Technical Information

Ubuntu Image for F3RP70 User's Guide

TI 34M06T02-02E

YOKOGAWA
Yokogawa Electric Corporation

TI 34M06T02-02E
Mar. 2021 2nd Edition (YK)

Ubuntu Image for F3RP70 User's Guide

TI 34M06T02-02E 2nd Edition

Contents

Appendix1 I/O Module Access Library

Appendix2 Web Maintenance Tool

Introduction

■ Overview

This manual describes how to use the Ubuntu image, which is provided for the OS-free CPU module.
The OS-free CPU module is e-RT3 CPU module that incorporates only a boot loader. Users can develop their own system, while it takes time and effort to gain knowledge for using the module.
Use of the Ubuntu image allows you to easily take advantage of a system with a combined set of some open-source software.

■ Other Instruction Manuals

In addition to this manual, refer to the following manuals.

Product manuals
- e-RT3 CPU Module (F3RP7□) Hardware Manual (IM 34M06M52-01E)
- e-RT3 CPU Module (SFRD□2) BSP Common Function Manual (IM 34M06M52-02E)
- e-RT3 OS-free CPU Module Startup Manual (IM 34M06M52-25E)

Related manuals
- Hardware Manual (IM 34M06C11-01E)
- Analog Input Modules (IM 34M06H11-02E)
- Analog Output Module (IM 34M06H11-03E)
- High-speed Data Acquisition Module (F3HA06-1R, F3HA12-1R) (IM 34M06G02-02E)
- Temperature Monitoring Module (IM 34M06H63-01E)

*This manual contains current information as of March 2021.
The features or specifications of the product may be subject to change in the future.

1. F3RP70-2L

1.1 Overview

F3RP70-2L is one of the models in the e-RT3 CPU modules. It incorporates a boot loader only and allows its users to construct a flexible system, including the operating system.

After F3RP70-2L is turned on, the boot loader starts its operation and initializes hardware and e-RT3/FA-M3 modules. The boot loader of F3RP70-2L provides the features of starting the OS according to the setup parameters and of self-diagnosing the module, based on the state of the MODE switch.

1.2 Ubuntu image

Ubuntu image file to be installed in F3RP70-2L for easy use is provided. The Ubuntu image file available on the e-RT3 website allows you to start development early.
You will store this provided Ubuntu image file in an SD memory card before using the image. To use it, follow the procedure in the next chapter to write the operating system into the SD memory card and then insert the card into F3RP70-2L.

2. Writing the Ubuntu image file to the SD memory card and startup

This chapter describes the procedure for writing the Ubuntu image file to an SD memory card and startup.

2.1 Procedure overview

This section provides an overview of the procedure for writing the Ubuntu image file to an SD memory card and startup.
For details on the procedure, refer to “2.3 Procedure for writing to the SD memory card” and “2.4 Starting from the SD memory card” of this manual.

■ Writing to an SD memory card

Use the following procedure to write the Ubuntu image file to an SD memory card:

- Download the Ubuntu image file from the Yokogawa website.
- Let your PC recognize an SD memory card.
- Use a tool for writing disk images to write the Ubuntu image file to the SD memory card.

■ What you need

You need to have the following items for the write to the SD memory card:

- PC that supports SD memory cards
- SD memory card (SDHC card: 4 to 32 GB)
- Tool for writing disk images
- Ubuntu image file

You need to have the following items for starting Ubuntu from SD memory card:

- PC
- Terminal software (ex. PuTTY, tera term or.)
- RS-232-C conversion cable (KM72-2N) or Ethernet cable

2.2 The SD memory card for starting

This section describes the SD memory card image you create in this chapter.

The SD memory card image consists of all copied files of the Ubuntu operating system (OS) that runs on F3RP70-2L and a collection of setting files necessary for starting the OS. The OS section contains the OS settings as well as the stored files.
By inserting the SD memory card that has the Ubuntu image into an SD memory card slot of F3RP70-2L, you can start Ubuntu with F3RP70-2L-suitable settings and necessary libraries and packages installed in it.

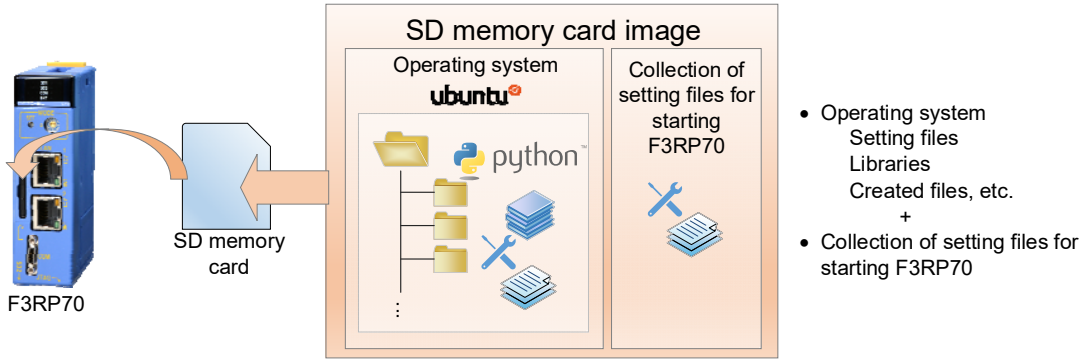


Figure 2.1 Description of the SD memory card for starting

2.2.1 Specifications of the Ubuntu image

● Revision

The revision of the Ubuntu image is confirmed in the file below.

Revison	File
R.1.1.1	None
R.1.2.1 or later	/usr/local/etc/sfrd14-release

● OS

Ubuntu18.04LTS, GNU/Linux4.14LTS+PREEMPT_RT is started.
With the following command, you can see kernel configuration of the Ubuntu image.

```
$ zcat /proc/config.gz
```

Same information is described in the file /boot/config-xxx-ert3xlnx (xxx is version of kernel).

● Ubuntu development package

Python 3 and the build-essential toolchain are available as a program development environment.

- **e-RT3 module access**

  It provides the API functions for working with various e-RT3 I/O modules and e-RT3 CPU and sequence CPU modules in the multi-CPU configuration, together with the signal notification feature used for synchronization operations between CPU modules.

Note

For details on the API functions for e-RT3 I/O module access, refer to “Appendix1 I/O Module Access Library” of this document.

- **PLC device access**

  PLC device access is a feature to emulate the structure of data in a sequence CPU module. It provides a service for connecting programmable indicators through PC link commands (specifications from Yokogawa) and a mechanism for shared devices in the multi-CPU configuration.
  It also offers the API functions for working with these PLC devices.

Note

For details on the API functions for e-RT3 I/O module access, refer to “Appendix1 I/O Module Access Library” of this document.

- **External equipment communication service**

  It provides a communication feature with external equipment, such as indicators and PCs, via the command interface. With this service, you can monitor and configure CPU devices and work with programs in sequence CPU modules to operate or stop them via external equipment.

