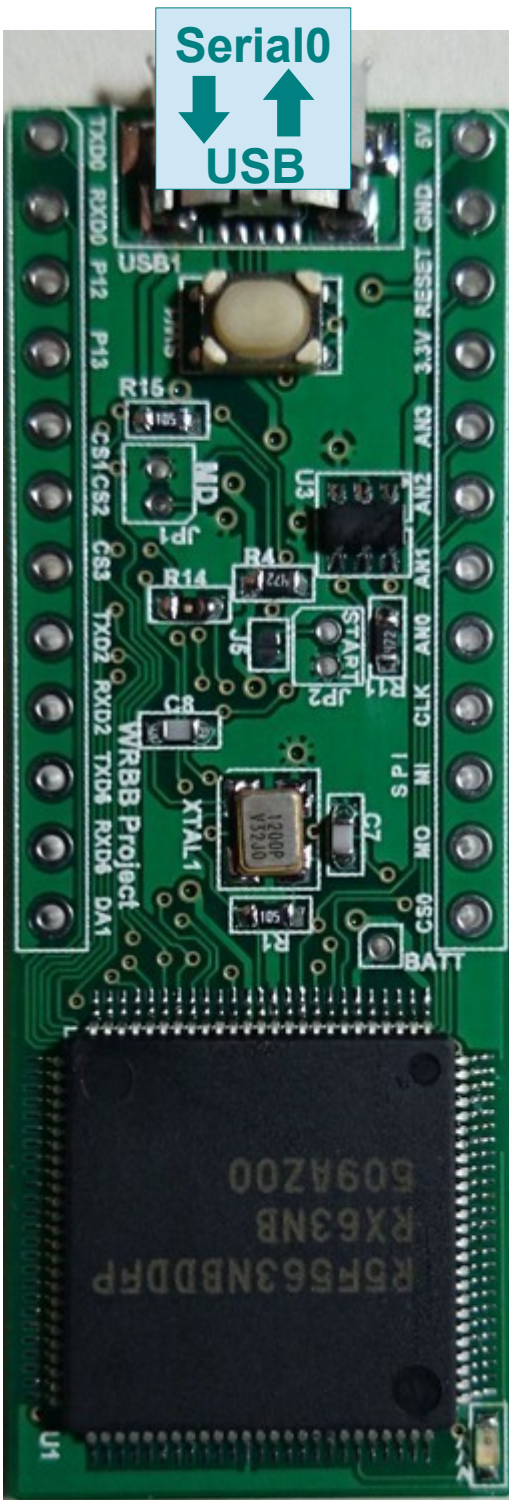
50×18mm



RX63Nピン番号



赤文字ピン番は  
5Vトレラント



Serial0  
↓ ↑  
USB

5V

GND

RESET

3.3V

A3

A2

A1

A0

CLK

12

11

CS0

SPI

P43 PE1

P42 PB5

P41 PB3

P40 P27

PC5

PC7

PC6

PC4

RX63Nピン番号



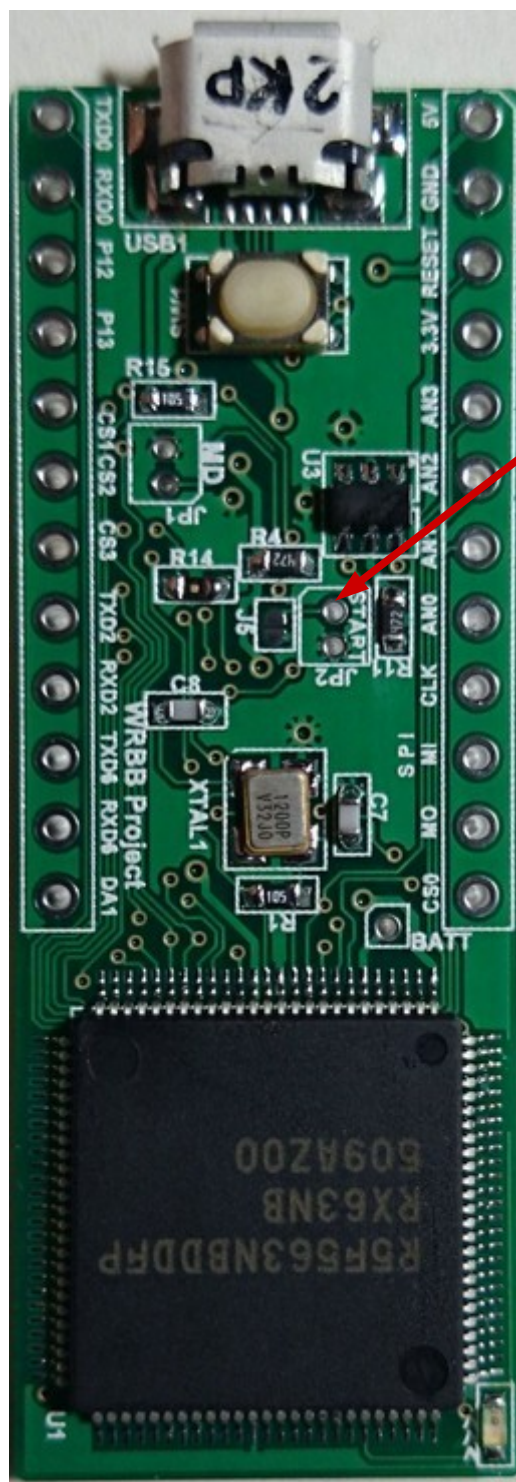
# GR-SAKURAのピン対比

|        |    |
|--------|----|
|        | 1  |
|        | 0  |
|        | 30 |
| 33     | 31 |
| TMS    | 22 |
| TDI    | 23 |
|        | 8  |
| 5 TRST | 24 |
| 29     | 26 |
|        | 6  |
| 7      |    |
| TDO    | 53 |

GR-SAKURA割当番号

|     |     |     |   |
|-----|-----|-----|---|
|     | P20 | 0   |   |
|     | P21 | 1   |   |
|     | P12 | 18  |   |
| P15 | P13 | 19  |   |
| P31 | PC0 | 2   |   |
| P30 | PC1 | 3   |   |
|     | PC2 | 4   |   |
| P25 | P34 | P50 | 5 |
|     | P55 | P52 | 6 |
|     | P32 | 7   |   |
|     | P33 | 8   |   |
| P26 | P05 | 9   |   |

RX63Nピン番号



5V  
GND  
◀ RESET  
3.3V

|    |     |     |
|----|-----|-----|
| 17 | P43 | PE1 |
| 16 | P42 | PB5 |
| 15 | P41 | PB3 |
| 14 | P40 | P27 |
| 13 | PC5 |     |
| 12 | PC7 |     |
| 11 | PC6 |     |
| 10 | PC4 |     |

RX63Nピン番号

|     |    |
|-----|----|
| P35 | 54 |
|-----|----|

|    |     |
|----|-----|
| 17 | 45  |
| 16 |     |
| 15 |     |
| 14 | TCK |
| 13 |     |
| 12 |     |
| 11 |     |
| 10 |     |

GR-SAKURA割当番号

# JTAGの端子

**TMS TDI MD**

**TRST**

**TDO**

**EMLE**



**GND RESET V<sub>3.3</sub>**

**TCK**

**SPIMISO**



# ジャンパの説明

J4

P27(TCK)と14番を接続します。  
P40ともショートになります。

J3

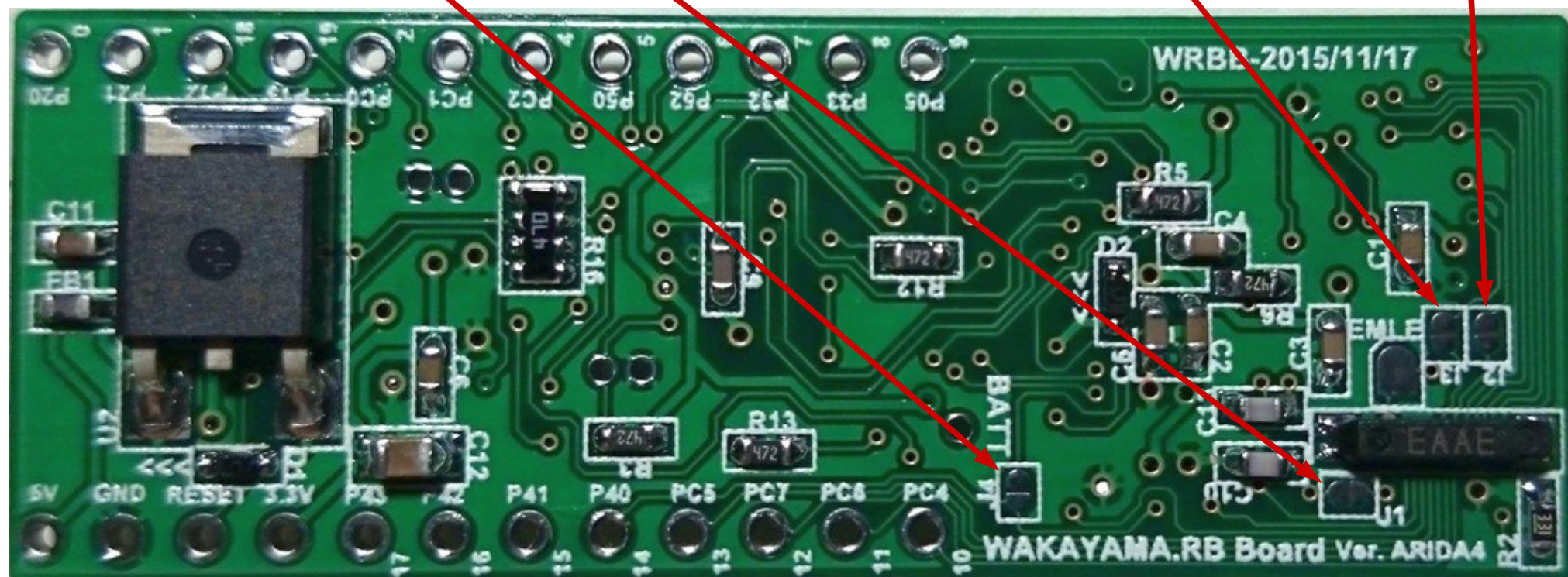
PB3と15番を接続します。  
P41ともショートになります。

J1

PE1と17番を接続します。  
P43ともショートになります。

J2

PB5と16番を接続します。  
P42ともショートになります。



# ジャンパの説明

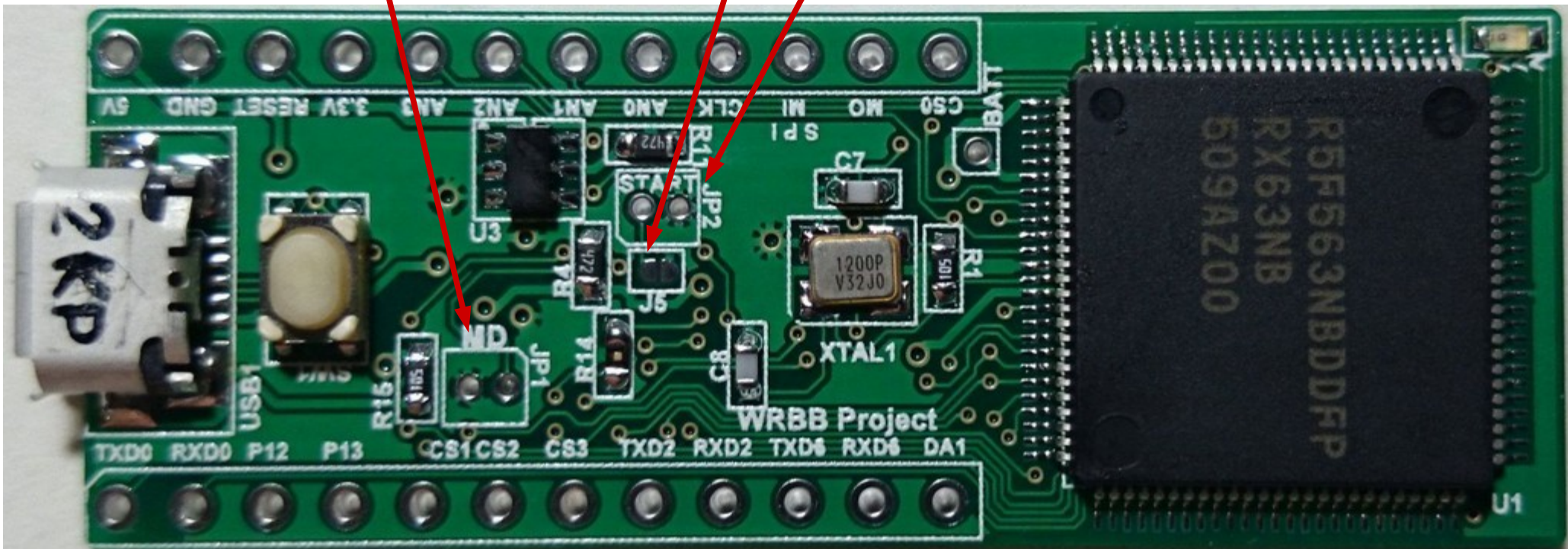
JP1

MDをGNDに落とします。  
ファームを書き換えるときに、GND  
とショートさせます。

J5

P35(NMI)をGNDに落とします。  
電源ON時にデフォルトでGNDにしたいと  
きに使用します。

JP2もJ5と同じです。





## カーネルクラス

- pinMode(pin, mode)
- digitalRead(pin)
- digitalWrite(pin, value)
- analogRead(number)
- pwm(pin, value)
- pwmHz(value)
- analogDac(value)
- delay(value)
- millis()
- micros()
- led(sw)
- tone(pin, freq[, duration])
- noTone(pin)
- randomSeed(value)
- random([min, ] max)

