Raspberry Pi Mouse

Assembling and Programming

Manual

Version 2.0
Dec. 7th, 2016
RT Corporation

RT CORPORATION

Contents

1 Safety Guide 3
1.1 To use this product safely 3
2 Hardware of Raspberry Pi Mouse 5
2.1 Parts and accessories 5
2.2 Parts names 5
3 Assembling Raspberry Pi Mouse 6
4 Installing OS 8
5 Installing device drivers 14
6 Using device drivers 17
6.1 LED operation 17
6.2 Buzzer operation 17
6.3 Motor operation 18
6.4 Reading sensors 19
6.5 Reading tactile switches 19
7 Using device driver 21
7.1 Step 1 : Turning on LED 21
7.1.1 Shell 21
7.1.2 C language 23
7.1.3 Python 25
7.2 Step 2 : Making buzzer sound 26
7.2.1 Shell 26
7.2.2 C language 28
7.2.3 Python 30
7.3 Step 3 : Using tactile switches 31
7.3.1 Shell 31
7.3.2 C language 32
7.3.3 Python 35
8 Remarks 37
8.1 References 37
8.2 Copy right 37
9 Contact 37
10 Revision history 38

1 Safety Guide

1.1 To use this product safely

① Thank you for purchasing Raspberry Pi Mouse. Before start using this product, read this manual carefully and understand important safety notices.

② We recommend you to work together with an experienced person if you are a robot beginner.

③ **[CAUTION] Short circuit**
Short circuit around sensors, motors, CPU board and power circuit may not only cause damages to the robot but also may start fire. Let the cables have some latitude and do not let it fall between devices. Never use a damaged wire. Take special care to the connections using the power supply lines and the ground.

④ **[IMPORTANT] Caution when using Lithium-ion Polymer cells**
In Raspberry Pi Mouse uses Lithium-ion Polymer ("Li-Po") cells. When using Li-Po cells, use them correctly. Li-Po cells are small and light weight, and an impulse current that can be flowed is higher compared to other cells. In addition, they have little battery-memory effect and thus, they are very good for the use in robots. On the other hand, they are more expensive. They may catch fire or explode when overcharged or electrically shorted. Although, protective circuits are equipped in Li-Po cells in general, be extremely careful when handling them. Magnitude of voltage of Li-Po cells depend on the number of cells. Average voltage is 3.7V for 1 cell, 7.4V for 2 cells and 11.1V for 3 cells. Regarding the discharging, unit "C" is used for all batteries. C refers to the discharge rate to the capacity. When the number of C is 1, it means the discharge rate is the same as the battery capacity. When the number is 2 (2C), the discharge rate is 2 times greater than the battery capacity and thus, 3C means 3 times greater discharge rate than the battery capacity.
Use dedicated charger for Li-Po cells when charging them. (We recommend you to charge Li-Po cell in 1C). When storing them, it is said that they should be 90 to 100% charged. When the cell is discharged and the voltage becomes less than 3.3V for 1 cell, it is over-discharged and the cell cannot be used. As raspberry Pi Mouse needs 3 cells, never let them discharge under 10 V. For your reference, when the cells are full and the power of the robot is on, the voltage drops to around 10 V in 20 minutes. **An alarm buzzer goes off when the voltage of the cells drops to less than 10 V. When the buzzer goes off, immediately turn off the power.** Follow the precautions when using Li-Po cells and be careful not to overcharge or over discharge them.
Should you get hurt or start fire when you falsely use them, the manufacturer nor the shops have no responsibilities.

Recharging
Use dedicated charger. Do not charge the cells close to flammables and never leave them alone. After charging, disconnect the connector of the cells from the charger. As the Li-Po cells have little memory effect, charge them frequently. Make sure to charge before the cells become completely empty.

<If the cells start swelling during charging>
If the cells start swelling or you notice abnormal smells during charging, immediately stop charging and disconnect the charger. (If the cells are kept charged in such conditions, they may explode.) After disconnecting the charger, remove the cells from the charger and put them in the safe place farther from flammables. Leave them there about one day. Never use such cells again but dispose them. (When disposing, refer to "How to dispose cells".)

Discharging
When the Li-Po cells used in Raspberry Pi Mouse are over discharged, the cells cannot be used again. Be careful not to over discharge them. **Remove the connector** after using Raspberry Pi Mouse.

When using Li-Po cells
Read this manual carefully and correctly use them. The cells may start fire or explode if it is electrically shorted, receives impacts, or nailed. For example, never put the cells into the tool box with other sharp tools, do not touch the connector with sweat hand, do not drop them in water accidentally. Never use cells in the situation that can be thought to cause damages, fire or explosion.
Li-Po cells have a certain current rate. When they are used with Raspberry Pi Mouse, the cells never exceed the maximum usable current as it is designed so. However, if you change any components of Raspberry Pi Mouse to your own or if you design a circuit not described in the manual, never let the cells exceed discharge capacitance. If the current more than the rating flows, cells may explode.

How to handle cells while not used for a long time
Charge enough and store them in the place farther from the conductors and flammables.

How to dispose cells
Immerse cells in salt water whose concentration is about the same as the sea water to completely discharge them and then dispose them as non-burnable rubbish. Exact concentration of the salt water is 30 g of salt put in 1 L water.

Warranty using Li-Po cells
It is user's responsibility to use the Li-Po cells safely. Manufacturer nor stores do not bear responsibilities for bodily injury or property accident, damage, or breakage. They are easy to use if you use them safely. Use them with correct knowledge.

2 Hardware of Raspberry Pi Mouse

This section introduces the contents and component names of Raspberry Pi Mouse.

2.1 Parts and accessories

Raspberry Pi Mouse is delivered after it is completely assembled. Components are a connector that connects Raspberry Pi and the mouse body, Raspberry Pi (only in the complete kit), and a microSD card that a Debian type Raspbian OS is installed (only in the complete kit).