- **RAS**

  It provides the API functions for examining or monitoring failures in systems and a mechanism for receiving alarms when a failure occurs. You can receive alarms from the momentary power failure detection feature for power supply voltage or about abnormal temperatures of CPU modules.

Note

For details on the features above, refer to “e-RT3 CPU Module (SFRD□2) BSP Common Function Manual” (IM 34M06M52-02E).

- **Web Maintenance Tool**

  This tool offers features for monitoring and configuring I/O modules and internal

parameters of the system provided by the Ubuntu image.
It is available on a Web browser, such as Google Chrome. Therefore, end users who do not have any development environment and engineers in charge of maintenance or launching can easily work on their configuration or maintenance tasks on the Web browser regardless of their PC environment.

Note

For details on the features above, refer to “Appendix2 Web Maintenance Tool” of this document.

- **Python 3 related packages**

The Python-related packages listed in the table below are installed.
If necessary, use the apt command or the pip3 command to add or remove a package.

No.	Class	Package
1	Machine learning	scikit-learn
2	Numerical processing	numpy
3	Numerical processing	pandas
4	Numerical processing	scipy
5	Graph drawing	matplotlib
6	Communication	pymodbus
7	Development environment	jupyter-notebook
	Development environment	ptvsd

2.2.2 User settings

The Ubuntu OS provided by this image file has the users below.
If necessary, change the password or add or remove a user.

- **Root user**

User name: root
Password: root_ert3

- **Ordinary user**

User name: ert3
Password: user_ert3

2.2.3 Network settings

The Ubuntu OS provided by this image file has the network settings below.
Change them to suit the user's environment. The Ubuntu OS starts with the new settings if you reboot it after modifying the setting file.

- **eth0 (LAN port 1)**

  IP address: 192.168.3.72
  Network mask: 255.255.255.0

  Setting file:
  /etc/systemd/network/10-eth0.network

  A setting example for stable IP address is described below. You should modify “Address”, “Gateway” and “Destination” for your environment.

```
[Network]
Address=192.168.3.72/24
[Route]
Gateway=192.168.3.1
Destination=192.168.3.0/24
```

- **eth1 (LAN port 2)**

  IP address: get from DHCP
  Network mask: get from DHCP

  Setting file:
  /etc/systemd/network/20-eth1.network

2.3 Procedure for writing to the SD memory card

This section describes the detailed procedure for writing the Ubuntu image file to the SD memory card.

Note

In this procedure, all components (including your settings and applications) in SD memory card are overwrote. When you use new version of Ubuntu image, you shall re-install your settings and applications in the new Ubuntu.

2.3.1 Environment installation

This subsection describes the environment necessary for the tasks in this section.

- **PC**

  You need to have a PC that meets the following criteria:

  - It supports SD memory cards.
    You need to use a PC with a built-in SD memory card drive, or have an external SD memory card reader and connect it to your PC.

  - It supports a given tool for writing disk images.

- **SD memory card**

  F3RP70-2L supports an SDHC memory card with a capacity of 4 to 32 GB.
  We recommend that you use a card with a higher program/erase cycle, such as an SLC- or MLC-type card.

- **Ubuntu image file**

  You download it from our website “Yokogawa Partner Portal”.
  Access the following URL and download “OS image file for OS-free CPU Module”
  URL: https://partner.yokogawa.com/global/itc/index.htm

- **Tool for writing disk image files**

  You can have any tool for writing disk image files.
  This manual shows a procedure for Rawrite32, free software for Windows.

  How to install

  1. Access the following URL and click the [Download] link at the top of the Rawrite32 website.

https://www.netbsd.org/~martin/rawrite32/index.html

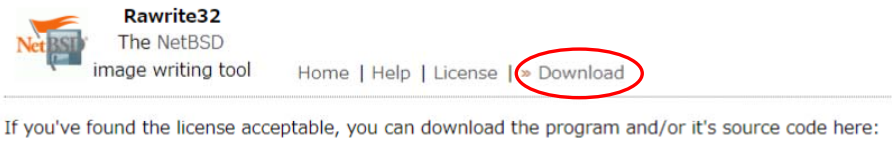


Figure 2.2 Download link for Rawrite32

2. Click the [rw32-setup-1.0.7.0.exe] button to download the file.

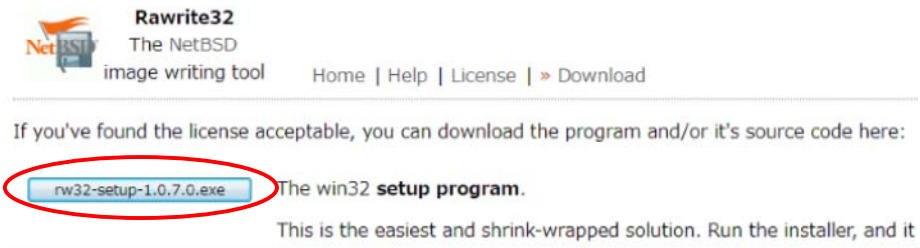


Figure 2.3 Selecting the file for Rawrite32

3. Open the downloaded file to start the installer.
   If you see a dialog box saying “Do you want to allow this app to make changes to your device?” instead of the installer being started, click [Yes]. The installer is then started.