## 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()

## ファイルクラス

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

## シリアルクラス

- Serial.begin(number, bps)
- Serial.setDefault(number)
- Serial.print(number, string)
- Serial.println(number, string)
- Serial.read(number)
- Serial.write(number, buf, len)
- Serial.available(number)
- Serial.end(number)

## I2Cクラス

- I2c.sdasci(sda, scl)
- I2c.write(id, address, data)
- I2c.read(id, addressL[, addressH])
- I2c.begin(id)
- I2c.lwrite(data)
- I2c.end()
- I2c.request(id, count)
- I2c.lread()
- I2c.freq(Hz)

# 基本ソフト仕様 (V1ライブラリ)

## 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.detach(ch)`

## リアルタイムクロッククラス

`Rtc.begin()`

`Rtc.setTime(Array)`

`Rtc.getTime()`

# ***ruby*プログラムの実行**

WRBボードは、内部にrubyプログラムを保存できます。ファイル形式はmrbcによりコンパイルしたmrb形式のファイルとなります。

WRBボードは、後述する「電源オンで即実行するモード」に切り替わっていない限り、通常、電源をオンするとコマンドモードとなります。

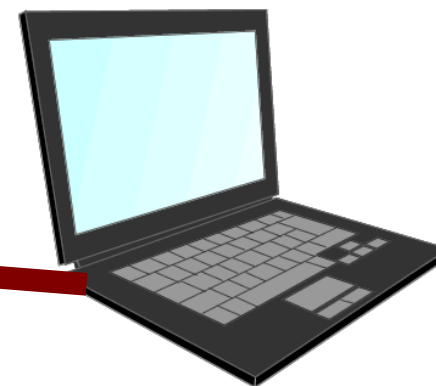


# プログラムの書き込み

WRBボードはPCとUSB経由で接続し、シリアル通信を用いて通信します。  
この通信を使って、Rubyのプログラムを書き込んだり、実行したり、WRB  
ボードからデータをPCに出力したりします。



**シリアル通信**



CoolTerm



TeraTerm

シリアル通信には、ターミナルソフトを使います。

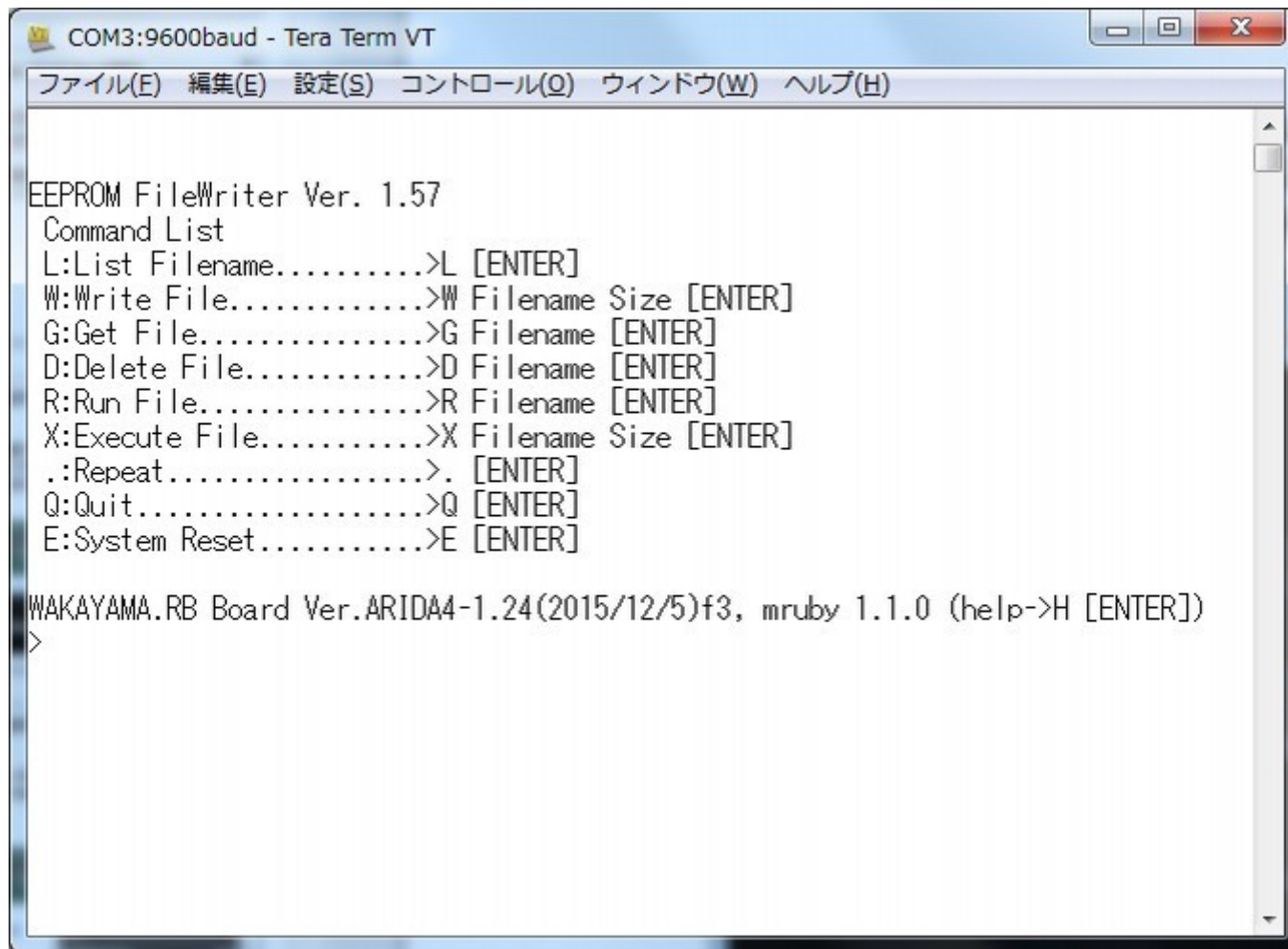
代表的なものにTeraTermやCoolTerm  
があります。



# プログラムの書き込み方法

ターミナルソフトを用いてUSBからシリアル通信をしてプログラムを書き込みます。  
ENTERキーで画面にコマンド一覧が表示されます。

アルファベット1文字のコマンドを持っています。



```
COM3:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

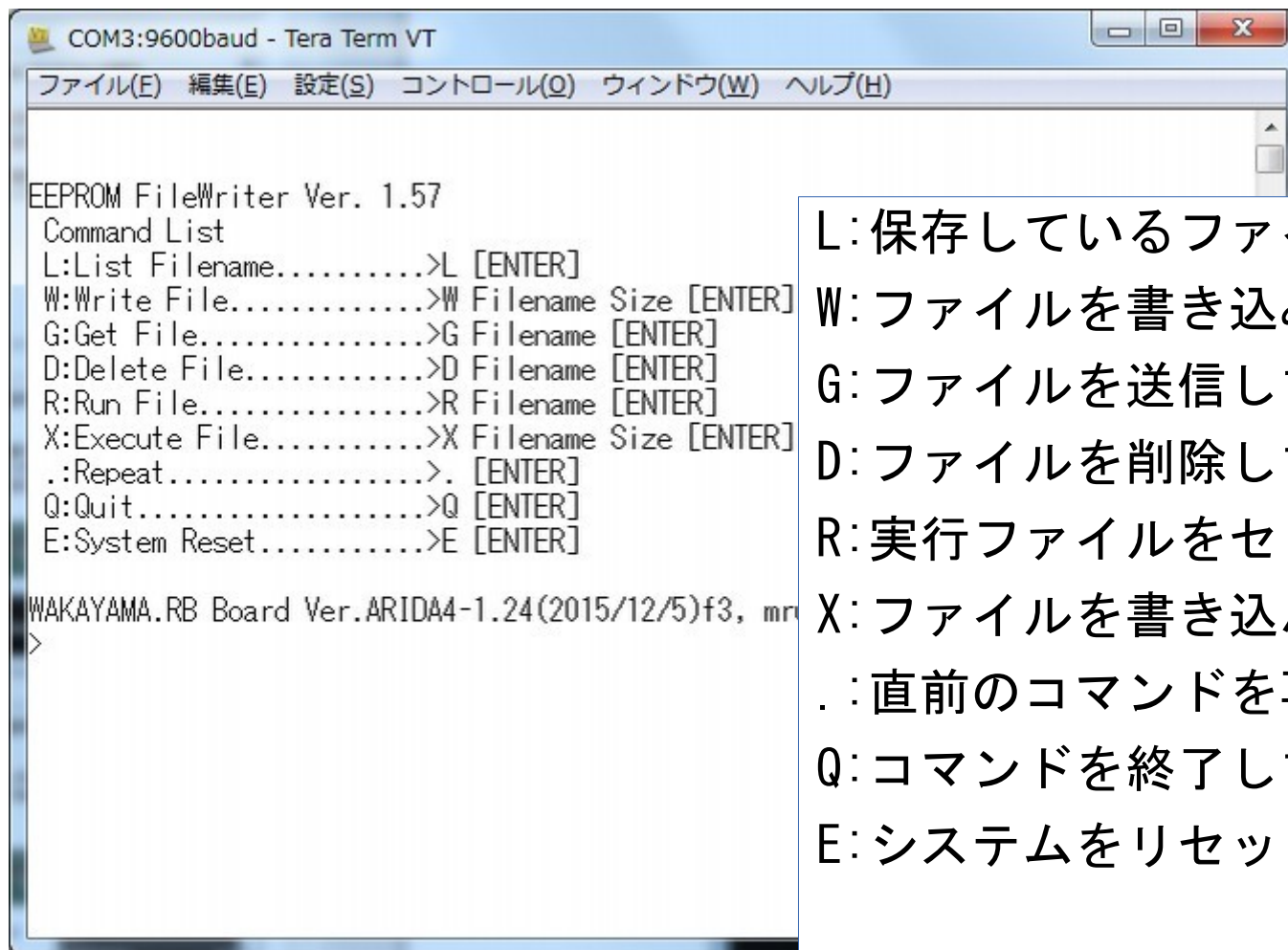
EEPROM FileWriter Ver. 1.57
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]

WAKAYAMA.RB Board Ver.ARID44-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
```