2.2 Parts names

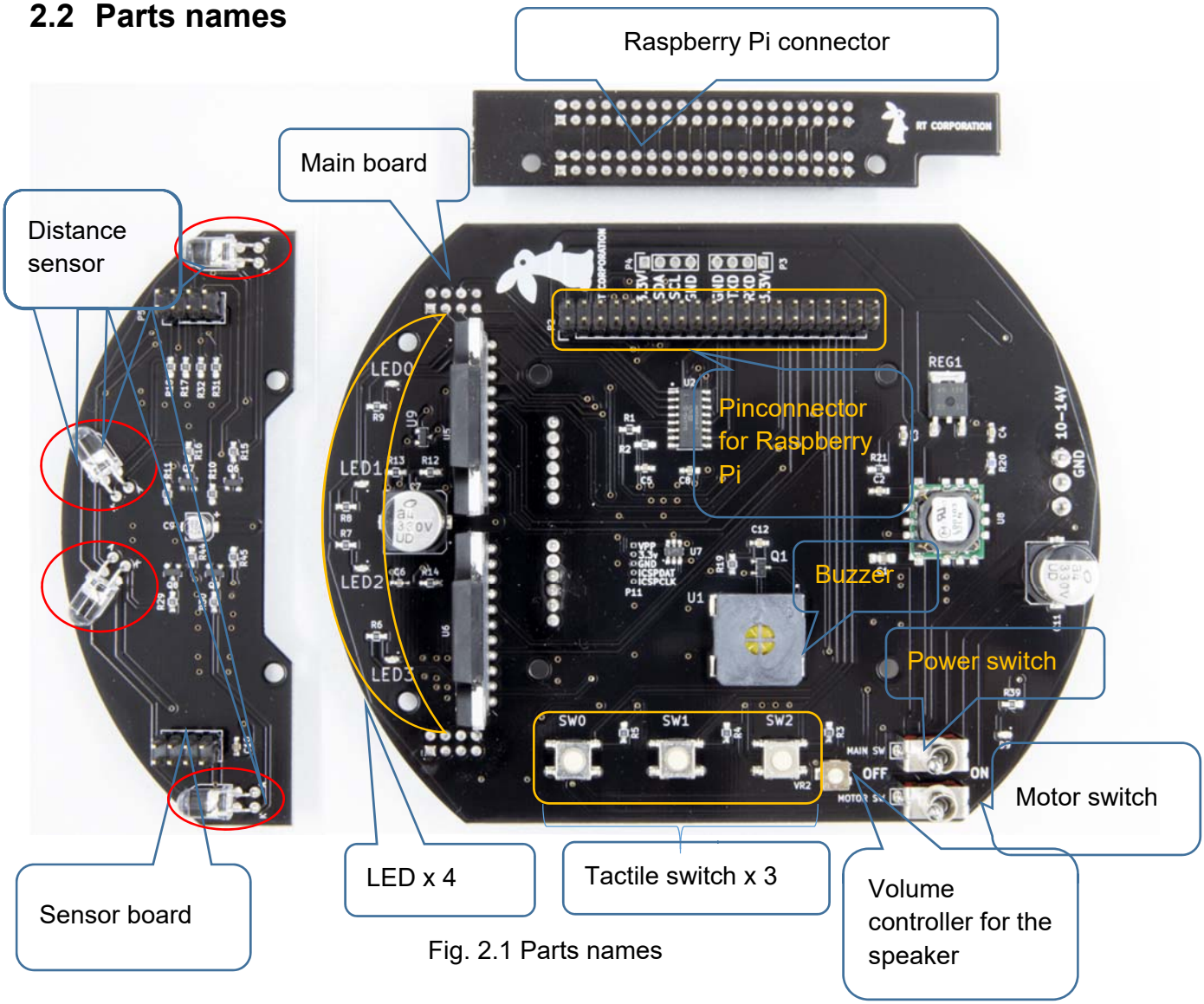
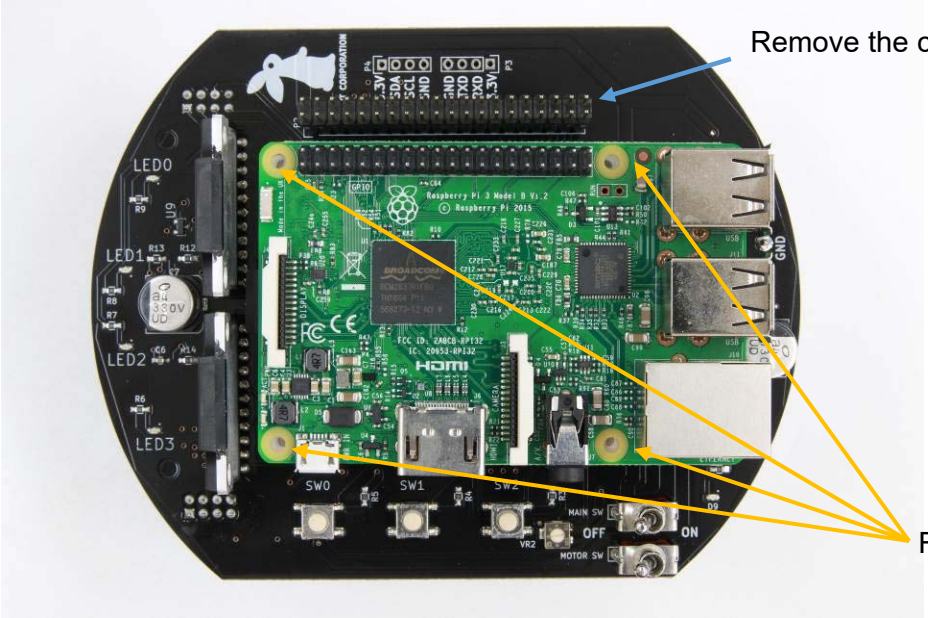


Fig. 2.1 Parts names

3 Assembling Raspberry Pi Mouse

Insert the microSD with on OS, in the direction as shown below. In Raspberry Pi3, SD card is not lockable.

Remove screws to fix Raspberry Pi and the connector and place the Raspberry Pi on a spacer.



Screw Raspberry Pi with screws removed.

Connect Raspberry Pi and the Raspberry Pi Mouse using the connector. Pay attention not to miss pins when connecting.

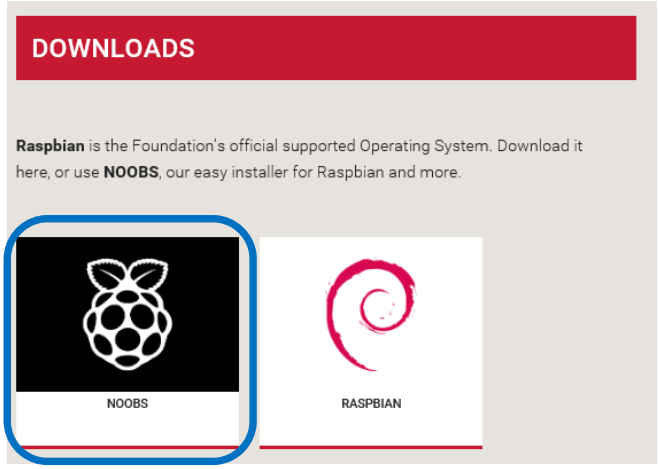
RT CORPORATION


LED0
LED1
LED2
LED3
SW0
SW1
SW2
MAIN SW
MOTOR SW
OFF
ON
RT CORPORATION
GND

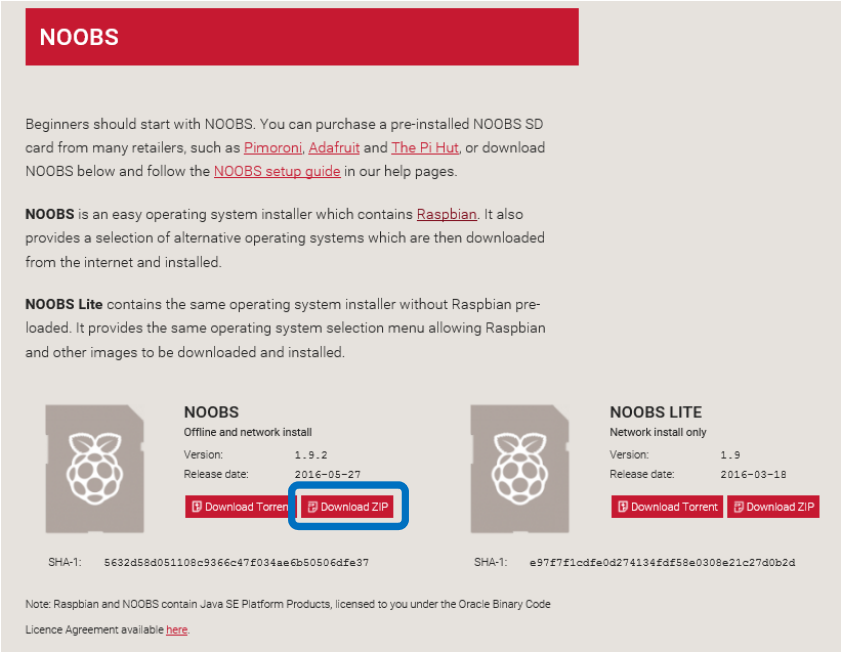
4 Installing OS

Raspberry Pi runs safely on various OSs such as Raspbian, Ubuntu (ROS and RTM). In this section, how to install Raspbian, the Raspberry Pi's official OS is explained. Download Raspbian OS from the URL below.
https://www.raspberrypi.org/downloads/
Save the downloaded file to the SD card.

Following procedures are for Windows OS.