4. Without making particular changes to the settings, click the [Install] button. The installation is now started.

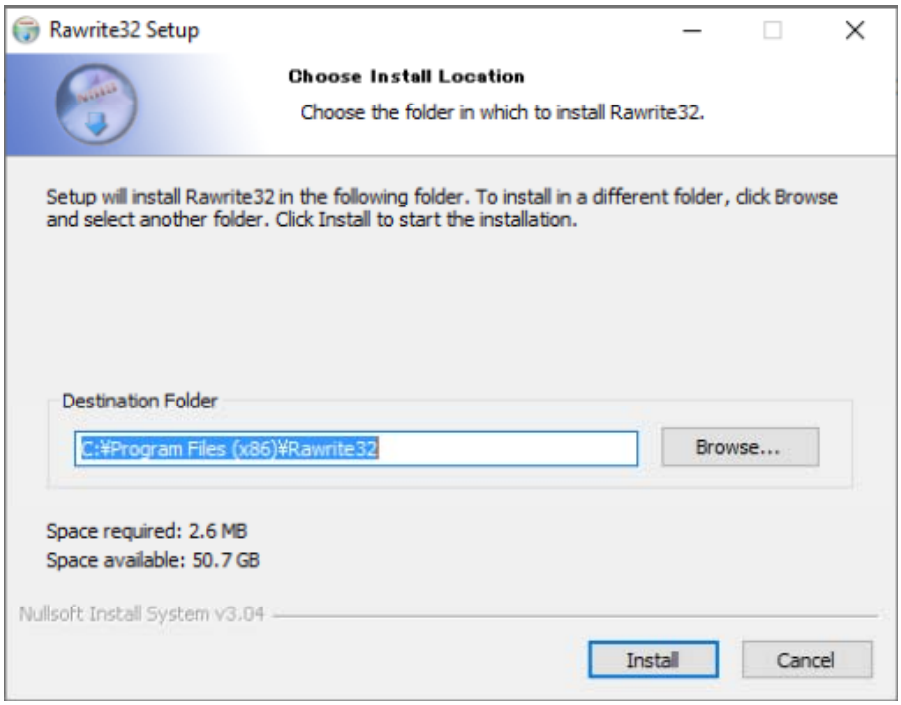


Figure 2.4 Rawrite32 setup dialog box

5. Once the installation is complete, click the [Finish] button to exit the installer.

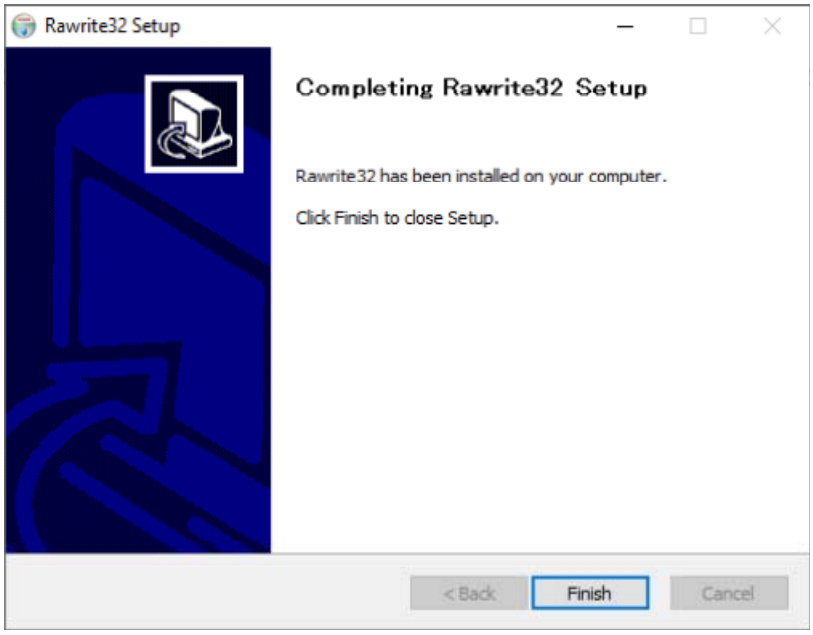


Figure 2.5 Complete Rawrite32 installation screen

Note

The following PC environment was used to check the procedure described in this section.
- OS: Windows 10 Enterprise (64-bit version)

- SD memory card support: Built-in SD memory card drive

The following SD memory card is available:
- SDHC memory card (4 to 32 GB)

The size of the Ubuntu image file for use ranges from 1 to 2 GB. Choose the capacity of your card by considering the fact that data is also stored in the SD card while you are using F3RP70-2L.
For details on the recommended standard and the use of the SD memory card slot of e-RT3, refer to “4.5 SD memory card” of “e-RT3 CPU Module (F3RP7□) Hardware Manual” (IM 34M06M52-01E).

In the procedure described in this section, you do not have to uncompress the downloaded file.

You cannot use a general operation for pasting a file to write the SD memory card image to the SD card. Make sure that you have a tool for writing disk image.

2.3.2 How to write to the SD memory card

This subsection details the writing procedure.

- **Let your PC recognize an SD memory card**

  Before starting Rawrite32, SD memory card have to be recognized by PC.

- **Start Rawrite32**

  If you see a dialog box saying “Do you want to allow this app to make changes to your device?” instead of Rawrite32 being started, click [Yes]. Rawrite32 is then started.
  In the startup screen, check that [Target] is set to the location of the SD memory card drive and the capacity of the card is indicated in [ ]. In the following example, a 32-GB SD memory card is used.

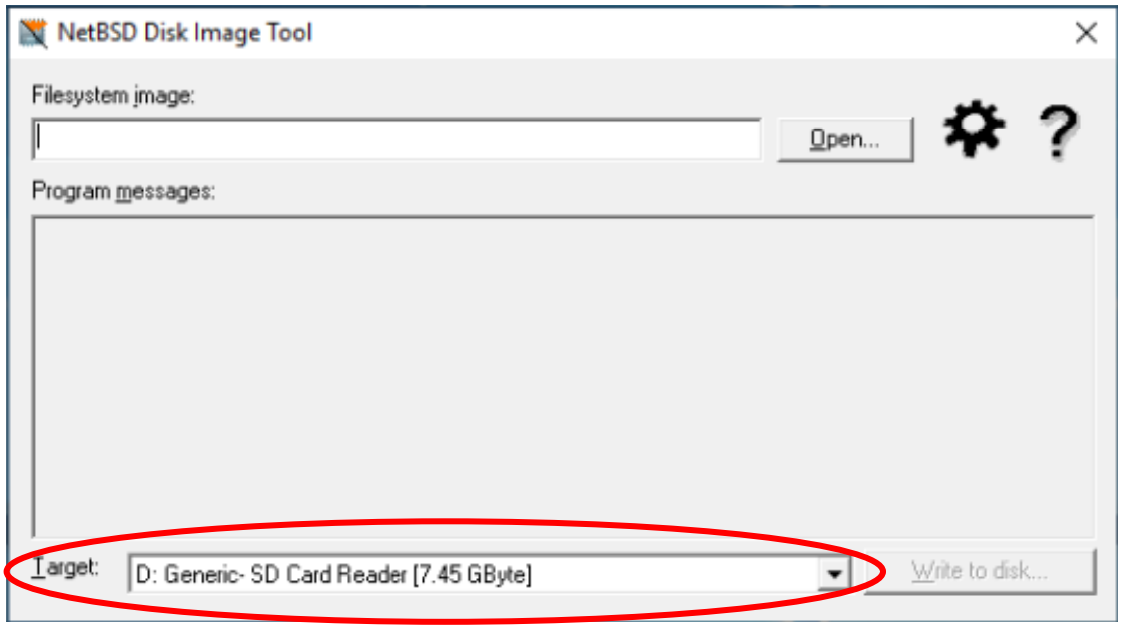


Figure 2.6 Rawrite32 startup screen

- **Select the Ubuntu image file to be written**

  At the top right of the screen, click the [Open...] button and select the compressed Ubuntu image file you downloaded. Hash values are then calculated and displayed in the [Program messages] section in the middle of the screen.
  The [Write to disk...] button is also activated at the bottom right of the screen so that you can click it.