WRBボードの起動画面



## コマンドの種類



L:保存しているファイルを一覧します。

W:ファイルを書き込みます。

G:ファイルを送信します。

D:ファイルを削除します。

R:実行ファイルをセットします。

X:ファイルを書き込んで直ぐ実行します。

.:直前のコマンドを再実行します。

Q:コマンドを終了します。

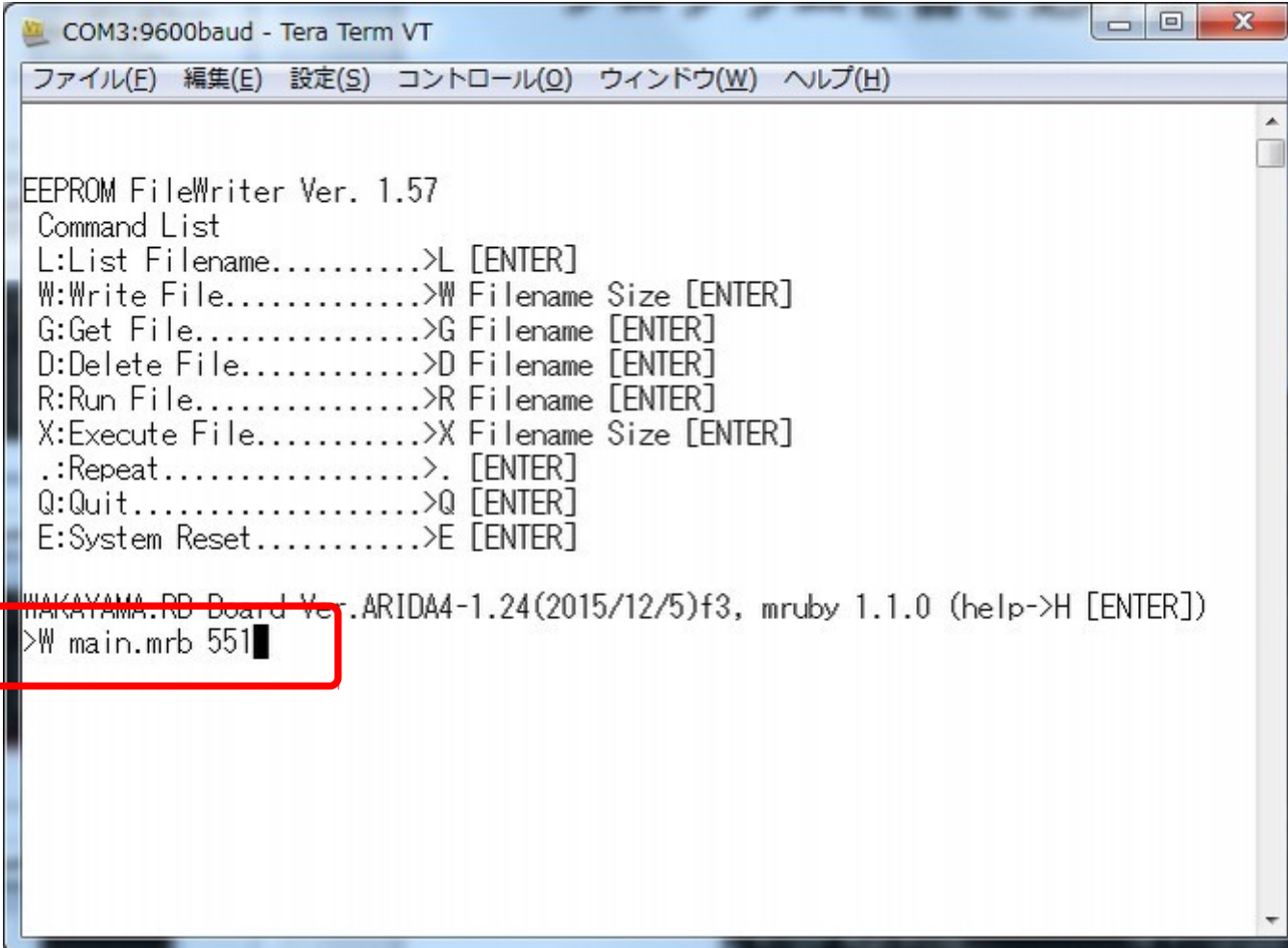
E:システムをリセットします。

## プログラムを書き込みます。(W コマンド)

Wコマンドを用いて、mrbファイルを書き込みます。

Wの後にスペースで区切って、ファイル名とファイルサイズを書き、ENTERキーを押します。

>W ファイル名 ファイルサイズ



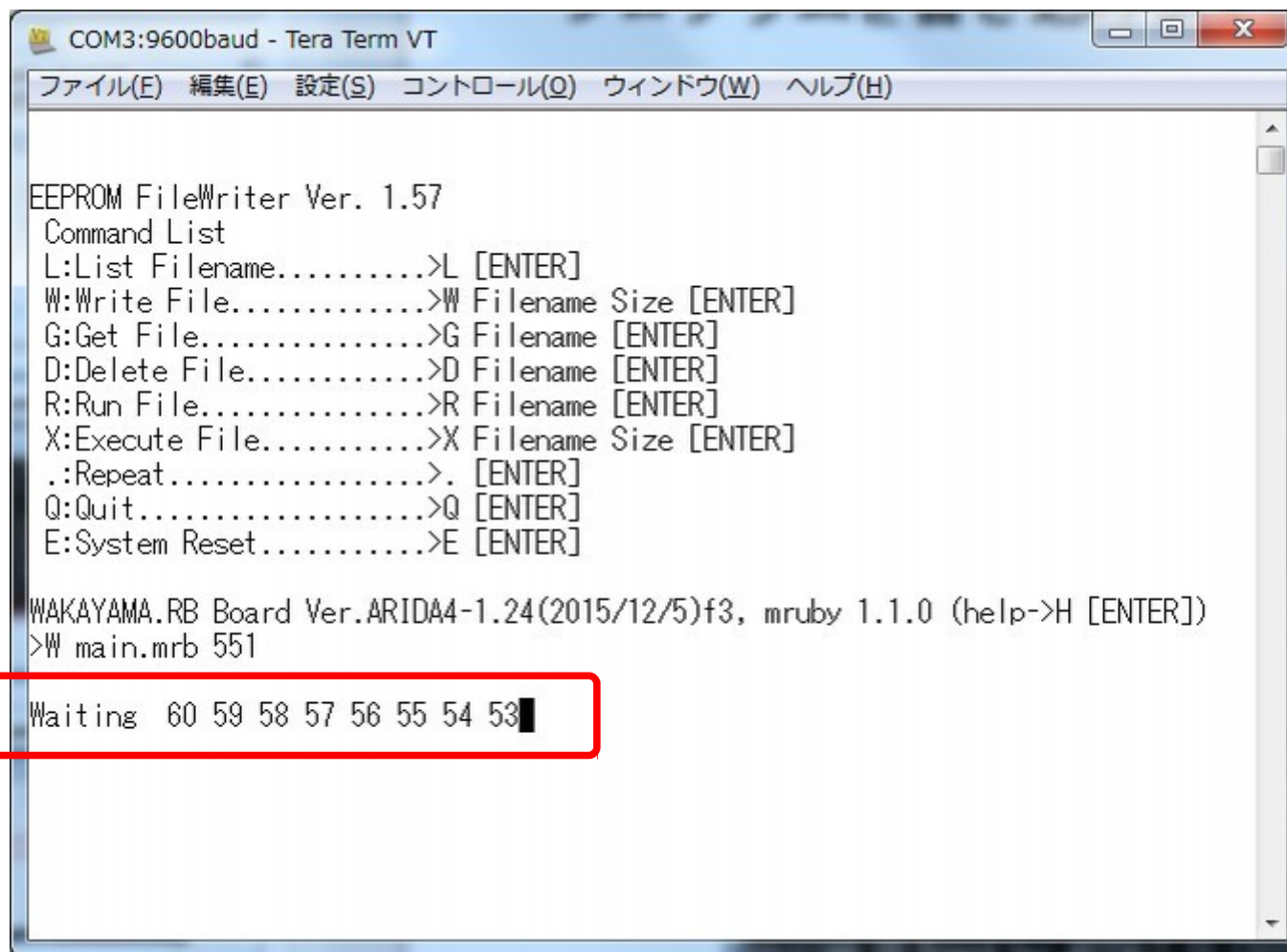
```
COM3:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

EEPROM FileWriter Ver. 1.57
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]

WAKAYAMA.RD Board Ver..ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>W main.mrb 551
```

## プログラムを書き込みます。(W コマンド)

ENTERキーを押すと、カウントダウンが始まります。60sec以内にファイルをバイナリ送信してください。



COM3:9600baud - Tera Term VT

ファイル(F) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)

EEPROM FileWriter Ver. 1.57

Command List

L:List Filename.....>L [ENTER]

W:Write File.....>W Filename Size [ENTER]

G:Get File.....>G Filename [ENTER]

D>Delete File.....>D Filename [ENTER]

R:Run File.....>R Filename [ENTER]

X:Execute File.....>X Filename Size [ENTER]

.:Repeat.....>. [ENTER]

Q:Quit.....>Q [ENTER]

E:System Reset.....>E [ENTER]

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])