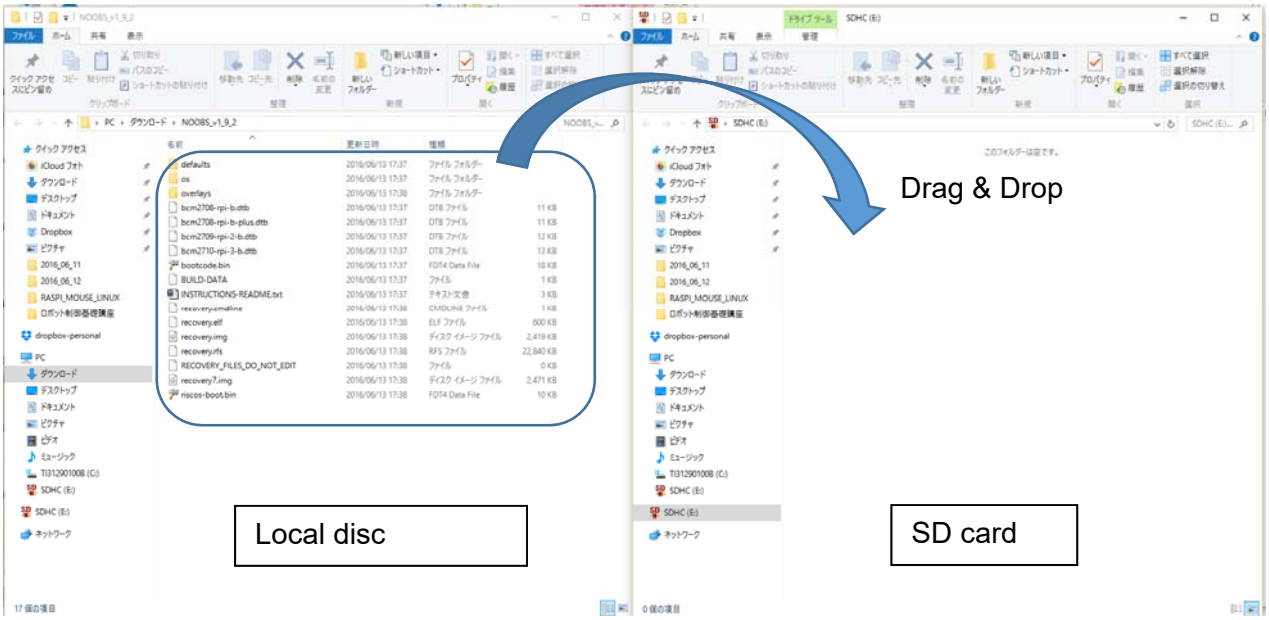


There are two files (NOOBS and RASPBIAN) in the download site (above). RASPBIAN is an OS image file. To download RASPBIAN and save it to SD card, you need software to write an image. NOOBS doesn't need any software to download it and save it to SD card. If you are a beginner, downloading NOOBS is recommended. the following describes how to install NOOBS. Click NOOBS in the screen above and the below screen appears.

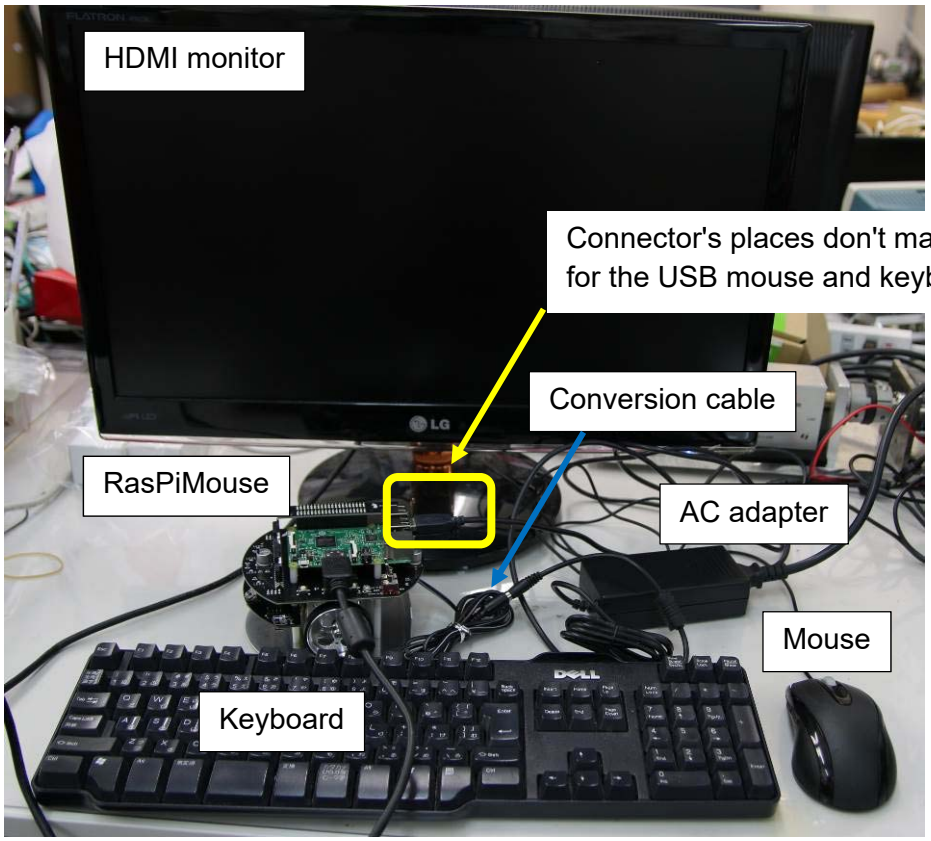


Click Download.ZIP in NOOBS (not NOOBS LITE) and save the file to your local disc.

Unzip the file into the SD card. (Right click the file and select "Extract All" to unzip if your Windows OS is Windows 7 or later.) 16 GB or more SD card is recommended. You need at least 8 GB to arrange an environment to compile device drivers on Raspberry Pi.



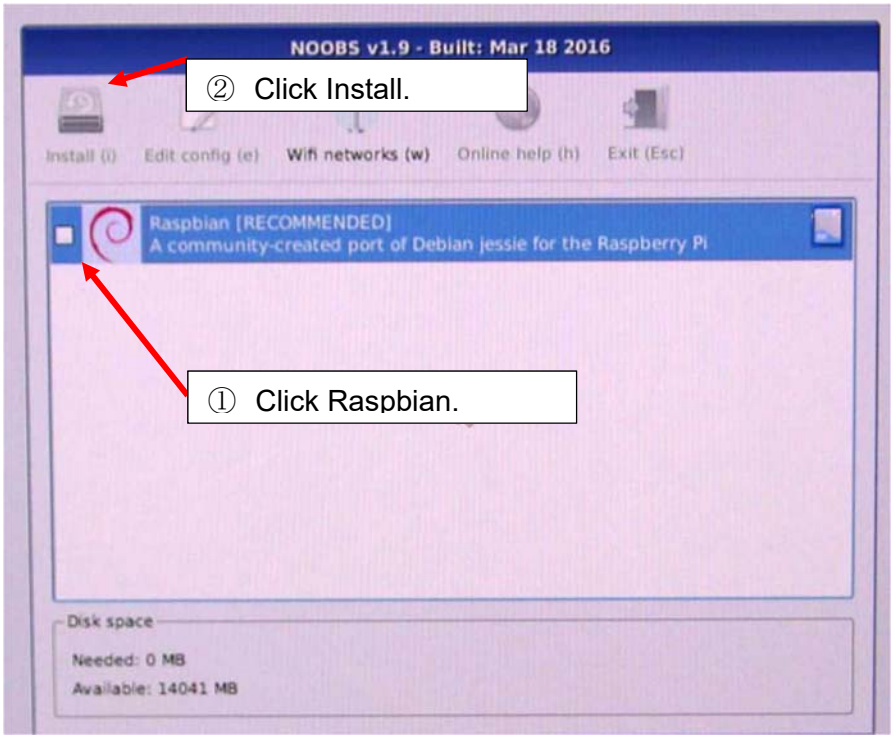
After copying all the basic data of the OS to SD card, assemble Raspberry Pi Mouse referring to Chapter 3. After assembling, connect the USB mouse, USB keyboard and HDMI monitor to Raspberry Pi.



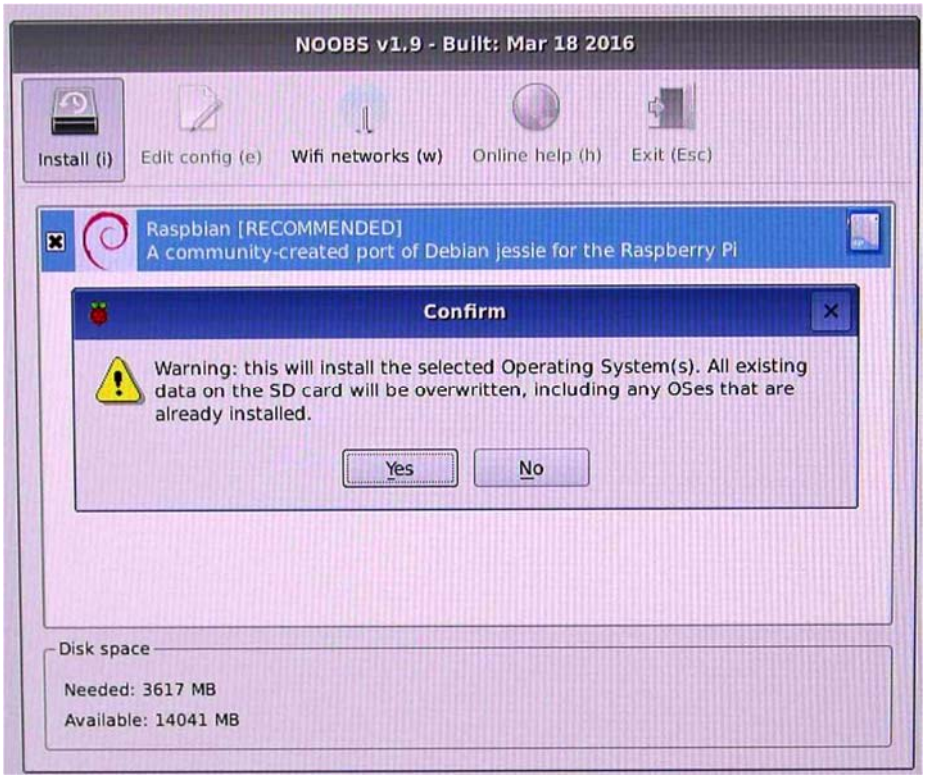
In the connection example above, an AC adapter is used instead of batteries. (The AC adapter is included in the complete kit. If you want to purchase an AC adapter separately, it is available in our webshop. Li-Po charger LBC-010 [AC adapter and conversion cable are included] 6,500yen (excluding tax)]

If you want to use batteries instead of an AC adapter, fully charge them before setting up the initial environment. After connecting the mouse, keyboard and HDMI monitor, turn the main switch ON. Turn the motor switch OFF. If it is turned on before the OS is started running, current flows on the motor. If you are using batteries and you turn the motor switch on during setting up the environment, the batteries voltages may drop below 10V before the setting up is completed so make sure to turn off the motor switch.