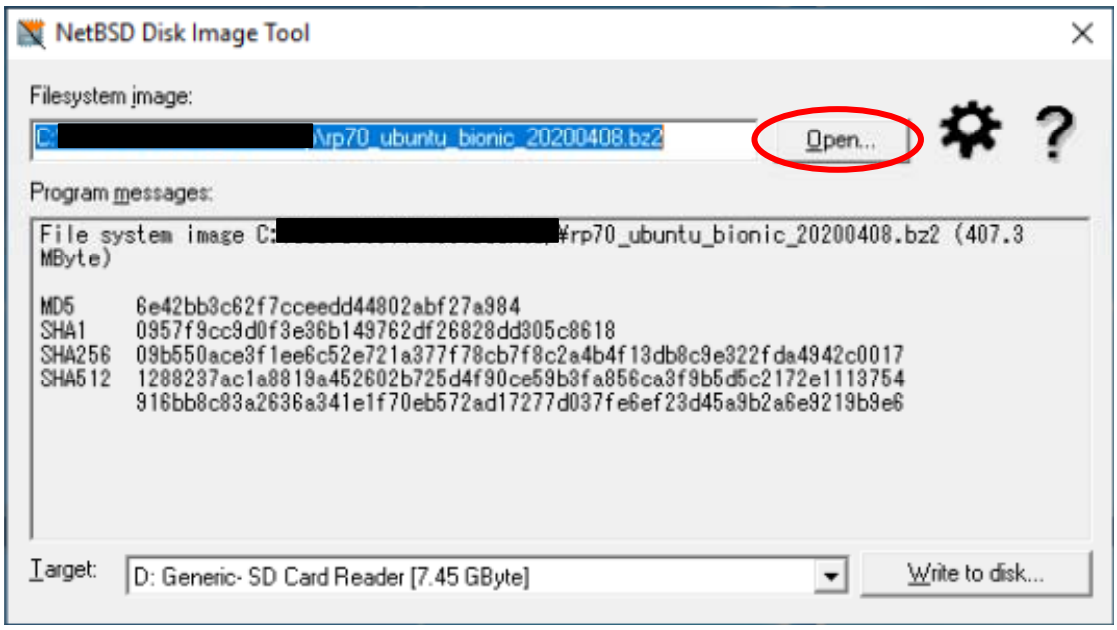


Figure 2.7 Complete Ubuntu image loading screen

● Write the image

At the bottom right of the screen, click the [Write to disk...] button to open the dialog box as shown in the figure below. Click the [Yes] button to start writing to the SD memory card.

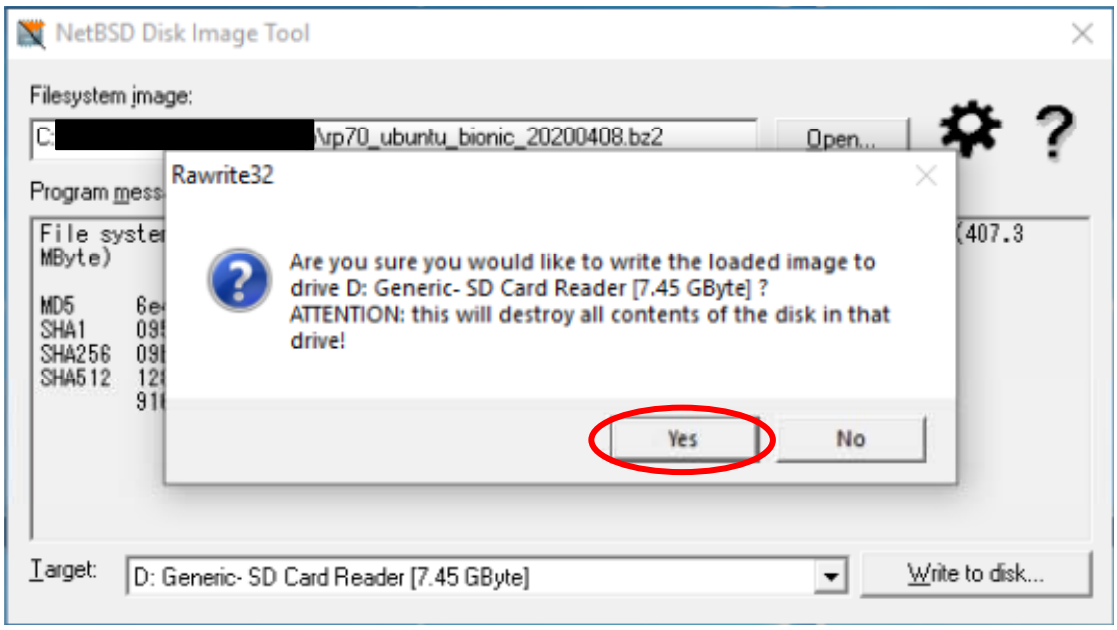


Figure 2.8 Write confirmation screen

● Confirm the completion of writing

The writing is complete when you see the message saying “successfully written to disk” in the [Program messages] section, as shown in the figure below. At the top right of the screen, click the [x] button to exit Rawrite32.

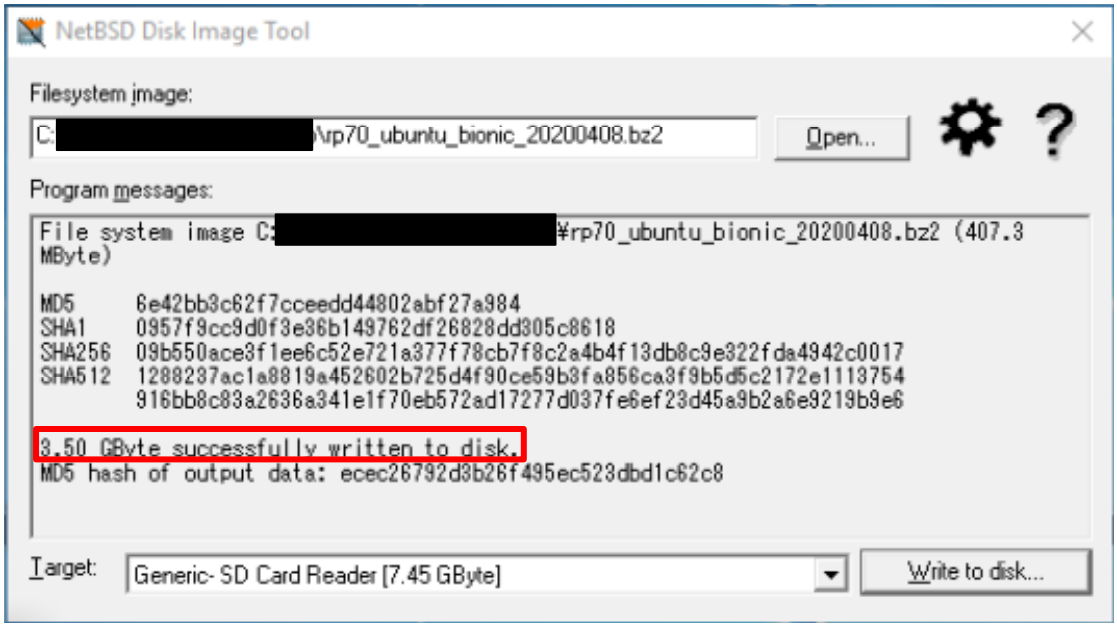
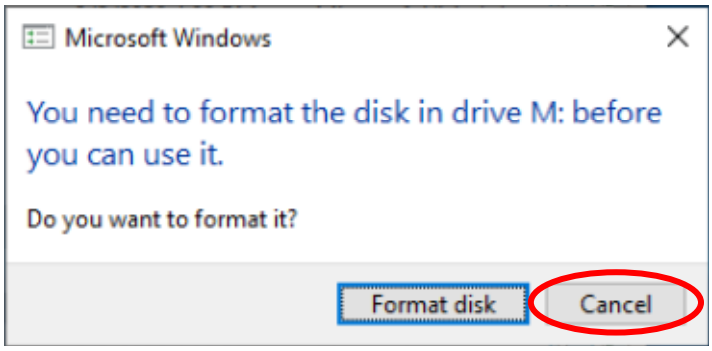