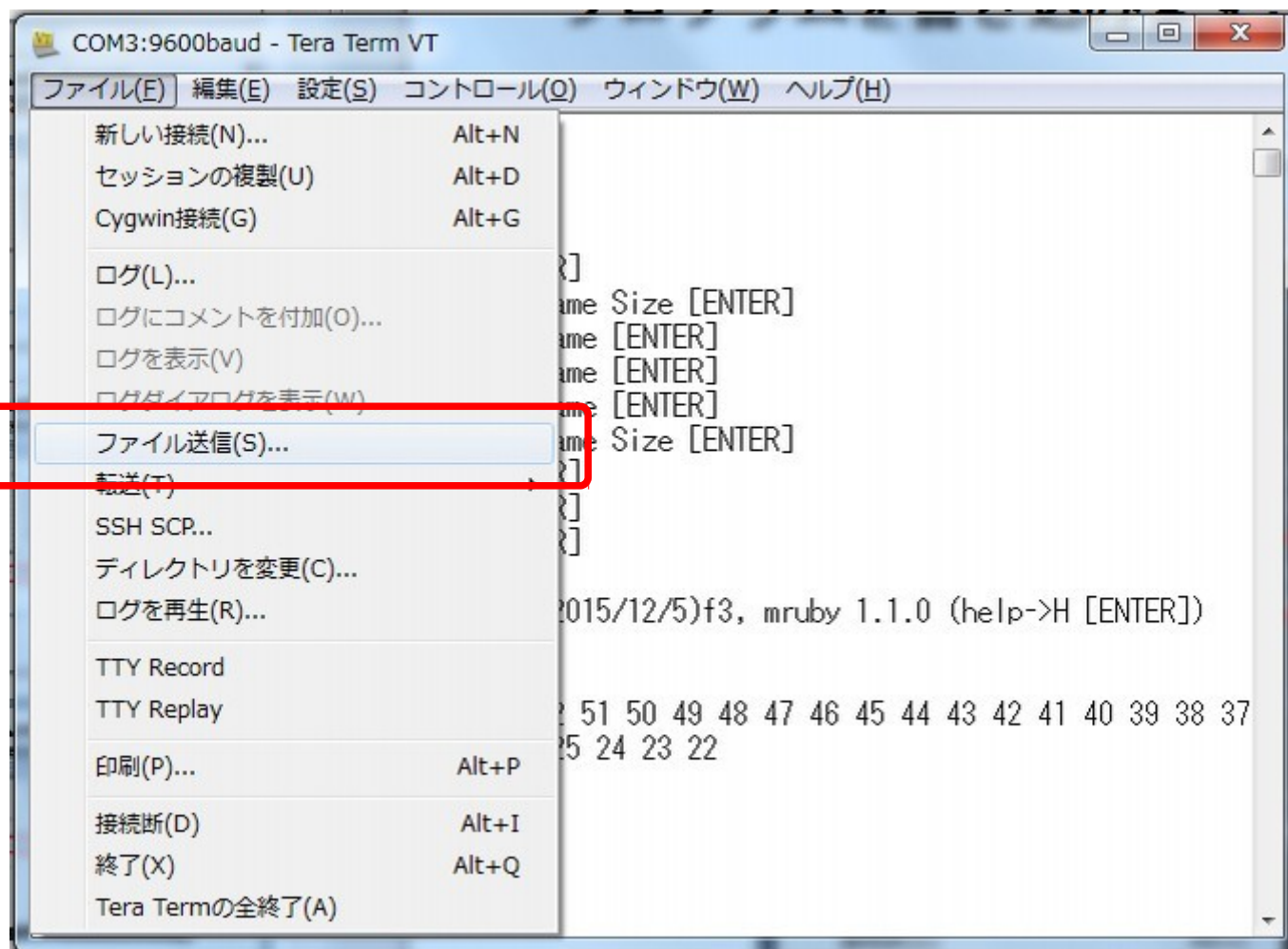
>W main.mrb 551

Waiting 60 59 58 57 56 55 54 53



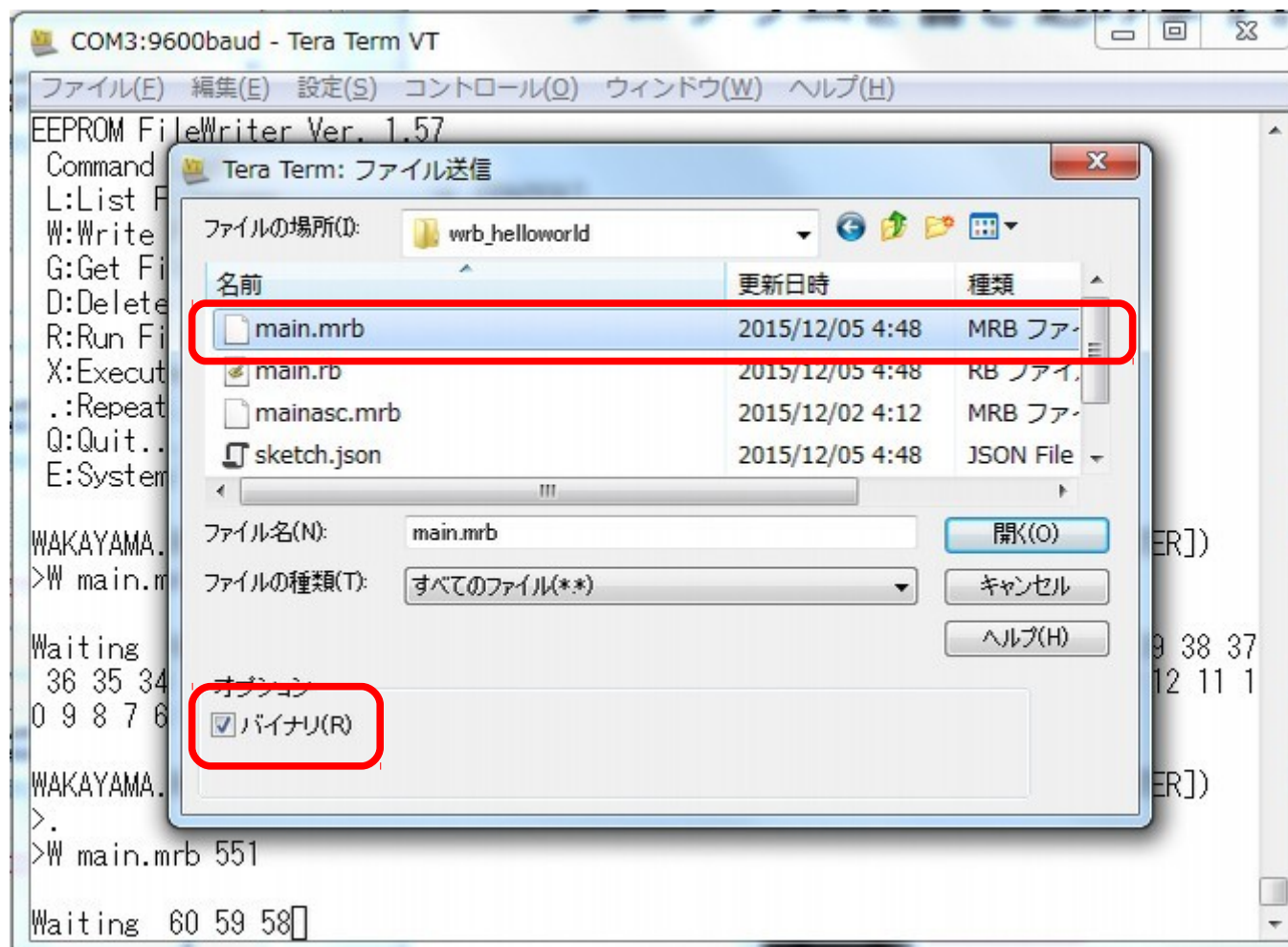
## プログラムを書き込みます。(W コマンド)

Tera Termの場合、ファイル→ファイル送信 を選択します。



## プログラムを書き込みます。(W コマンド)

Tera Termの場合、オプションのバイナリにチェックを入れます。  
その後、送信するファイルを選択して、開く を押します。



## プログラムを書き込みます。(W コマンド)

ファイルの書き込みが終了すると、コマンド入力待ちに戻ります。

```
COM3:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>W main.mrb 551

Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 1
0 9 8 7 6 5 4 3 2 1 0..Wait Error!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>W main.mrb 551

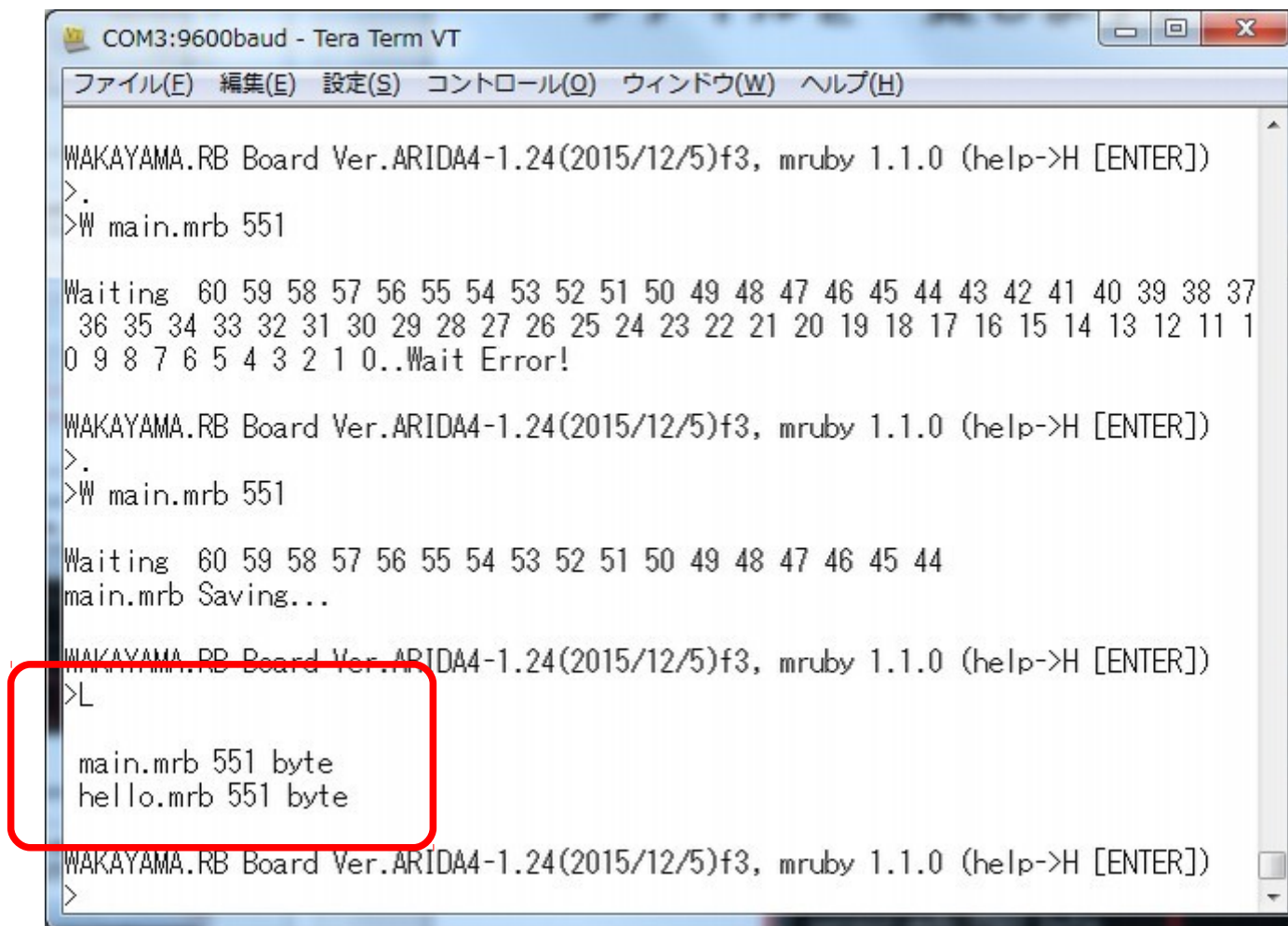
Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 1
0 9 8 7 6 5 4 3 2 1 0..Wait Error!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>W main.mrb 551
Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44
main.mrb Saving...

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
```

## ファイルを一覧します。(L コマンド)

Lコマンドを入力すると保存されているファイルの一覧が表示されます。



The screenshot shows a Tera Term VT window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The terminal output shows the following sequence of commands and responses:

```
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>W main.mrb 551

Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 1
0 9 8 7 6 5 4 3 2 1 0..Wait Error!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>W main.mrb 551

Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44
main.mrb Saving...

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>L

main.mrb 551 byte
hello.mrb 551 byte

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
```

The command ">L" and its output are highlighted with a red rectangle.



## mrubyファイルを実行します。(R コマンド)

Rコマンドは、mrubyファイルを実行することができます。

Rの後にスペースで区切って、実行させたいファイル名を書き、ENTERを押します。

.mrubyは省略可能です。

>R ファイル名

```
COM3:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

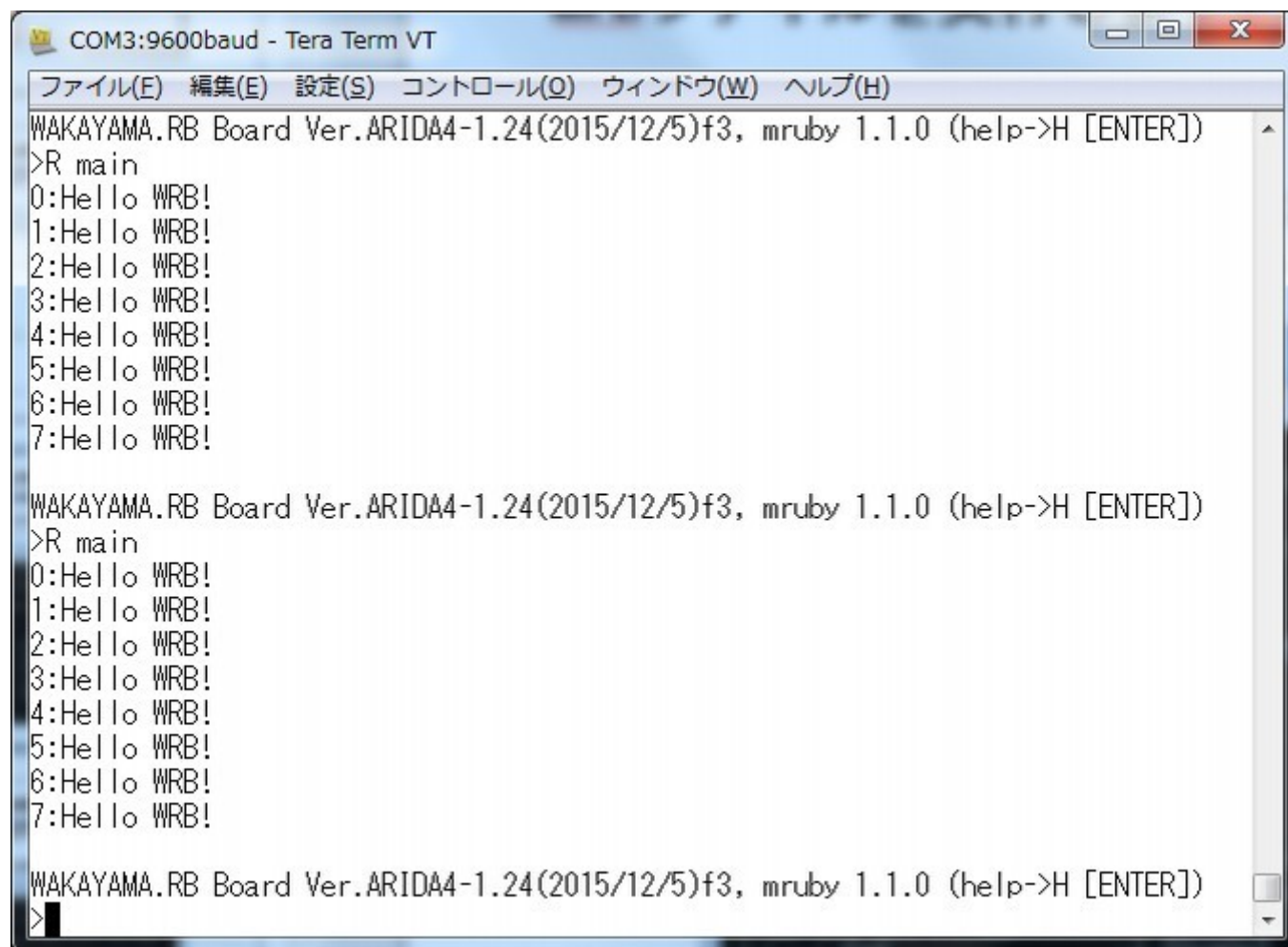
main.mrb 551 byte
hello.mrb 551 byte

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
```

## mrubyファイルを実行します。(R コマンド)

実行が終了するとコマンドモードに戻ります。



The screenshot shows a Tera Term window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The main text area displays the output of an mruby script execution. The script starts with a header line: "WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])". This is followed by a prompt ">R main". The script then outputs eight lines: "0:Hello WRB!", "1:Hello WRB!", "2:Hello WRB!", "3:Hello WRB!", "4:Hello WRB!", "5:Hello WRB!", "6:Hello WRB!", and "7:Hello WRB!". After this, the header line is repeated, followed by another ">R main" prompt. The script then outputs the same eight lines again. Finally, the header line is repeated, followed by a ">" prompt, indicating the return to the command mode.

```
COM3:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
```