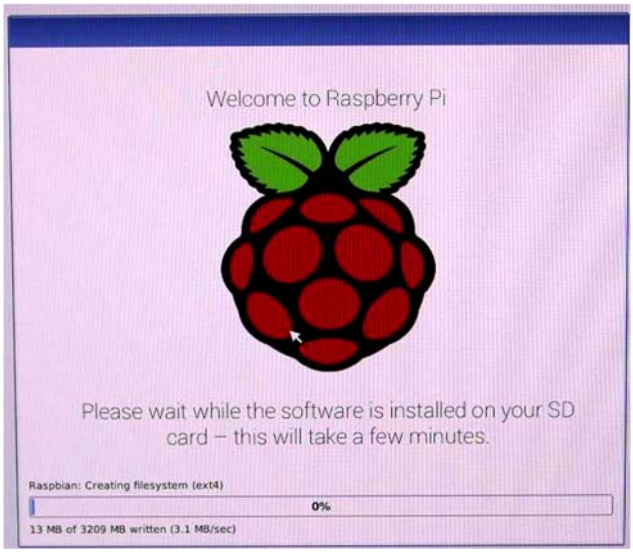
As you will have to download device drivers from github, connect LAN cable or set up Wi-Fi.



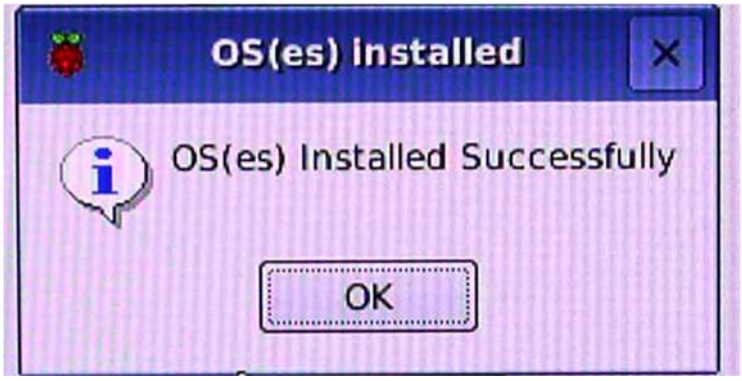
When you turn on the main switch, the screen above appears after a while. Click Raspbian and then click Install button. Install button becomes clickable after you click Raspbian.



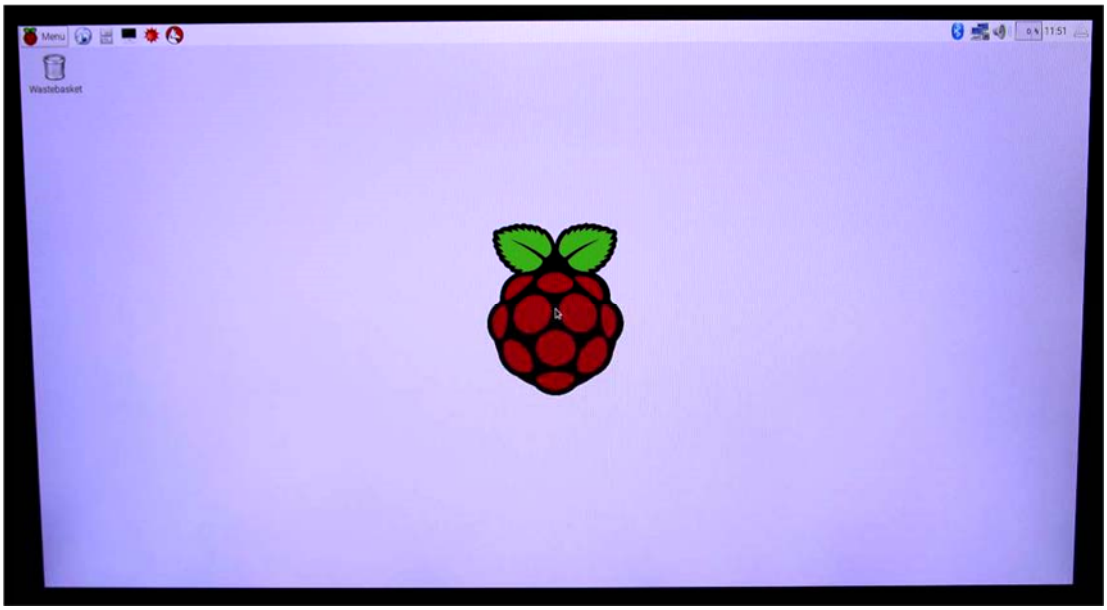
After clicking Install button, a confirmation screen appears. Click "Yes".



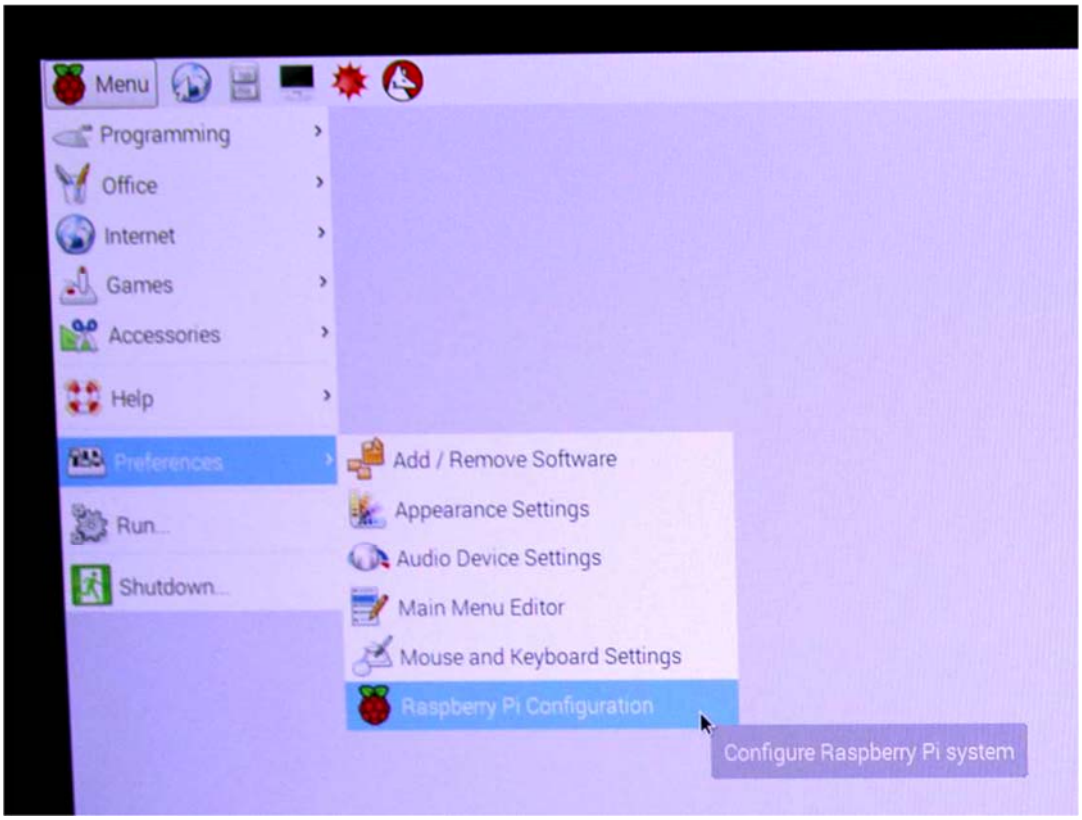
After clicking "Yes", installation starts. Wait for a while. Installation takes place for about 15 minutes if you are using Raspberry Pi3.