Figure 2.9 Writing completed screen

Note

When you perform the procedure in this section, all the data in the SD memory card is overwritten. Use a blank SD memory card, or back up the data beforehand.

After the writing is completed, you sometimes see a dialog box that request you to format the SD memory card. If this happens, cancel the format.



This is because the written image file contains a Linux file system (ext4) that cannot be read by Windows. If you format the card accidentally, follow the procedure in this subsection to write to the SD card again.

2.4 Starting from the SD memory card

This section describes how to start the Ubuntu image file written to the SD memory card.

2.4.1 Procedure of startup

This subsection details the startup procedure.

- **Insert the SD memory card**

Insert the SD memory card into SD slot 1 or 2 of F3RP70.
If two memory cards are inserted at the same time, the image in slot 1 is used in preference to the one in the other slot.

When you use SD slot 2, you have to set “rootdev” environment variable of u-boot to ”/dev/mmcblk1p2”. And when you use SD slot 1, you have to remove “rootdev” environment variables.

Example for setting “rootdev” to “/dev/mmcblk1p2”

```
f3rp7x> setenv rootdev /dev/mmcblk1p2
f3rp7x> saveenv
```

Example for removing value of “rootdev”

```
f3rp7x> setenv rootdev
f3rp7x> saveenv
```

Note

For details on environment variable of u-boot, refer to “e-RT3 OS-free CPU Module Startup Manual” (IM 34M06M52-25E).

- **Start the system**

With the MODE switch set to 0, turn on the power.

2.4.2 Procedure of log in to Ubuntu

This subsection details log-in procedure.

- **What you need**

You need to have the following items for log-in to Ubuntu using serial console:

- PC that is installed terminal software
- RS-232-C conversion cable (KM72-2N)
- USB-serial converter (when your pc doesn’t have seral port)

You need to have the following items for log-in to Ubuntu using SSH connection:

- PC that is installed terminal software
- Ethernet cable

Construction of devices

Figure 2.10 shows the construction of devices.
Log in to the Ubuntu through a serial console connection using the COM port at the front of the CPU module or from an SSH terminal using the LAN port. In this section, log in using the default value of eth0 (LAN port 1) shown in section 2.2.3 of this document, so connect the ethernet cable to LAN port1 on the upper front of the F3RP70-2L.

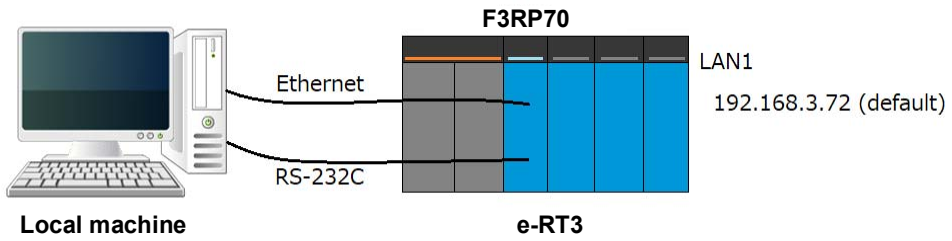


Figure 2.10 Construction of devices

Installing a terminal software

A terminal software, such as “Putty” or “Tera term”, is needed when you log in to the Ubuntu. This subsection describes installing procedure of “PuTTY” as an example.

1. Access the following URL and click the [here] in [Download PuTTY] contents.

Figure 2,11 Top page of PuTTY web site

2. Download the installer that matches your PC from the “Package files”.In this document, we will explain using the 64-bit version of “MSI (‘Windows Installer’)”