## プログラムを書き込み実行します。(X コマンド)

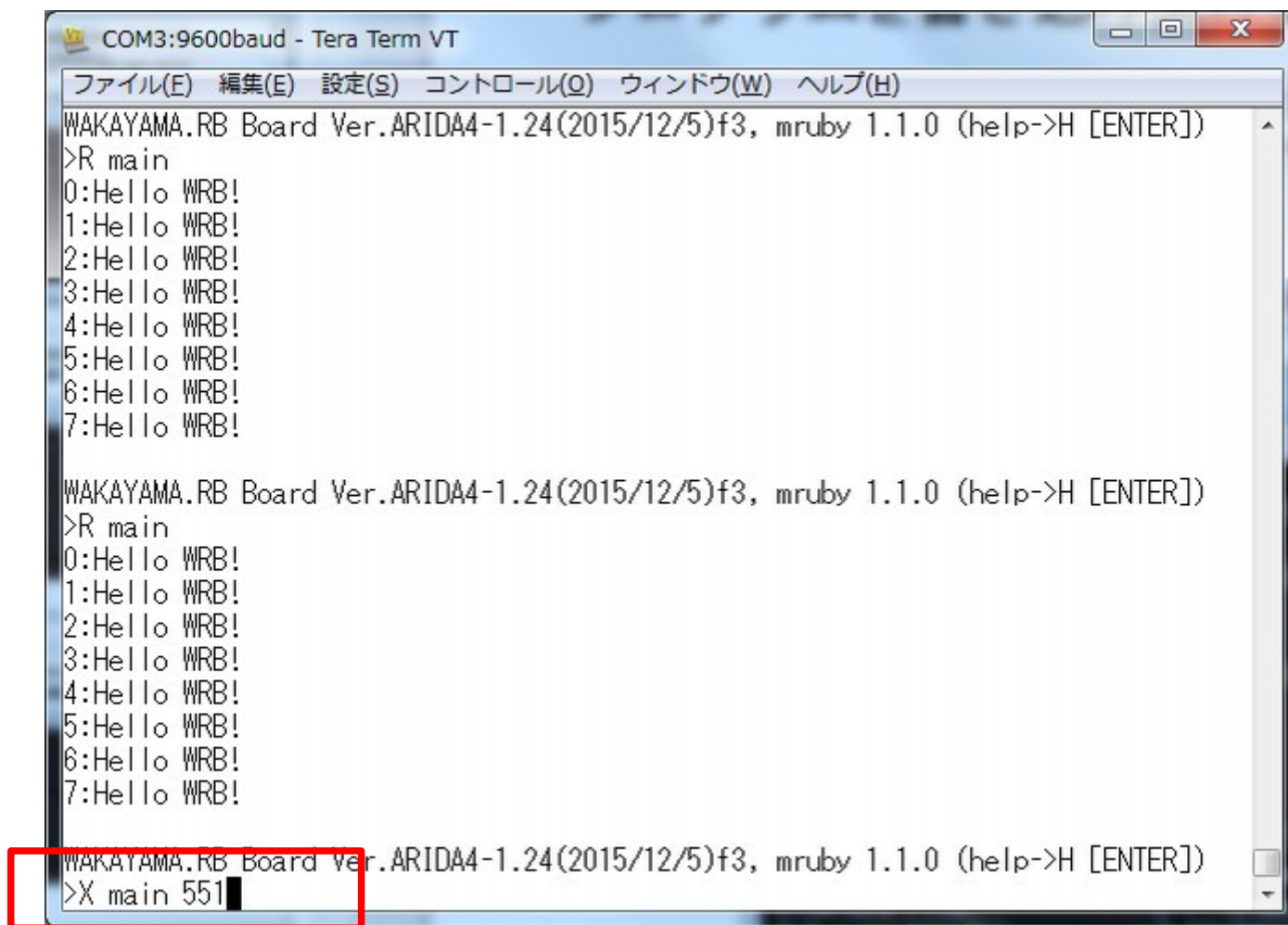
Xコマンドを用いて、mrbファイルを書き込み後直ぐ実行します。

Xの後にスペースで区切って、ファイル名とファイルサイズを書き、ENTERキーを押します。

.mrbは省略可能です。

>X ファイル名 ファイルサイズ

あとは、Wコマンドと同様です。プログラムの書き込みが終了後、直ぐに実行されます。



The screenshot shows a Tera Term window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The terminal output shows the following sequence of commands and responses:

```
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

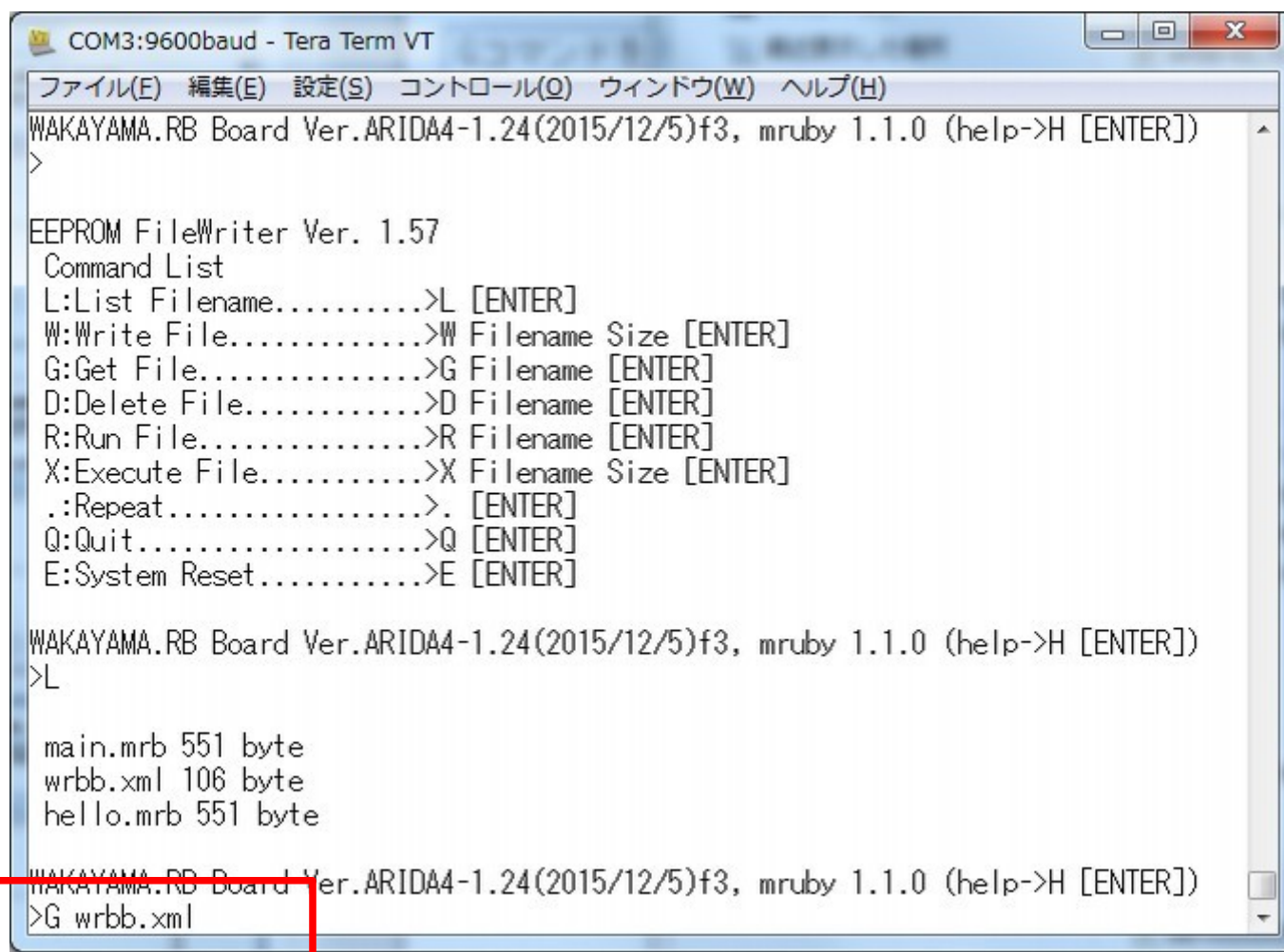
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>X main 551
```

The last line, ">X main 551", is highlighted with a red rectangle.

## ファイルを送信します。(G コマンド)

Gコマンドを用いるとWRBボードに保存されているファイルをPCに読み出すことができます。  
Gの後にスペースで区切って、ファイル名を書き、ENTERキーを押します。

### >G ファイル名



```
COM3:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(Q) ウィンドウ(W) ヘルプ(H)
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
EEPROM FileWriter Ver. 1.57
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
D:Delete File.....>D Filename [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>L
main.mrb 551 byte
wrbb.xml 106 byte
hello.mrb 551 byte
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>G wrbb.xml
```



## ファイルを送信します。(G コマンド)

ENTERキーを押すと、カウントダウンが始まります。60sec以内にPCから1バイト送信すると、送信するファイルのファイルサイズが送信されます。

そして、再びカウントダウンが始まります。

```
COM3:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>L

main.mrb 551 byte
wrbb.xml 106 byte
hello.mrb 551 byte

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>G wrbb.xml

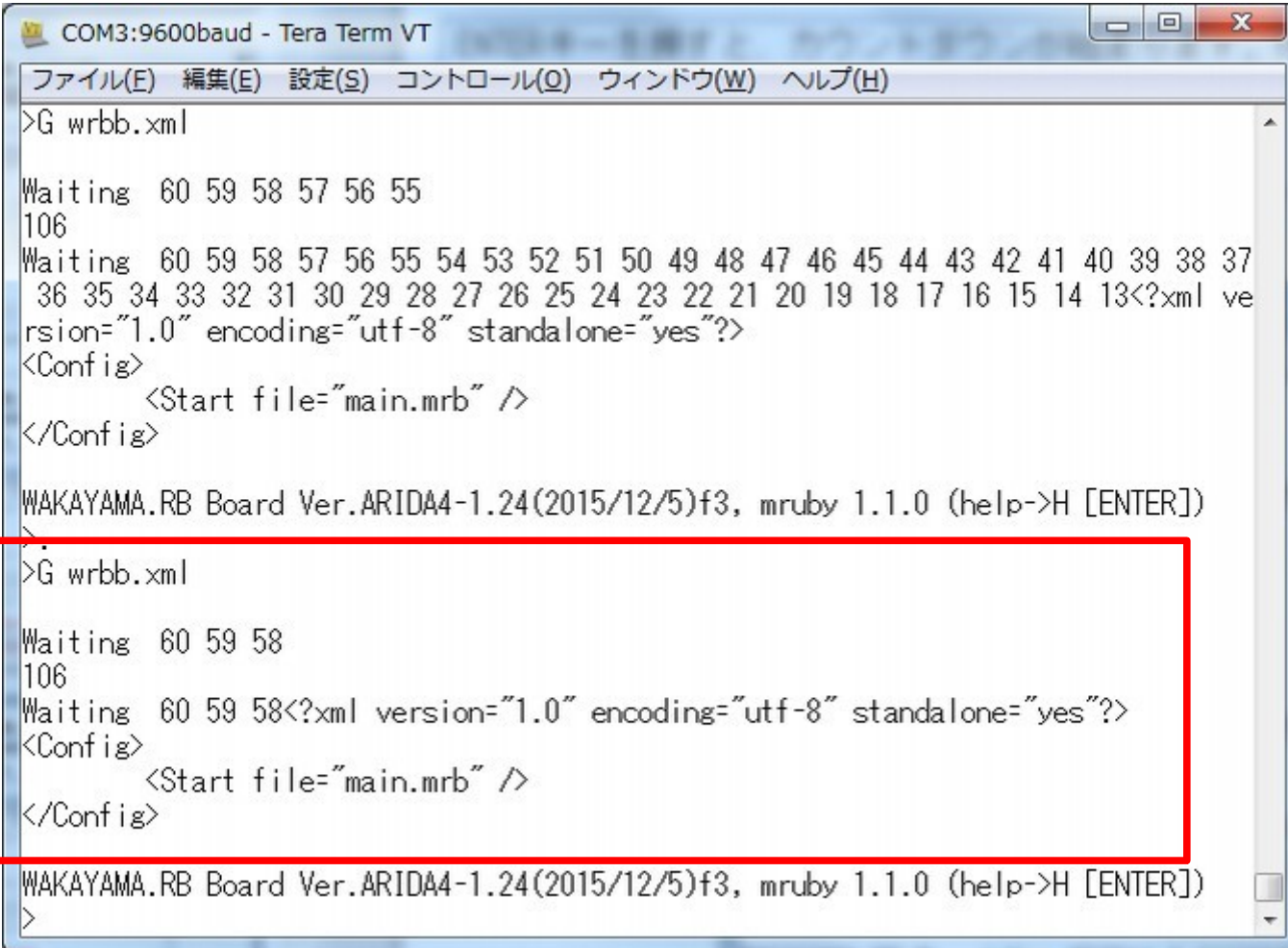
Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10
9 8 7 6 5 4 3 2 1 0..Wait Error!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>G wrbb.xml

Waiting 60 59 58 57 56 55
106
Waiting 60 59 58 57 56 55 54 53
```