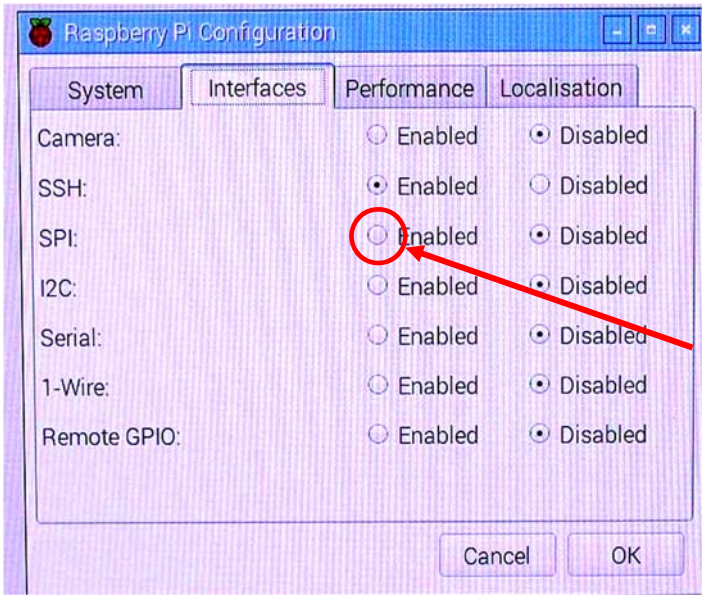


After installation is completed, the screen above appears. Click "OK". The OS reboots automatically.



Then GUI starts up. Password and user information are not asked.
Next, enable SPI for Raspberry Pi Mouse to be able to use it.
Click Raspberry Pi Configuration in Preferences in the Menu.



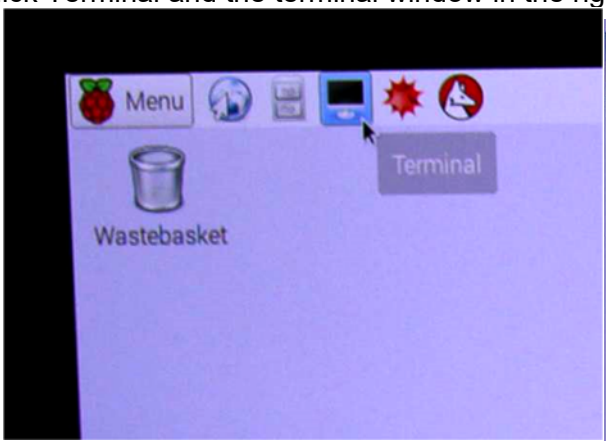


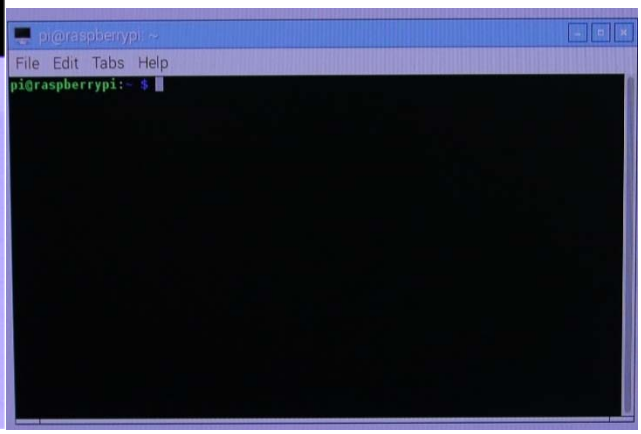
Click

Raspberry Pi Configuration starts. Select the Interfaces tab. Select SPI enabled and then click "OK" button.
All above environmental setup has been done in the SD card with OS. If you don't want to work for environmental setup, we recommend you our Raspberry Pi Mouse complete kit.

5 Installing device drivers

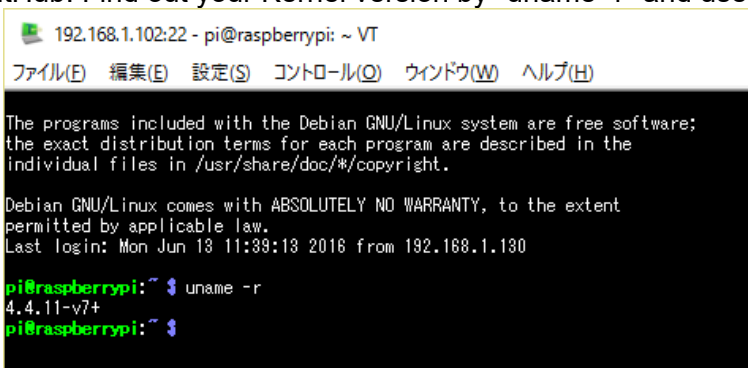
Download device drivers from "GitHub" using Git *1. To use Git, you have to input commands in the terminal window. To start the terminal window from GUI, follow the procedure below.
Click Terminal and the terminal window in the right appears.





You can also install device drivers by logging in with SSH. To login with SSH, initial user name, "pi" and a password, "raspberry" are needed.

Device drivers are in "./lib/RaspberryPi version/Kernel version" in the file downloaded from GitHub. Find out your Kernel version by "uname -r" and use the kernel of the right version.



Next perform Git clone. (Delete RaspberryPiMouse directory if it exists.)

```
git clone https://github.com/rt-net/RaspberryPiMouse.git
```

If Git asks a password, input "raspberry". If you switch your authority to root using "sudo su -" command, Git will not ask for a password.

After Git clonine is performed, the file construction is as follows.

```
RaspberryPiMouse
 ├lib
 |  ├Pi1B+
 |  |  └3.18.14+ etc.
```

*1 https:github.com/rt-net/RaspberryPiMouse.git

```
 |  |      └rtmouse.ko
 |  └Pi2B+
 |      └4.4.13-v7+ etc.
 |          └rtmouse.ko
 ├LICENSE
 ├README.md
 ├src
 |  └drivers
 |      ├Makefile
 |      ├Makefile.rasbian
 |      ├Makefile.ubuntu14
 |      └rtmouse.c
 ├supplement
 |  ├201507
 |  |  ├Buzzer_circuit.jpg
 |  |  ├Drive_circuit4.jpg
 |  |  └ハードの簡単な説明.rst
 |  ├201508
 |  |  └RPiMouse-schematic.pdf
 |  └201509
 |      └MEMO.raspi-config.rst
 └utils
     ├build_install.raspbian.bash
     └build_install.ubuntu14.bash
```

RaspberryPiMouse/lib/RaspberryPi version/Kernel version/rtmouse.ko is the device driver. This file includes a series of programs operate as part of kernel and called "Kernel module". Enable SPI function as it is used by sensors. Refer to Chapter 4.

Device drivers whose kernel version is later than 4.1.19 can be used both in Raspberry Pi2 and RaspberryPi3. To implement a device driver in Raspberry Pi3, use Pi2+ device driver.

Use "insmod" to implement device driver. Type "lsmod" command to list all the implemented kermel modules. Confirm "rtmouse" exists in the list. After the device driver is successfully implemented, a buzzer goes out and an LED lights.

```
Install
$sudo insmod rtmouse.ko
If no message appears, installation is successful.

or
$sudo insmod /home/pi/RaspberryPiMouse/lib/Pi?B+/`uname -r`/rtmouse.ko

To check the installment is completed,
$lsmod | grep  rtmouse
rtmouse 12664 0
```

Raspberry Pi version

Kernel version

When the installation is successful, a file name rt<something> is found in the /dev/ directory as follows. Files under /dev/ directory are called "device file". They control the machine and it is input and output.

```
$ls /dev/rt*
/dev/rtbuzzer0    /dev/rmotor_raw_r0
/dev/rtled0       /dev/rtmotoren0
(omitted)
```

If you change the permission, you can handle device drivers without changing the authority to "root". For your reference, use "rmmod" command to uninstall the driver.

```
$sudo chmod 666 /dev/rt*
$ls -l /dev/rt*
crw-rw-rw- 1 root root 245, 0  4 月 27 14:38 /dev/rtbuzzer0
crw-rw-rw- 1 root root 246, 0  4 月 27 14:38 /dev/rtled0
crw-rw-rw- 1 root root 246, 1  4 月 27 14:38 /dev/rtled0
(omitted)
```

6 Using device drivers

Device drivers turn LED on/off , turn on/off buzzer, acquire sensor values of the illumination reflecting type distance sensor, turn motor on/off, and confirm on/off of switch.