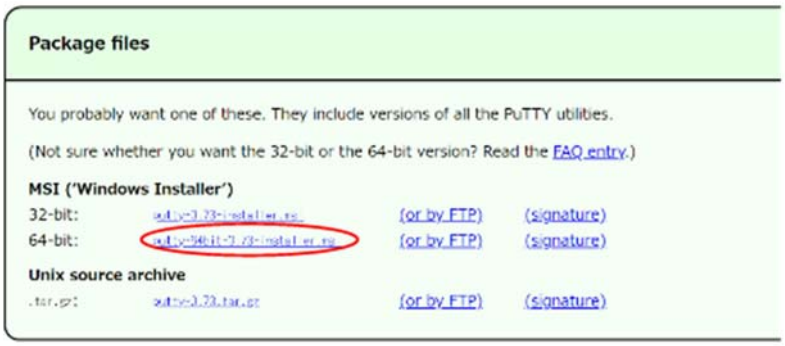


Figure 2.12 Download installer

3. Open the downloaded file to start the installer. When the following dialog is shown, Click the “Run”.

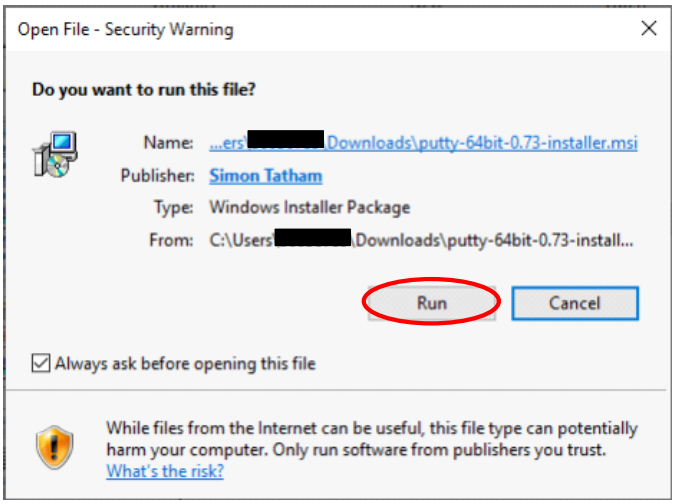


Figure 2.13 Security dialog

4. Click the “Next”.

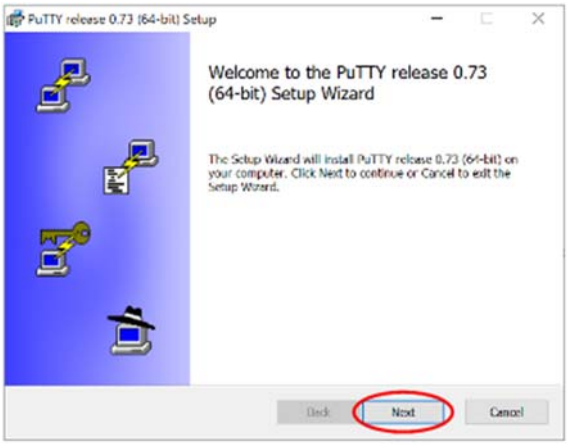


Figure 2.14 PuTTY installer

5. Specify the install location. In this document, do not change the destination folder and click “Next”.

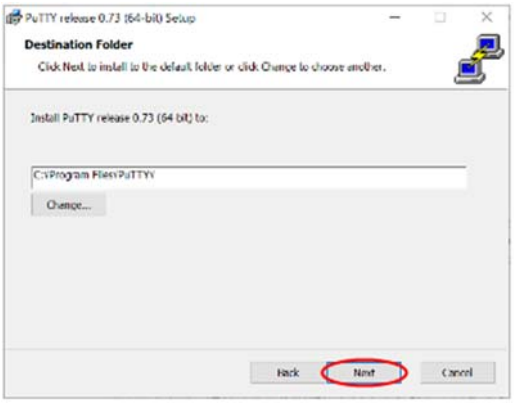


Figure 2.15 Specify the install location

6. Click “Next”.

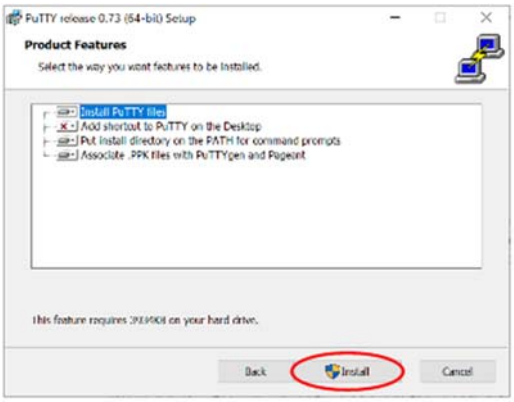


Figure 2.16 Selection of install components

7. When User Account Control dialg is displayed, click “Yes”.

8. Click “Finish” in the dialog of install completion. And then installation of “PuTTY” is completed.

Figure 2.17 Dialog of install completion

- **Log in to Ubuntu using serial console**

1. Start PuTTY and set “Connection type” to “Serial. And then set some items as follows and click “Open”

・ Serial-line: device of serial port
・ Speed: 115200

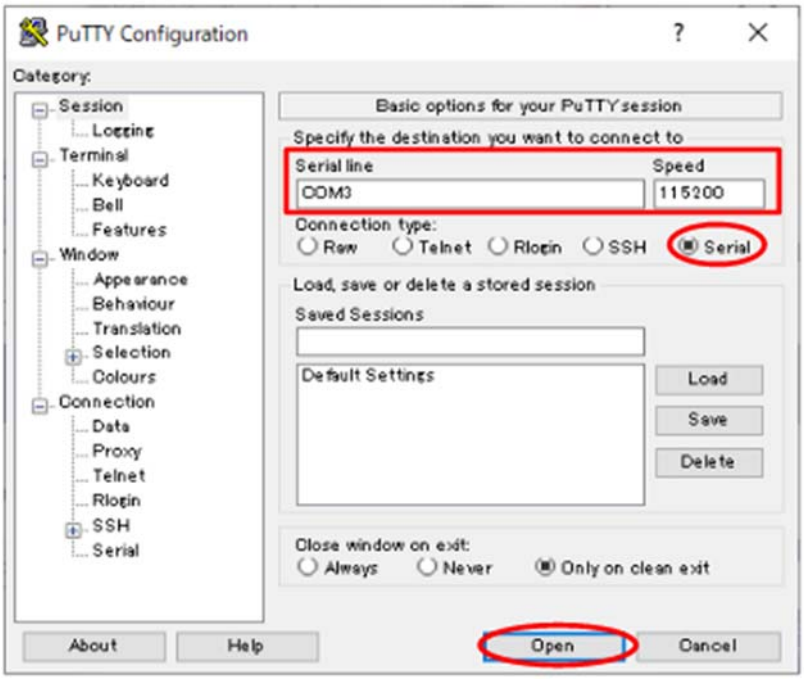


Figure 2.18 PuTTY setting

2. When connection to F3RP70-2L for the first time, the PuTTY Security Alert dialog is displayed. Click “Yes” to continue the connection.

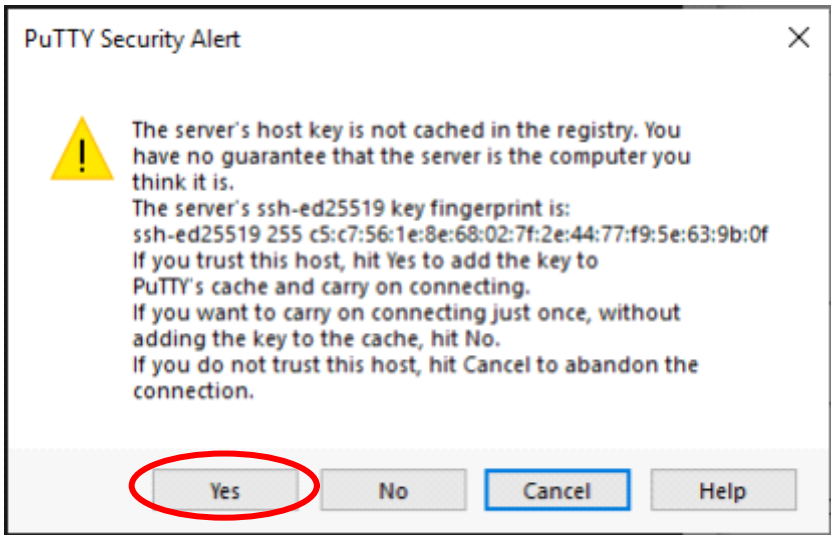


Figure 2.19 PuTTY Security Alert dialog

3. Turn on F3RP70-2L.

4. Login prompt is displayed on the console after boot sequence. Log in using the

user account you have set up.