## ファイルを送信します。(G コマンド)

2回目のカウントダウンの60sec以内にPCから1バイト送信すると、指定したファイルが送信されます。  
バイナリファイルの場合もバイナリのまま送信されます。



The screenshot shows a Tera Term VT window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The terminal output shows the following sequence of events:

```
>G wrbb.xml  
Waiting 60 59 58 57 56 55  
106  
Waiting 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37  
36 35 34 33 32 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13<?xml ve  
rsion="1.0" encoding="utf-8" standalone="yes"?>  
<Config>  
    <Start file="main.mrb" />  
</Config>  
WAKAYAMA.RB Board Ver.ARID44-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])  
>  
>G wrbb.xml  
Waiting 60 59 58  
106  
Waiting 60 59 58<?xml version="1.0" encoding="utf-8" standalone="yes"?>  
<Config>  
    <Start file="main.mrb" />  
</Config>  
WAKAYAMA.RB Board Ver.ARID44-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])  
>
```

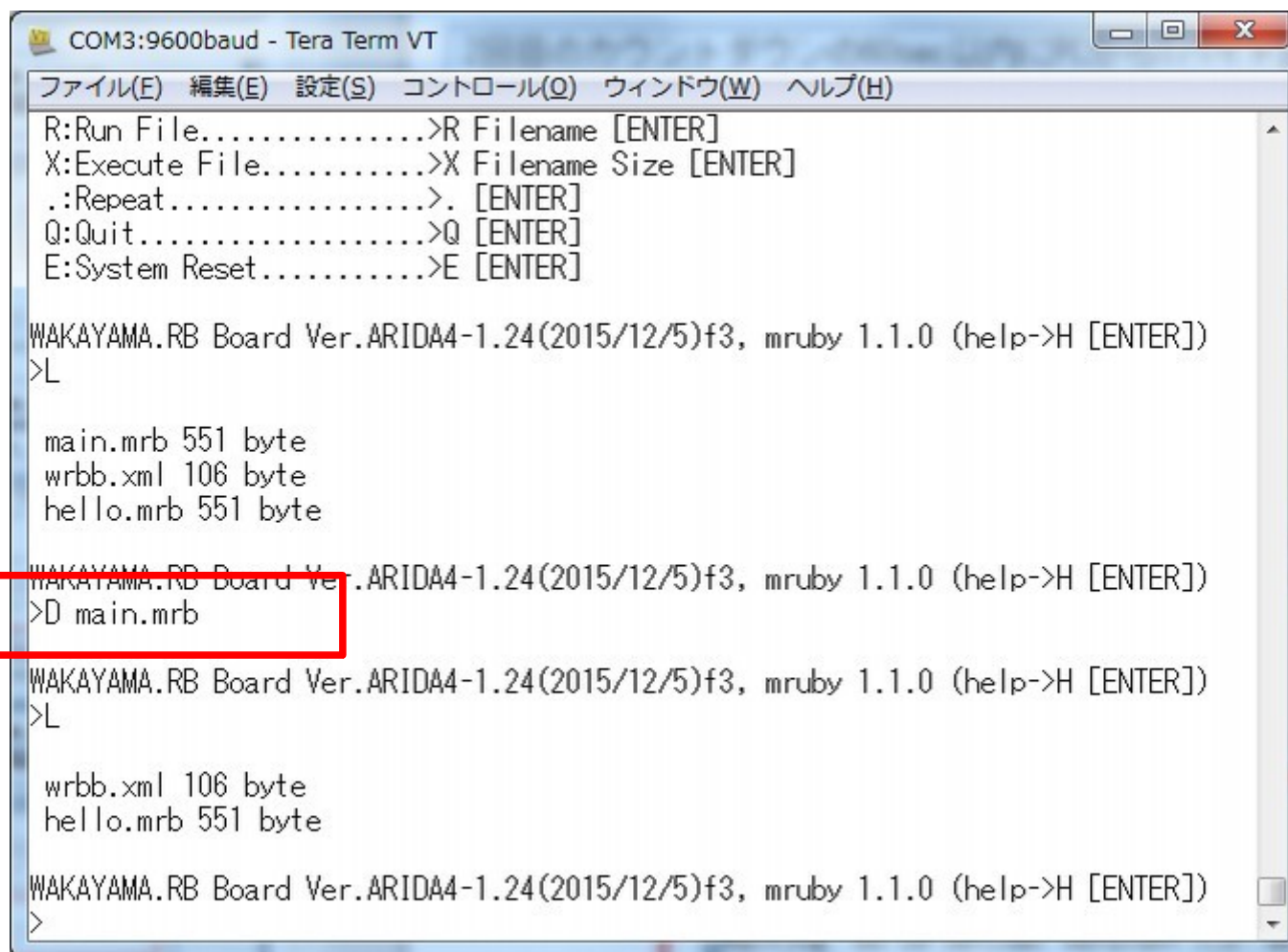
A red rectangular box highlights the second transmission attempt, starting from the command ">G wrbb.xml" and ending with the prompt ">".

## ファイルを削除する。(D コマンド)

DコマンドはWRBボード保存しているファイルを削除します。

Dの後にスペースで区切って、ファイル名を書き、ENTERキーを押します。

>D ファイル名



The screenshot shows a Tera Term window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(Q)", "ウィンドウ(W)", and "ヘルプ(H)". The command history shows the following sequence of commands and responses:

```
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>L

main.mrb 551 byte
wrbb.xml 106 byte
hello.mrb 551 byte

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>D main.mrb

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>L

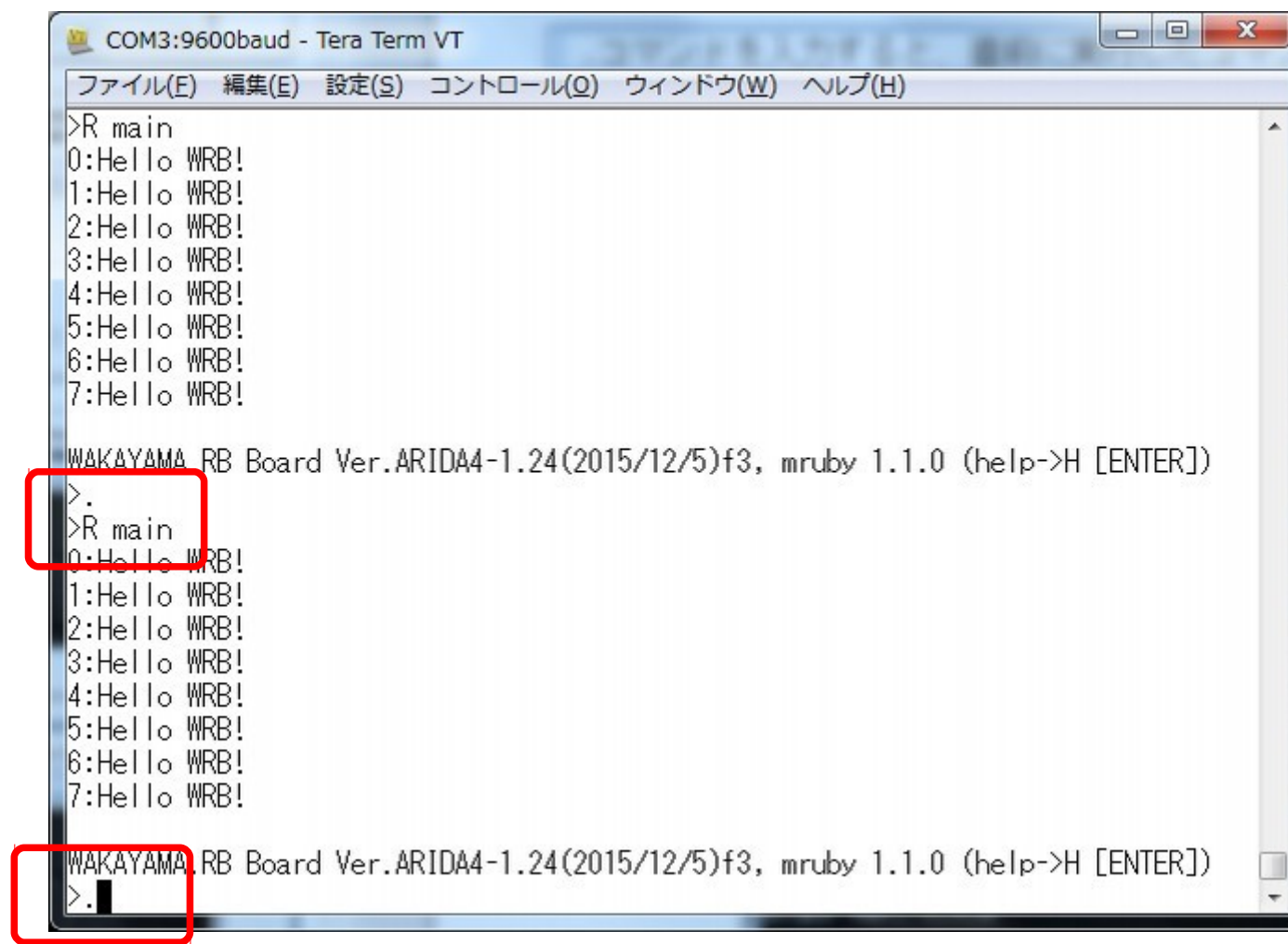
wrbb.xml 106 byte
hello.mrb 551 byte

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>
```

The command ">D main.mrb" is highlighted with a red rectangle.

## コマンド再実行する。( . コマンド)

. コマンドを入力すると、直前に実行したコマンドを再実行します。



The screenshot shows a Tera Term VT window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(E)", "編集(E)", "設定(S)", "コントロール(Q)", "ウィンドウ(W)", and "ヘルプ(H)". The terminal output shows a sequence of commands and responses:

```
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

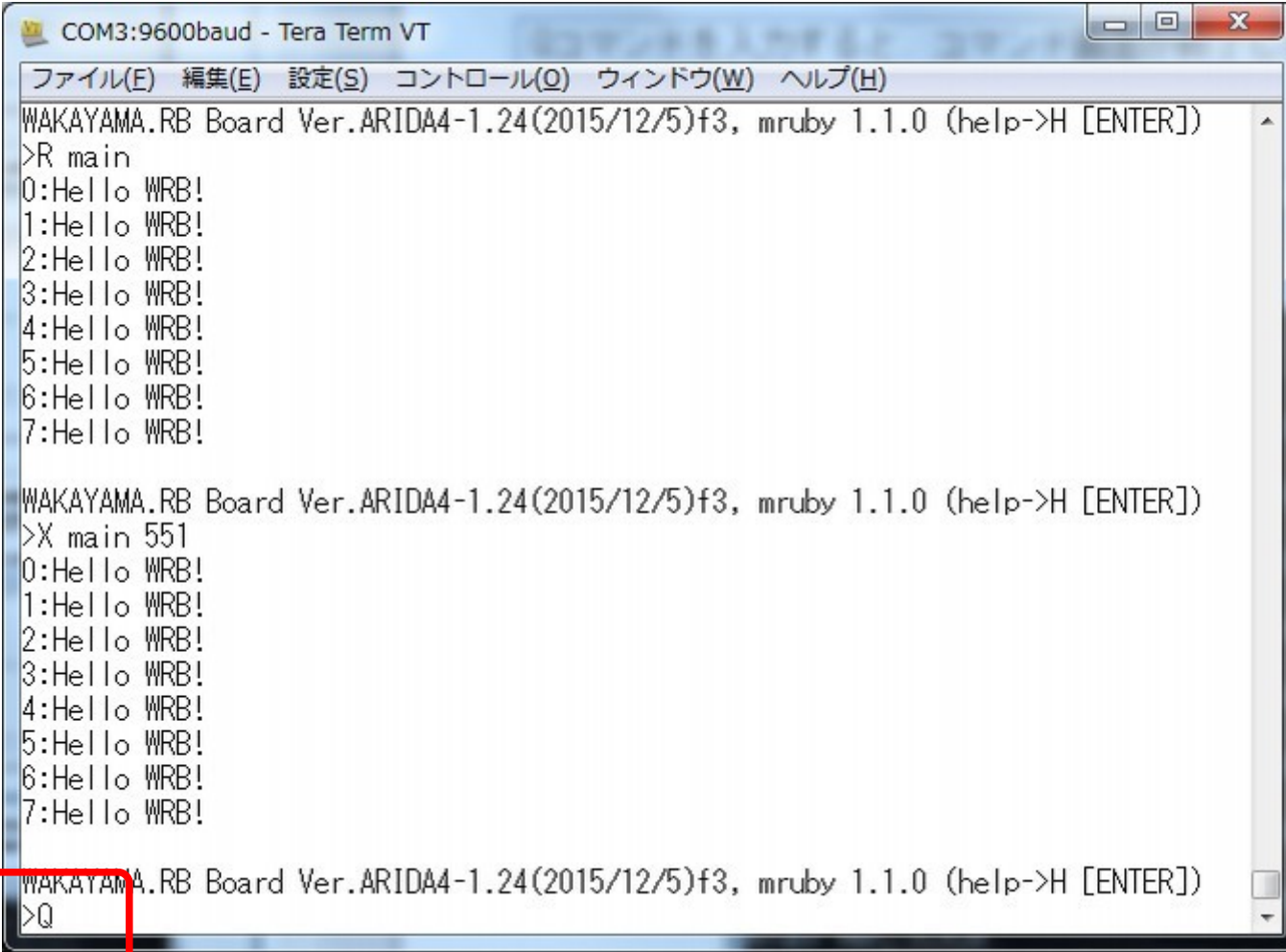
WAKAYAMA RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.
```

Two red boxes highlight the input of the dot command (.) in the second and third instances, demonstrating its use to re-execute the previous command.