6.1 LED operation

There are three LEDs that are from LED0 to LED3. Commands below are an example statement to turn LED0 on/off.
If you want to use LED1, change utled0 to rtled1. Change the number at the last of the command from 0 to 3 to change the LED to turn on/off.

```
Turn on
        $ echo 1 > /dev/rtled0
Turn off
        $echo 0 > /dev/rtled0
```

Fig. 6-1 LED operation

6.2 Buzzer operation

Use "echo" command to specify frequencies (in Hz).

```
Sound the buzzer at 440Hz
        $ echo 440 > /dev/rtbuzzer0

Turn off buzzer
        $ echo 0 > /dev/rtbuzzer0
```

Fig. 6-2 Buzzer operation

For your reference, frequencies of musical scales are listed in Table 1.
Decimal points are omitted (only 0 and positive integers are available). Round up or down each scale by yourself.

Table 1 Frequencies for musical scale.

Octave	Scale											
	C(Do)	C#	D(Re)	D#	E(Mi)	F(Fa)	F#	G(So)	G#	A(La)	A#	B(Ti)
0	16.35	17.32	18.35	19.45	20.60	21.83	23.12	24.50	25.96	27.50	29.14	30.87
1	32.70	34.65	36.71	38.89	41.20	43.65	46.25	49.00	51.91	55.00	58.27	61.74
2	65.41	69.30	73.42	77.78	82.41	87.31	92.50	98.00	103.8	111.0	116.5	123.5
3	130.8	138.6	146.8	155.6	164.8	174.6	185.0	196.0	207.7	220.0	233.1	246.9
4	261.6	277.2	293.7	311.1	329.6	349.2	370.0	392.0	415.3	440.0	466.2	493.9
5	523.3	554.4	587.3	622.3	659.3	698.5	740.0	784.0	830.6	880.0	932.3	987.8
6	1047	1109	1175	1245	1319	1397	1480	1568	1661	1760	1865	1976
7	2093	2217	2349	2489	2637	2794	2960	3136	3322	3520	3729	3951
8	4186	4435	4699	4978	5274	5588	5920	6272	6645	7040	7459	7902
9	8372	8870	9397	9956	10548	11175	11840	12544	13290	14080	14917	15804

Ref to step 6.

6.3 Motor operation

A motor runs with both the hardware switch (MOTOR SW) and the software switch (/dev/rtmotoren0) on. The hardware switch is equipped for preventing Raspberry Pi from running out of control. Even though the hardware switch is not necessary if there is no programming errors for the software switch, a hardware switch is needed to forcibly halt the operation of Raspberry Pi.

```
Turning ON software switch
$ echo 1 > /dev/rtmoteren0□

Turning only the left motor to forward direction at 400Hz
$ echo 400 > /dev/rtmotor_raw_l0□

Turning the motor to backward direction at 400Hz
$ echo -400 > /dev/rtmotor_raw_l0□

Turning only the right motor to forward direction at 250Hz
$ echo 250 > /dev/rtmotor_raw_r0□

Halting the motor
$echo 0 > /dev/rtmotor_raw_r0□
$echo 0 > /dev/rtmotor_raw_l0□
```

Fig. 6-3 Motor operation (1/2)

Turning the motor right and left at the specified frequency for the specified time.
Turning the left motor at 500Hz and the right motor at 10Hz for 1 second. The unit of the time is "ms".

```
          Left     Right      Time
$echo 500 10 1000 > /dev/rtmotor0

Turning OFF software switch
$ echo 0 > /dev/rtmoteren0
```

Fig. 6-3 Motor operation (2/2)

Only integers can be used.
The motor used in Raspberry Pi Mouse is a stepping motor that turns 0.9 [deg] per 1 pulse and turns once around with 400 pulses. The number after "echo" comes frequencies. If the number is 400, 400 pulses are input per 1 second and the motor turns around once in 1 second.

6.4 Reading sensors

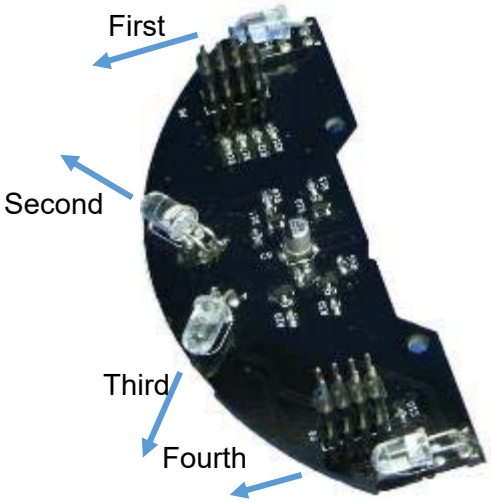


Fig. 6-4 Four visible LEDs on the Raspberry Pi Mouse board

```
$ cat /dev/rtlightsensor0
187 189 184 191
```

The values read from the sensors are shown in the order from the first to the fourth.

6.5 Reading tactile switches

Three tactile switches are attached and they are numbered from 0 to 2. Example below is to see SWO if it is pushed or released.
The number after "rtswitch" specifies the tactile switch. If you want to see the condition of tactile switch 1, the command is "rtswitch1".

```
When switch is being pushed.
$ cat  /dev/rtswitch0□
0

When the switch is released;
$cat  /dev/rtswitch0□
1
```

Fig. 6-5 Switch operations

7 Using device driver

Chapter 6 briefly described how to use device drivers. This section demonstrates detailed programming using Shell, C language, and python and explains the program in detail.

7.1 Step 1 : Turning on LED

Step1 is a program to turn the LED on. The following program makds the LED blink every 0.5 second.

7.1.1 Shell

```
#!/bin/bash
while true
do
        echo 1 | tee /dev/rtled?        #turn on
        sleep 0.5                       #waiting for 0.5 second
        echo 0 | tee /dev/rtled?        #turn off
        sleep 0.5                       #waiting for 0.5 second
done
```

Fig. 7-1 turning on LED (Shell)

Operation check and description(How to use this sample program)

Save the program above in the file "step1.sh". Add an authorization before the program name (chmod + x step1.sh) and execute the command "$ ./step1.sh". LED blinks. To finish the program, execute "Ctrl + C ⏎".

Use "while" to repeat the command infinitely.

while statement

```
while conditional expression
do
        process
done
```

Fig. 7-2 while statement

While the condition is true, command from "do" to "done" repeatedly executes. NULL command ":" can be used instead of "true" for this infinite loop program.

"echo" is used to send data to a device driver. "echo" command displays characters or variables specified as arguments.

```
echo 1 > /dev/rtled0
```

Above command sends "1" to the device driver LED0 and the LED lights. When "0" is sent, the LED turns off. ">" is called redirection. The redirection newly writes the data at the left side of it to the file (device driver) specified at the right side of it. Additional writing is available with ">>". In the above example program, ">" is not used but instead "tee" command and the wild card "?" are used to write LED 0 to LED 3. "tee" is used with the pipe "|" that links commands. With the pipe, the command executes from left to right and the result of the command execution is transferred to the next command. In the example, "1" in "echo 1" is transferred to "tee /dev/rtled?". "tee" displays the result of the command execution while transferring the result to the file (in this example, device driver).

"sleep" halts the program for the specified time. The unit of the time without unit is "second".

7.1.2 C language

```
#include "fcntl.h"

void main(void)
{
int led[4];
int i;


led[0] = open("/dev/rtled0",O_WRONLY);
led[1] = open("/dev/rtled1",O_WRONLY);
led[2] = open("/dev/rtled2",O_WRONLY);
led[3] = open("/dev/rtled3",O_WRONLY);

while(1)
{
        for(i=0;i<4;i++)
        {
                write(led[i],"1",1);
        }
        usleep(500*1000);
        for(i=0;i<4;i++)
        {
                write(led[i],"0",1);
        }
        usleep(500*1000);
}
        for(i=0;i<4;i++)
        {
                close(led[i]);
        }

}
```

Fig. 7-3 turning on LED (C language)

Operation check and description(How to use this sample program)

Save the program above in the file "step1.c" and compile using "$gcc step1.c –o step1". When "-0" is not specified, it automatically becomes "a.out". Execute the compiled file with "$ ./step1". LED blinks. To finish the program, execute "Ctrl + C ⏎".