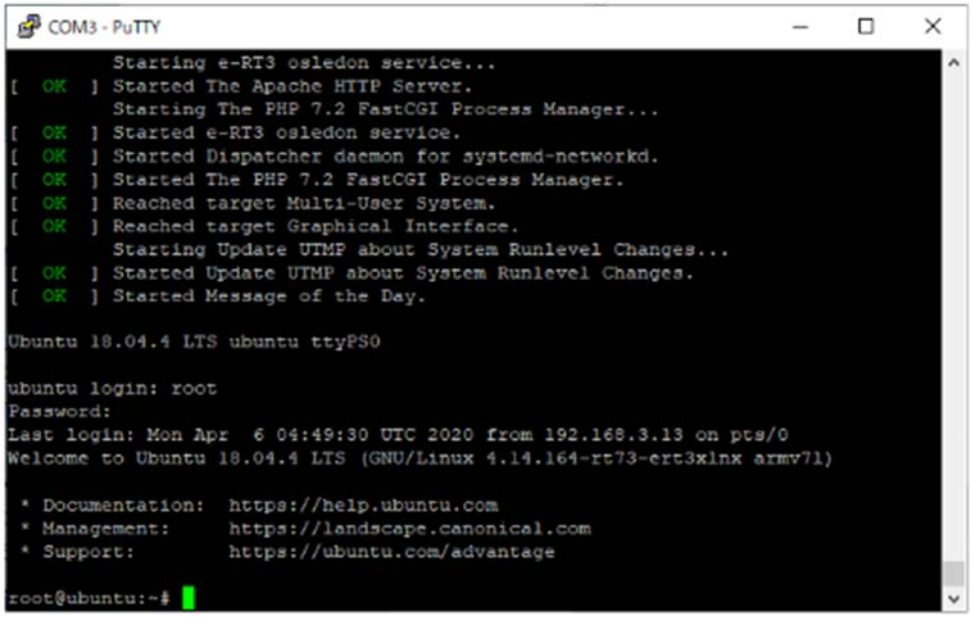


Figure 2.20 Login prompt of serial console

● Log in to Ubuntu using SSH connection

1. Turn on F3RP70-2L.

2. Set an IP address of your PC to “192.168.3.☐☐”

3. Start PuTTY and set “Connection type” to “SSH”. And then set some items as follows and click “Open”

・ Host Name (or IP address): 192.168.3.72
・ Port: 22

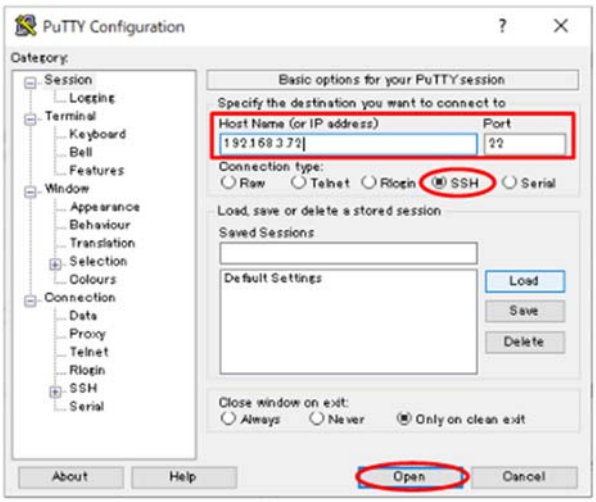


Figure 2.21 SSH setting of PuTTY

4. When connection to F3RP70-2L for the first time, the PuTTY Security Alert dialog is displayed. Click “Yes” to continue the connection.

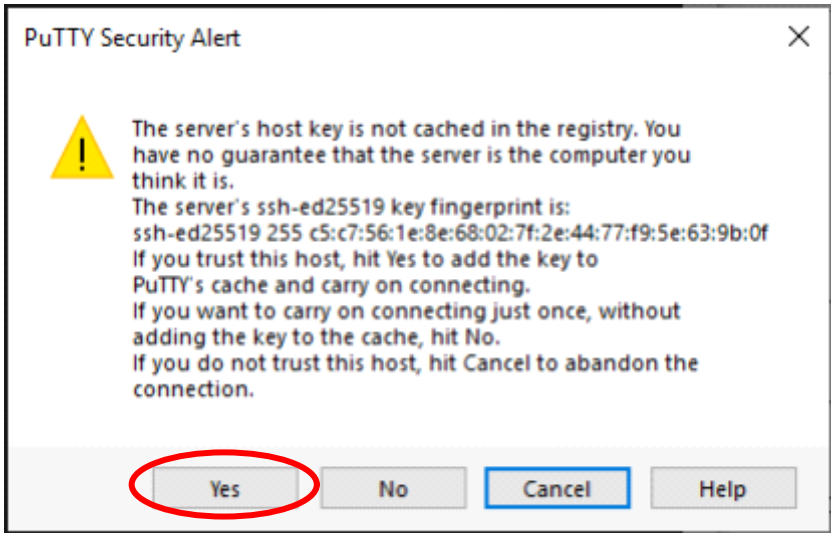


Figure 2.22 PuTTY Security Alert dialog

5. Login prompt is displayed on the console. Log in using the user account you have set up.

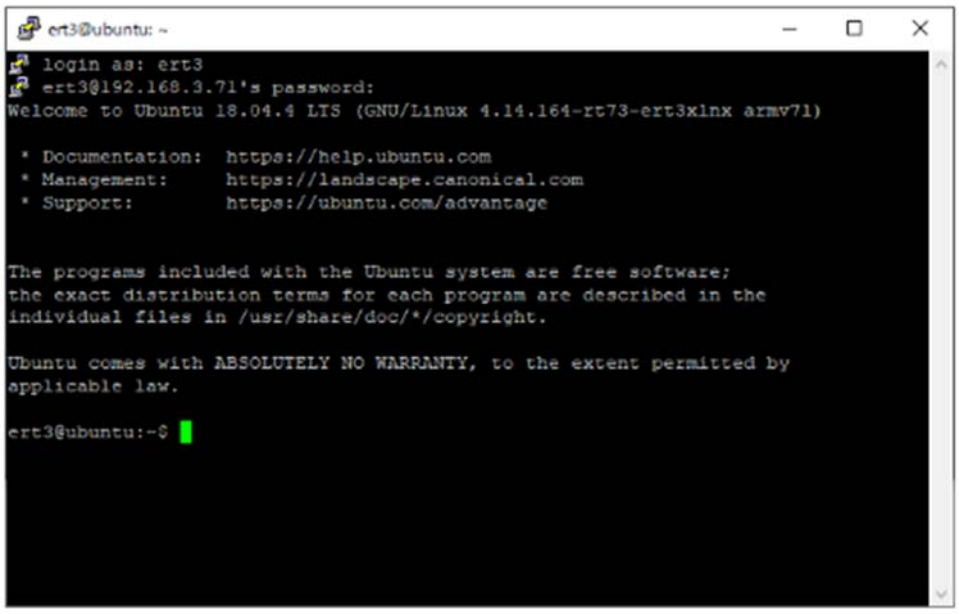


Figure 2.23 Login prompt of SSH connection

Note

If you change the settings of your computer's network adapter according to the instructions in this manual, you may not be able to connect to the Internet using that adapter.
Connect the F3RP70-2L to a port that is not normally used for internet connection or restore the settings after the connection is completed.