## コマンド画面を終了する。(Q コマンド)

Qコマンドを入力すると、コマンド画面が終了します。プログラムの途中で呼び出されている場合は、元のプログラムに戻ります。



The screenshot shows a Tera Term window titled "COM3:9600baud - Tera Term VT". The menu bar includes "ファイル(F)", "編集(E)", "設定(S)", "コントロール(O)", "ウィンドウ(W)", and "ヘルプ(H)". The main text area displays the output of a program running on a WAKAYAMA.RB Board. The program starts with a prompt ">R main" and outputs "0:Hello WRB!" through "7:Hello WRB!". Then, it prompts ">X main 551" and repeats the same output. Finally, it prompts ">Q", which is highlighted by a red rectangle, indicating the command to exit the command screen.

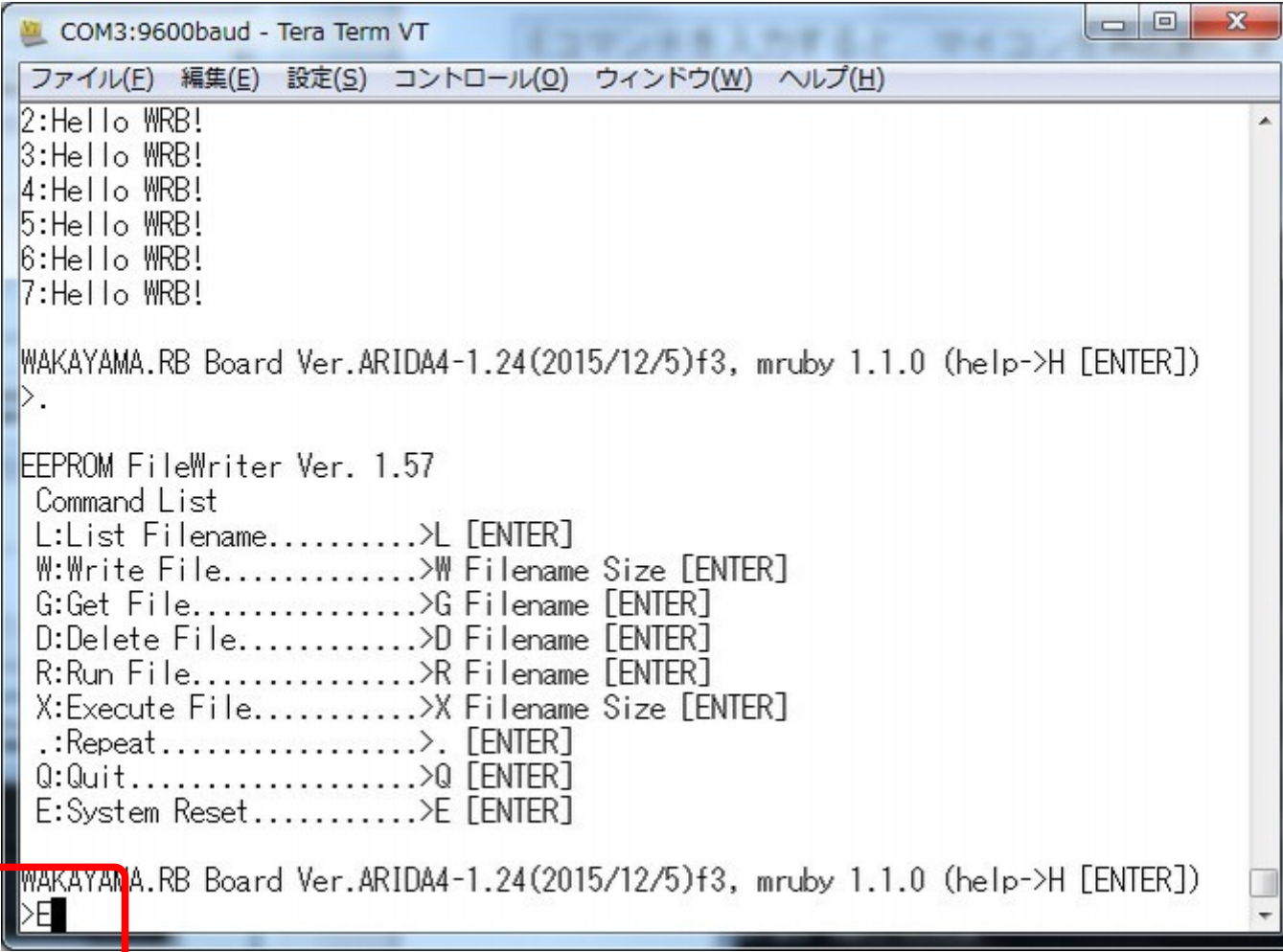
```
COM3:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>R main
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>X main 551
0:Hello WRB!
1:Hello WRB!
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>Q
```

## 再起動する。(E コマンド)

Eコマンドを入力すると、マイコンを再起動します。



```
COM3:9600baud - Tera Term VT
ファイル(E) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)
2:Hello WRB!
3:Hello WRB!
4:Hello WRB!
5:Hello WRB!
6:Hello WRB!
7:Hello WRB!

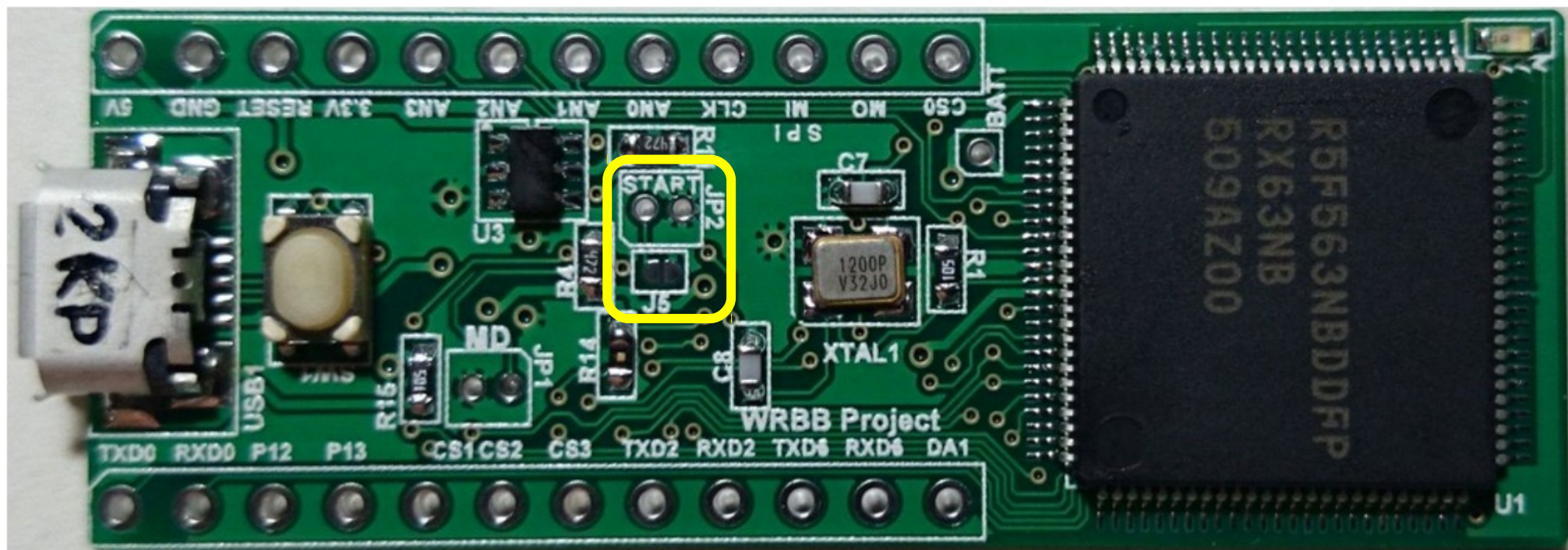
WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>.

EEPROM FileWriter Ver. 1.57
Command List
L:List Filename.....>L [ENTER]
W:Write File.....>W Filename Size [ENTER]
G:Get File.....>G Filename [ENTER]
D>Delete File.....>D Filename [ENTER]
R:Run File.....>R Filename [ENTER]
X:Execute File.....>X Filename Size [ENTER]
.:Repeat.....>. [ENTER]
Q:Quit.....>Q [ENTER]
E:System Reset.....>E [ENTER]

WAKAYAMA.RB Board Ver.ARIDA4-1.24(2015/12/5)f3, mruby 1.1.0 (help->H [ENTER])
>E
```

## 電源ONで即実行する方法

電源をONしてプログラムを即実行したい場合は、J5をショートさせるか、JP2にハーフピッチジャンパを取り付けて、ジャンパをショートさせてください。



WRBボードにwrbb.xmlファイルがあり、Startタグで開始プログラム名が書かれているときには、該当プログラムを実行します。無い場合はmain.mrbが実行されます。main.mrbが無い場合は、コマンドモードになります。

# rubyプログラム自動実行の仕組み

自動実行する場合のrubyプログラム実行条件  
条件(1)

WRBボードは、先ず wrbb.xml ファイルを検索します。wrbb.xmlとはXML形式で書かれたファイルです。

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Config>
  <Start file="wrbb.mrb" />
</Config>
```

Startタグのfile要素に実行するmrbファイル名を書いておくと、そのプログラムを実行します。



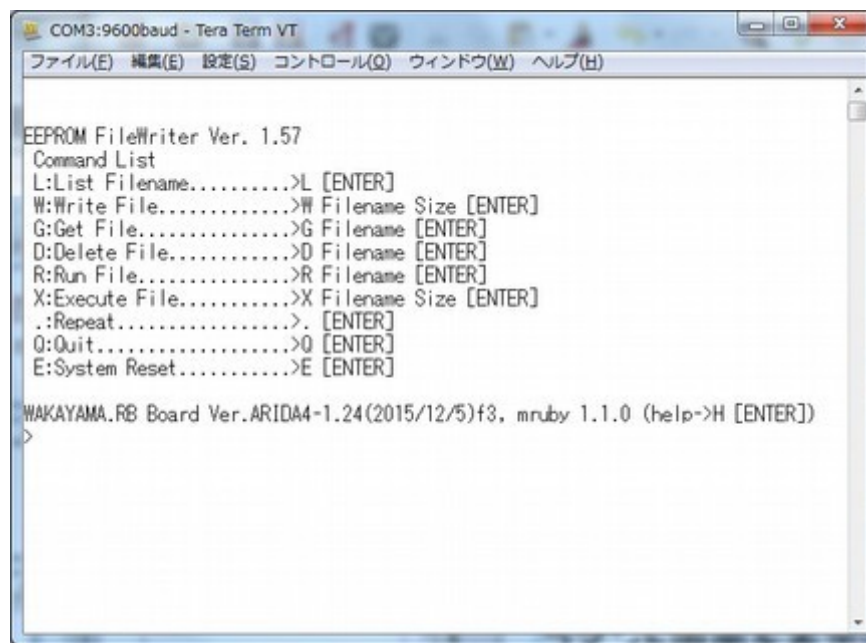
# rubyプログラム自動実行の仕組み

## 条件(2)

wrb.xml ファイルが見つからない場合は、main.mrbファイルを検索します。  
main.mrb ファイルが見つければ、main.mrbファイルを実行します。

## 条件(3)

wrb.xml、main.mrb 両方のファイルが見つからない場合は、USB接続先に  
コマンド画面を表示します。



# rubyプログラム実行の仕組み

## rubyプログラム例

LEDを5回 ON/OFFさせます。

```
sw = 1
10.times do
  led(sw)
  sw = 1 - sw
  delay(500)
end
```

以下のように書いても同じです。

```
sw = 1
for i in 1..10 do
  led(sw)
  sw = 1 - sw
  delay(500)
end
```

Hello WAKAYAMA.RB Board!と10回出力されます。