For writing and reading files with C language, "open" and "close" functions are used. Device drivers can be used regardless of file access. To enable "open" and "close" functions, include "fcntl.h".

Use "while" to repeat the statement infinitely.

while statement

```
while (conditional expression)
{
        processing
}
```

Fig. 7-4 while statement

While the condition is true, processes in { } repeatedly execute. The same kind of command "do { } while (conditional expression);" executes at least once even though the condition expression is false. However, the command inside the brace notations "{ }" in "while (conditional expression) { }" never executes when the conditional expression is false. But with "do { } while(false);" statement, the LED blinks once.

```
while(0){
    write (led0,"1",1);
    usleep(500 * 1000);
    write (led0,"0",1);
    usleep(500 * 1000);
}
```

Fig. 7-5 LED0 never blinks

```
do
    write (led0,"1",1);
    usleep(500 * 1000);
    write (led0,"0",1);
    usleep(500 * 1000);
} while(0)
```

Fig. 7-6 LED0 blinks once.

"write" command is used to output data to a device driver.
The following is the "write" statement.

```
int write(int fd, void *buf, unsigned int n);
```

Fig. 7-7 write statement

int fd: specifies a file to write data In this example, the file is a device driver.
void *buf: Specifies an address to write data In this example, data is directly written.
unsigned n: Specifies the size of data. In this example, 1 is specified as only 1 character is output.

To write data to the device driver, "fwrite" command also can be used; however, data is sent through the inner buffer with "fwrite", data is not output until the buffer is full or it is closed.

"sleep" function of the C language cannot specify decimals. "usleep" function for specifying micro second (μs) is used to specify 0.5 second instead. Units of the time are for 10^{-3} is "ms" and 10^{-6} is "μs". To convert "μs" to "ms", the number must be multiplied by 1000. As 0.5 second = 500ms, 500 x 1000 is 0.5 second.

7.1.3 Python

```
#!/usr/bin/python
import time
import sys

files = [ "/dev/rtled0","/dev/rtled1","/dev/rtled2","/dev/rtled3"]

while 1:
	for filename in files:
		with open(filename,'w') as f:
			f.write("1")


	time.sleep(0.5)

	for filename in files:
		with open(filename,'w') as f:
			f.write("0")

	time.sleep(0.5)
```

Fig. 7-8 turning on LED (python)

Operation check and description(How to use this sample program)
Save the program above in the file "step1.py" and execute the command "$python step1.py". LED blinks. To finish the program, execute "Ctrl + C ⏎".
"write" is used to write data to a device driver. When "write" command is used, data is once transferred to a buffer and the data will not be output to the device driver until the buffer is full. However, if "with" is used in the program with "write" command, the buffer will be automatically closed when the data is transferred and the data will be output as soon as the buffer is closed.
"sleep" function is included in the "time" so the time is imported.
With C language, "{ }" is used in the "while" statement. With Python, indenting is used instead of "{ }".

7.2 Step 2 : Making buzzer sound

When the device driver is used, the buzzer goes off when a frequency of the buzzer is sent to the device driver. Using a keyboard, musical scale is output with the program below.
Keys on the keyboard that correspond to each music note are as follows.

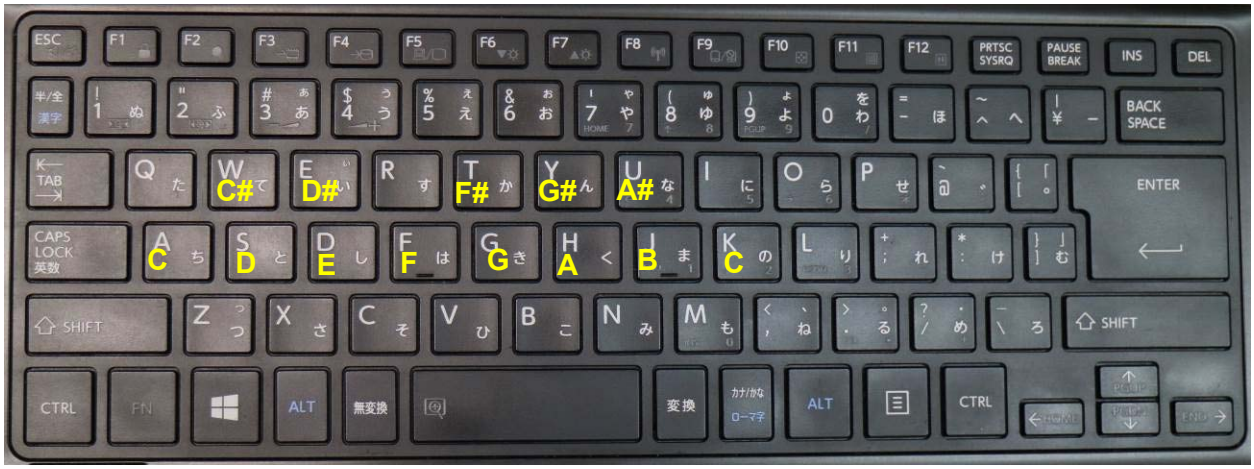


Fig. 7-9 Musical scale

7.2.1 Shell

step2.sh

```
#!/bin/bash
while read -N 1 b ; do
        awk -v "t=$b" '$3==t{print $2}' SCALE > /dev/rtbuzzer0
done
```

Fig. 7-10 making buzzer sound (shell 1/2)

SCALE file

```
off 0 0
C 261 a
C# 277 w
D 293 s
D# 311 e
E 329 d
F 349 f
F# 370 t
G 392 g
G# 415 y
A 440 h
A# 466 u
B 493 j
C 523 k
```

Fig. 7-11 Making buzzer sound (Shell 2/2)

Operation check and description(How to use this sample program)
Save above two files in the files"step2.sh" and "SCALE". Add an authorization and execute "$ ./step2.sh". The musical scale can be heard with pressing keys.
As shell cannot use the associative array, it is made by the program using "awk" command. "read –N 1 b" after the "while" command means that read one character from the key and transforms it to a variable b. "-N numbers" option specifies the number of characters to transform into variables. Without this option, characters keep being transformed into variables until the return key is pressed.

7.2.2 C language

```
#include "fcntl.h"

int _Getch(void)
{
Int ch;
        system("stty -echo -icanon min 1 time 0");
        ch = getchar();
        system("stty echo icanon");
        return ch;
}

void main(void){
int buzzer= open("/dev/rtbuzzer0" , O_WRONLY  );
int c=1;

while(c){
      switch(_Getch()){
            case '0'://off
                  write (buzzer,"0",2);
                  break;
            case 'a'://do
                  write(buzzer,"261",4);
                  break;
            case 'w'://do#
                  write(buzzer,"277",4);
                  break;
            case 's'://re
                  write(buzzer,"293",4);
                  break;
            case 'e'://re#
                  write(buzzer,"311",4);
                  break;
            case 'd'://mi
                  write(buzzer,"329",4);
                  break;
            case 'f'://fa
                  write(buzzer,"349",4);
                  break;
```

Fig. 7-12 making buzzer sound (C language 1/2)

```
                case 't'://fa#
                        write(buzzer,"370",4);
                        break;
                case 'g'://so
                        write(buzzer,"392",4);
                        break;
                case 'y'://so#
                        write(buzzer,"415",4);
                        break;
                case 'h'://ra
                        write(buzzer,"440",4);
                        break;
                case 'u'://ra#
                        write(buzzer,"446",4);
                        break;
                case 'j'://shi
                        write(buzzer,"493",4);
                        break;
                case 'k'://DO
                        write(buzzer,"523",4);
                        break;
                case ‘c’:// program ends
                        write(buzzer,”0”,2);
                        c=0;
                        break;
        }
}

close(buzzer);

}
```

Fig. 7-12 making buzzer sound (C language 2/2)

Operation check and description(How to use this sample program)
Save the program above in the file "step2.c" and compile using "$gcc step2.c –o step2".
Execute the command "$ step2" and press any key to make the buzzer to sound.
As the "getchar" command updates the input data to a variable when a return key is pressed, the buzzer cannot sound only with pressing keys. Therefore, "system" function is used to set not to echoback on Linux (-echo), invalidate special characters (-icanon), time out when the minimum number of character cannot be read with "-canon" command (time N), and set the minimum number of character N when "-canon" is set (min N). With these commands, characters are transformed to variables one by one immediately after it is entered.

7.2.3 Python

```
#!/usr/bin/python
import sys

class _Getch:
    def __init__(self):
        import tty, sys

    def __call__(self):
        import sys, tty, termios
        fd = sys.stdin.fileno()
        old_settings = termios.tcgetattr(fd)
        try:
            tty.setraw(sys.stdin.fileno())
            ch = sys.stdin.read(1)
        finally:
            termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
        return ch

dict= { "0":"0", "a":"261", "w":"277", "s":"293","e":"311","d":"329","f":"349","t":"370",
"g":"392","y":"415","h":"440","u":"446","j":"493","k":"523"}

while 1:
        getch = _Getch()
        d = getch()
        if d == "c" :
                break
        if d in dict :
                with open('/dev/rtbuzzer0','w') as f:
                        f.write(dict[d])
```

Fig. 7-13 making buzzer sound (Python)

Operation check and description(How to use this sample program)
Save the program above in the file "step2.py" and execute the command "$python step2.py". Press any key to make buzzer to sound. To finish the program, enter "Ctrl + C ⏎" or "c".
Python has an associative array (dictionary) and the program is not complicated. "raw_input( )" command of python receives key entries from the keyboard; however, data cannot be acquired without pressing the return key after the keyboard entry. To overcome this problem, "_Getch" function is created and with this function, data will be acquired soon after the keyboard entry.

7.3 Step 3 : Using tactile switches

The following is a program that LEDs turn on and off using tactile switches. By pressing switch 0 (SW0), LED 3 keeps turning on and off, pressing switch 1 (SW1), LED 1 and 2 keep turning on and off, and pressing switch 2 (SW2), LED 0 keeps turning on and off.

7.3.1 Shell

```
#!/bin/bash

state0=0
state1=0
state2=0

while true ; do
        if grep -q 0 /dev/rtswitch0 ; then
                sleep 0.1
                while grep -q 0 /dev/rtswitch0 ; do
                        sleep 0.1
                done
                state0=`expr $state0 + 1`
                state0=`expr $state0 % 2`
                echo $state0 > /dev/rtled3
        fi
        if grep -q 0 /dev/rtswitch1 ; then
                sleep 0.1
                while grep -q 0 /dev/rtswitch1 ; do
                        sleep 0.1
                done
                state1=`expr $state1 + 1`
                state1=`expr $state1 %2`
                echo $state1 > /dev/rtled2
                echo $state1 > /dev/rtled1
        fi
```

Fig. 7-14 using tactile switches (shell) (1/2)

```
        if grep -q 0 /dev/rtswitch2 ; then
                sleep 0.1
                while grep -q 0 /dev/rtswitch2 ; do
                        sleep 0.1
                done
                state2=`expr $state2 + 1`
                state2=`expr $state2 % 2`
                echo $state2 > /dev/rtled0
        fi
done
```

Fig. 7-14 using tactile switches (shell) (2/2)

Operation check and description (How to use this sample program)
Save above file in the file "step3.sh". Add an authorization and execute "$ ./step3.sh". LEDs turns on when tactile switches (SW0 to SW2) are pressed. When a tactile switch is pressed, "0" is input. When it is released, "1" is input. After "0" is input with "if grep –q 0 /dev/rtswitch??" command, the switch waits for a second with "sleep 0.1" . This waiting time is the time that the metal part of the mechanical switch vibrates with it pressed/ released. This is called chattering. While the switch chattering, the software cannot receive right information as it sends both "1" and "0". However, as chattering fades out after a few μs to a few ms, use "sleep 0.1" command to wait for a second.

7.3.2 C language

```
#include "fcntl.h"

char get_SW0(void){
        char buf[2];
        int SW0;
        SW0 = open("/dev/rtswitch0",O_RDONLY);
        read(SW0,buf,2);
        close(SW0);
        return buf[0];
}

char get_SW1(void){
        char buf[2];
        int SW1;
```

Fig. 7-15 using tactile switches (C language) (1/3)

```
    SW1 = open("/dev/rtswitch1",O_RDONLY);
        read(SW1,buf,2);

        close(SW1);
        return buf[0];
}

char get_SW2(void){
        char buf[2];
        int SW2;
        SW2 = open("/dev/rtswitch2",O_RDONLY);
        read(SW2,buf,2);
        close(SW2);
        return buf[0];
}

void main(void){
        int state0,state1,state2;
        int LED0,LED1,LED2,LED3;

        LED0 = open("/dev/rtled0",O_WRONLY);
        LED1 = open("/dev/rtled1",O_WRONLY);
        LED2 = open("/dev/rtled2",O_WRONLY);
        LED3 = open("/dev/rtled3",O_WRONLY);

        state0 = state1 = state2 = 0;

        while(1){
                if (get_SW0() =='0'){
                        usleep(10000);
                        while (get_SW0() == '0');
                        usleep(10000);
                        state0=(state0+1)&0x01;
                        if(state0==0){
                                write(LED3,"0",1);
                        }else{
                                write(LED3,"1",1);
                        }
                }
```

Fig. 7-15 using tactile switches (C language) (2/3)

```
                if (get_SW1() =='0'){
                        usleep(10000);
                        while (get_SW1() == '0');
                        usleep(10000);
                        state1=(state1+1)&0x01;
                        if(state1==0){
                                write(LED2,"0",1);
                                write(LED1,"0",1);
                        }else{
                                write(LED2,"1",1);
                                write(LED1,"1",1);
                        }
                }
                if (get_SW2() =='0'){
                        usleep(10000);
                        while (get_SW2() == '0');
                        usleep(10000);
                        state2=(state2+1)&0x01;
                        if(state2==0){
                                write(LED0,"0",1);
                        }else{
                                write(LED0,"1",1);
                        }
                }
        }
    }
```

Fig. 7-15 using tactile switches (C language) (3/3)

Operation check and description (How to use this sample program)
Save the program above in the file "step3.c" and compile using "$gcc step3.c –o step3".
Execute the program with "$ Step3" and then press the tactile switch to turn LED on.
"/dev/rtswitch??" command sends 2-byte information. One is a switch status, "0" or "1", and the second is the line feed code. "read" receives 2-byte data and one data is used. If the statement of switch input is written in the main function, the program becomes too long and complicated. Therefore, the statement regarding switching is included in the new function.

7.3.3 Python

```
#!/usr/bin/python
import time

state0=state1=state2=0

while 1 :
	with open("/dev/rtswitch0","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch0","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state0 = (state0 + 1 ) & 1
			with open("/dev/rtled3","w") as f:
				f.write(str(state0))
	with open("/dev/rtswitch1","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch1","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state1 = (state1 + 1 ) & 1
			with open("/dev/rtled2","w") as f:
				f.write(str(state1))
			with open("/dev/rtled1","w") as f:
				f.write(str(state1))
	with open("/dev/rtswitch2","r") as f:
		if f.readline() == "0\n" :
			time.sleep(0.1)
			while 1 :
				with open("/dev/rtswitch2","r") as f:
					if f.readline() != "0\n" :
						break
			time.sleep(0.1)
			state2 = (state2 + 1 ) & 1
			with open("/dev/rtled0","w") as f:
				f.write(str(state2))
```

Fig. 7-16 using tactile switches 3 (python)

Operation check and description (How to use this sample program)
Save the program above in the file "step3.py" and execute the command "$python step3.py". Press the tactile switch to turn LED on. Python utilizes a built-in function to convert numbers to characters. The program uses this function to directly output the state value to LED.

8 Remarks

8.1 References

"Let's start robot assembling using Raspberry Pi", June to October 2015, Nikkei Linux
Spring 2016, Nikkei BP Raspberry Pi magazine

8.2 Copy right

9 Contact

Contact us by e-mail or access our URL below.

RT Corporation
Yamaguchi Bld. 3F, 3-2-13 Sotokanda, Chiyoda-ku, Tokyo 101-0021, Japan
Email:shop@rt-net.jp
URL: http://www.rt-net.jp

10 Revision history

Version	Description	Date
1.0	-	August 04, 2015
1.1	- A path on Fig. 3-1 was revised. - Access method in Fig. 3-2 was revised. - Table 1 Musical scale was added.	September 15, 2015
2.0	- Illustrations were changed to the ones of Raspberry Pi MouseV2. - Assembling processes of Raspberry Pi Mouse were added. - Install processes of Raspbian OS was added. - Initial user name and password were added in the instruction of OS installation. - Description regarding device drivers was added in the device driver installation that they are used commonly between Raspberry Pi 2 and 3. - A sample program using an additional function "echo Left Hz Right Hz Time ms > /dev/rtmotor0 was added in the motor operation. - An example of using device driver was added. -Translate to English	Dec. 7th, 2016