```
10.times do
  Serial.println(0, "Hello WAKAYAMA.RB Board!")
  delay(500)
end
```

# rubyプログラム実行の仕組み

rubyプログラム中に `System.setrun` 命令を用いて、次に呼び出すrubyプログラムを指定しておく、実行が終了後、指定されたrubyプログラムが呼び出されます。

main.mrb 実行

```
sw = 1
10.times do
  led(sw)
  sw = 1 - sw
  delay(500)
end
System.setrun("hello.mrb")
```

hello.mrb 実行

```
Serial.println(0,"Hello WAKAYAMA.RB Board!")
led(1)
```

hello.mrb 終了

## mrbcファイルの作成方法

コマンドラインからmrbcを実行してください。

```
$ ./mrbc main.rb
```

```
$ ls -l main.mrb
```

```
----rwx---+ 1 minao None 865 6月 26 23:29 main.mrb
```

プログラムをデバッグしたい場合は、コンパイルオプションに `-g` を付けてコンパイルすることをお勧めします。エラーの行番号など詳しいエラーメッセージが出力されます。

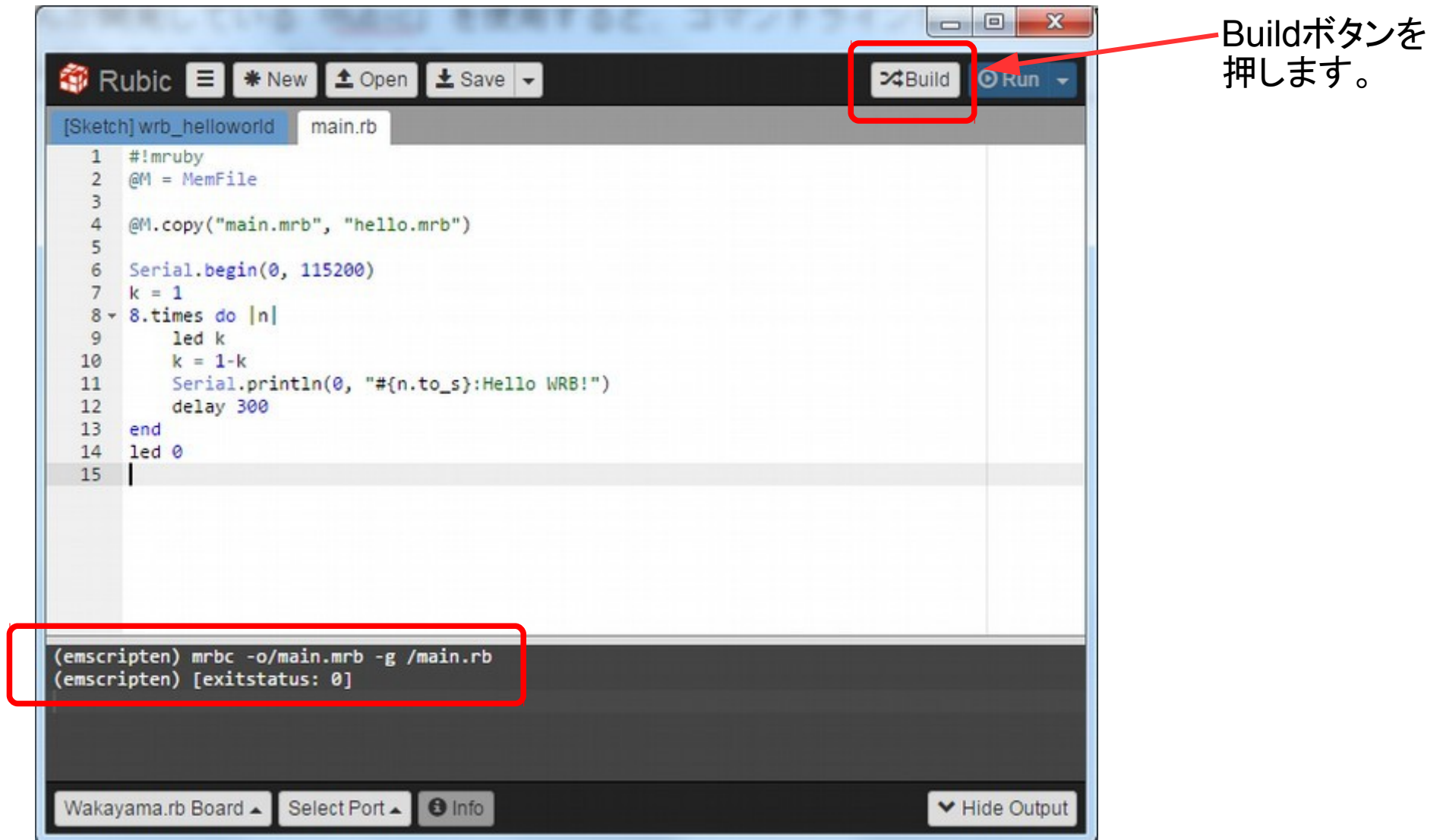
```
$ ./mrbc -g main.rb
```

```
$ ./mrbc -h
Usage: ./mrbc [switches] programfile
switches:
-c          check syntax only
-o<outfile> place the output into <outfile>
-v          print version number, then turn on verbose mode
-g          produce debugging information
-B<symbol>  binary <symbol> output in C language format
-e          generate little endian iseq data
-E          generate big endian iseq data
--verbose   run at verbose mode
--version   print the version
--copyright print the copyright
```



## mrbcファイルの作成方法

きむしゅさんが開発している「Rubic」を使用すると、コマンドラインからmrbcを使うことなくmrbcファイルを作成することができます。



# メソッドの説明 (V1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 (V1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

# メソッドの説明 (V1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。

PWM周波数設定 `pwmHz(value)`

PWM出力するときの周波数を設定します。

value: 周波数 (12~184999) Hz

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 (V1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 (V1 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    `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

# メソッドの説明 (V1ライブラリ)

## カーネルクラス

### 使用例

```
pinMode(4, 0)
pinMode(5, 1)

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

10.times do
  led(1)
  delay(1000)
  led(0)
  delay(1000)
end
```

# メソッドの説明 (V1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(V1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 (V1ライブラリ)

## システムクラス

### 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。
```



# メソッドの説明 (V1ライブラリ)

## シリアルクラス

### シリアル通信の初期化 `Serial.begin(num, bps)`

シリアル通信を初期化します。シリアル通信を私用する場合は、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

bps: ボーレート (bps) 基本的に任意の値が設定できます。

### シリアル通信のデフォルト通信番号の設定 `Serial.setDefault(num)`

シリアル通信のデフォルト通信番号を設定します。

num: 通信番号 (番号はSerial.begin参照)

※システム内部でエラーなどが発生した時に、出力されるメッセージの出力先となります。

### シリアルポートへの出力 `Serial.print(num[, str])`

シリアルポートに出力します。

num: 通信番号 (番号はSerial.begin参照)  
str: 文字列。省略時は何も出力しません設定できます。

### シリアルポートへの出力 (¥r¥n付き) `Serial.println(num[, str])`

シリアルポートに¥r¥n付きで出力します。

num: 通信番号 (番号はSerial.begin参照)  
str: 文字列。省略時は改行のみ

# メソッドの説明 (V1ライブラリ)

## シリアルクラス

### シリアル受信チェック `Serial.available(num)`

シリアルポートに受信データがあるかどうか調べます。

num: 通信番号(番号はSerial.begin参照)

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1バイト取得 `Serial.read(num)`

シリアルポートの受信データを1バイト取得します。

num: 通信番号(番号はSerial.begin参照)

戻り値

0x00~0xFFの値、データが無いときは-1が返ります。

### シリアルポートへデータ出力 `Serial.write(num, buf, len)`

シリアルポートにデータを出力します。

num: 通信番号(番号はSerial.begin参照)

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

## メソッドの説明 (V1ライブラリ)

### シリアルクラス

シリアルポートを閉じます `Serial.end(num)`

シリアルポートを閉じます。  
num: 通信番号(番号はSerial.begin参照)

# メソッドの説明 (V1ライブラリ)

## シリアルクラス

### 使用例

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

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

50.times do
  while(Serial.available(0) > 0) do #何か受信があった
    c = Serial.read(0).chr          #1文字取得
    Serial.print(0, c)              #エコーバック
    System.fileload()
  end

  #LEDを点滅させます
  led(Sw)
  Sw = 1 - Sw

  delay(500)
end
```

```
Serial.begin(1, 115200)      #0ピンと1ピンのシリアル通信初期化

data = 0x30.chr + 0x31.chr + 0.chr + 0x32.chr + 0x33.chr + 0x0d.chr + 0x0a.chr
Serial.write(1, data, 7 )    #1番ポートに7バイトのデータを出力

System.exit()
```

## メソッドの説明 (V1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 (V1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.copy(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 (V1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.copy('sample.txt', 'memfile.txt', 1)

Serial.begin(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
  Serial.write(0, c.chr, 1)
end
MemFile.close(0)
System.exit()
```

# メソッドの説明 (V1ライブラリ)

## I2cクラス

I2C通信を行うピンの初期化 `I2c.sdascI(sda, scl)`

I2C通信を行うピンを設定します。

sda: データピン  
scl: クロックピン

アドレスにデータを書き込みます `I2c.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

- 0: 成功
- 1: 送信バッファ溢れ
- 2: スレーブアドレス送信時にNACKを受信
- 3: データ送信時にNACKを受信
- 4: その他のエラー

アドレスからデータを読み込み `I2c.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

# メソッドの説明 (V1ライブラリ)

## I2cクラス

I2Cデバイスに対して送信を開始するための準備をする: `I2c.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 `I2c.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

- 0: 成功
- 1: 送信バッファ溢れ
- 2: スレーブアドレス送信時にNACKを受信
- 3: データ送信時にNACKを受信
- 4: その他のエラー

デバイスに受信シーケンスを発行しデータを読み出す `I2c.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 (V1ライブラリ)

## I2cクラス

送信バッファの末尾に数値を追加する I2c.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 I2c.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

周波数を変更する I2c.freq(Hz)

周波数を変更します。

Hz: クロックの周波数をHz単位で指定する。

有効な値は1~200000程度。基本的にソフトでやっているなので400kHzは出ない。



# メソッドの説明(V1ライブラリ)

## I2cクラス

### 使用例

```

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

#センサ接続ピンの初期化(17番SDA, 16番SCL)
I2c.sdascl( 17, 16 )
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
I2c.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)
delay(100)

#気圧を取得します -----
#Address 0x28, 0x29, 0x2A, 0x2B, 0x2C
v0 = I2c.read( @APTemp, 0x28, 0x29)
v1 = I2c.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 = I2c.read( @APTemp, 0x2B, 0x2C)
if v2 > 32767
    v2 = v2 - 65536
end
t = v2 / 480.0 + 42.5
Serial.println(0, a.to_s + ", " + t.to_s)

```

# メソッドの説明 (V1ライブラリ)

## I2cクラス

### 使用例

```
Serial.begin(0, 115200)           #USBシリアル通信の初期化
#センサ接続ピンの初期化 (17番SDA, 16番SCL)
I2c.sdasci( 17, 16 )
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
I2c.write(@APTemp, APTemp_CTRL_REG1, APTemp_SAMPLING)  # 7Hz
delay(100)

#Address 0x2B, 0x2C
I2c.begin(@APTemp)
I2c.lwrite(0x2B)
I2c.end()
I2c.request(@APTemp, 1)
datL = I2c.lread()

I2c.begin(@APTemp)
I2c.lwrite(0x2C)
I2c.end()
I2c.request(@APTemp, 1)
datH = I2c.read()
v = datL + datH * 256
if v > 32767
  v = v - 65536
end
t = v / 480.0 + 42.5
Serial.println(0, t.to_s)
```

# メソッドの説明 (V1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 (V1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

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

ch: サーボのチャネル 0~9まで指定できます

# メソッドの説明 (V1ライブラリ)

## サーボクラス

### 使用例

```
g_pos = 0
g_inc = 10

Serial.begin(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 * -1
  end
end
end
```

# メソッドの説明 (V1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.begin()`

戻り値

- 0: 起動失敗
- 1: 起動成功
- 2: RTCは既に起動していた

RTCの時計をセットします: `Rtc.setTime(Array)`

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の時計を取得します: `Rtc.getTime()`

戻り値

Year: 年  
Month: 月  
Day: 日  
Hour: 時  
Minute: 分  
Second: 秒



# メソッドの説明 (V1ライブラリ)

## リアルタイムクロッククラス

### 使用例

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

Rtc.begin #省略可能です

Rtc.setDateTime([2015, 12, 5, 17, 0, 0])

15.times do|i|
  led(i % 2)
  year, mon, da, ho, min, sec = Rtc.getDateTime()
  Serial.println(0, year.to_s + "/" + mon.to_s + "/" + da.to_s + " " + ho.to_s + ":" + min.to_s +
  ":" + sec.to_s)
  delay(500)
end
```