When you want to connect to SSH, please do after starting F3RP70-2L. Immediately after starting, the SSH server may not start and connection may fail.
With the initial settings downloaded Ubuntu image from the Yokogawa web site, you cannot use the SSH connection to log in with the root user account. Please log in with an ordinally user account.
For the default value of the user account, see "2.2.2 User setting" in this manual.

2.4.3. Enable the sudo command

In this Ubuntu image, the sudo command cannot be used by general users in the default state. If you want to use commands that require root privileges, enter the settings in this section to enable them. The following are some examples of when the sudo command is not available.

```
# Operation with general user username
$ sudo ls -a /root
[sudo] password for username:     # Enter Password
username is not in the sudoers file. This incident will be reported.
```

- **Enabling the sudo command**

1. Confirm the group of the user for whom you want to enable the sudo command. If sudo is not included in the group, the sudo command is not available.

```
# For general user username
$ groups username
username : username         # User name: Group
```

2. Since the operation is performed with root privileges, switch to the root account.

```
$ su
Password:                   # Enter root Password
root@ubuntu:/home/username#
```

3. Add the user for whom to enable the sudo command to the sudo group.

```
# gpasswd -a username sudo
Adding user username to group sudo
```

4. Confirm that sudo was added by checking the user's group in the same procedure as in 1. Once added, log out of the root account. The settings will be reflected when you log back in, so also log out of the general user account.

```
# groups username
username : username sudo    # User name: Group
# exit
```

```
exit
$ exit
logout
```

5. When you log in for the first time with the account that you added to the sudo group, the following appears, indicating that the sudo command is enabled.

```
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

username@ubuntu:~$
```

6. Confirm that the sudo command is received.

```
$sudo ls -a /root
[sudo] password for username:        # Enter Password
. .. .bash_history .bashrc .cache .gnupg .profile
username@ubuntu:~$
```

- **Disabling the sudo command**

  To disable the sudo command, follow the steps below. Cancel membership in the sudo group; the opposite of when you enabled it.

1. Just as with enabling the sudo command, you need root privileges, so switch to the root account as you did in enable step 2.

2. Remove the user for whom you want to disable the sudo command from the sudo group.

```
# gpasswd -d username sudo
Removing user username from group sudo
```

3. Using the same procedure as in enabling 1., check the user’s group. Once you have confirmed that sudo has been deleted, log out of the root account. The settings will be reflected when you log back in, so also log out of the general user account.

```
# groups username
username : username
# exit
exit
$ exit
logout
```

4. When you login to the account from which you canceled sudo group

membership, the sudo command is disabled.

Note

In this Ubuntu image, the ert3 default general user does not belong to the sudo group. To prevent unexpected operations, we recommend disabling the sudo command during operation.

3. e-RT3 I/O module configuration service

This chapter describes the features of IO module configuration service and how to use the service.

3.1 Functional overview

The IO module configuration service is a service that configures e-RT3 I/O modules.
In e-RT3 I/O modules, a single module can handle various input and output signals. For example, the F3AD08-6R analog input module can handle voltage signals or current signals as input signals of various ranges. It also has module-specific features, such as scaling and filtering. You can select these ranges of input signals and use the specific features by setting parameters for the configuration area of each module. In general, you need to specify these parameters with user programs before handling data, when you use I/O modules.
The IO module configuration service allows you to automatically configure the features on these modules according to the setting file. You do not have to write programs in order for configuring modules and therefore you can create programs dedicated for data processing.

Table 3.1 lists e-RT3 I/O modules supported by the module configuration service.

Table 3.1 Modules supported by the module configuration service

Type of module	Model of module	Overview of specification
Digital input	F3XD08-□□	8-bit digital input
	F3XD16-□□	16-bit digital input
	F3XD32-□□	32-bit digital input
	F3XD64-□□	64-bit digital input
Digital output	F3YD04-□□	4-bit digital output
	F3YD08-□□	8-bit digital output
	F3YD14-□□	14-bit digital output
	F3YD32-□□	32-bit digital output
	F3YD64-□□	64-bit digital output
Analog input	F3AD04-5R	4-channel voltage input (0 to 5 V, 1 to 5 V, -10 to 10 V, 0 to 10 V)
	F3AD08-5R	8-channel voltage input (0 to 5 V, 1 to 5 V, -10 to 10 V, 0 to 10 V)
	F3AD08-6R	8-channel voltage input (0 to 5 V, 1 to 5 V, -10 to 10 V, 0 to 10 V) or 8-channel current input (0 to 10 mA, 0 to 20 mA, 4 to 20 mA)
	F3AD08-4R	8-channel current input (0 to 10 mA, 0 to 20 mA, 4 to 20 mA)
Analog output	F3DA04-6R	4-channel voltage output (-10 to 10 V, 0 to 10 V, 0 to 5 V, 1 to 5 V) or 4-channel current output (4 to 20 mA, 0 to 20 mA, -20 to 20 mA)
	F3DA08-5R	8-channel voltage output (-10 to 10 V, 0 to 10 V, 0 to 5 V, 1 to 5 V)
High-speed data acquisition	F3HA06-1R	6-channel voltage input
	F3HA12-1R	12-channel voltage input
Temperature monitor	F3CX04-0N	4-channel thermocouple (K, J, T, B, S, R, N, E, L, U, W, Platinel) or 4-channel resistance temperature detector (JPt100, Pt100) or 4-channel voltage input (0 to 10 mV, 0 to 100 mV, 0 to 1 V, 0 to 5 V, 1 to 5 V, 0 to 10 V)

Note

For details on the module specifications, refer to each manual.
Refer to “Hardware Manual” (IM 34M06C11-01E) for details on the digital I/O modules, “Analog Input Modules” (IM 34M06H11-02E) for the analog input modules, “Analog Output Module” (IM 34M06H11-03E) for the analog output modules, “High-speed Data Acquisition Module (F3HA06-1R, F3HA12-1R)” (IM 34M06G02-02E) for the high-speed data acquisition modules, and “Temperature Monitoring Module” (IM 34M06H63-01E) for the temperature monitor module.

3.2 Usage

The module configuration service is provided as a daemon managed by systemd. Systemd is a utility platform for daemon management designed for Linux. Using the systemctl command, you can configure services to start or stop and whether they are run automatically on startup.

3.2.1 Setting file

- **File format**

The setting file is written in JSON format.
A single file contains all module settings. Specify a setting name as a JSON key and a setting for each module as a value.

```
{
  " setting name 1 ":{ "unit":m, "slot":k, "modid":"module ID", ... },
  " setting name 2 ":{ setting for each module },
  " setting name 3 ":{ setting for each module }
}
```

The *setting name* can accept any string. You can specify it as you want for identification because it does not matter in terms of the settings. The subsequent sections describe the settings for each module. As the settings common to all modules, specify the unit number and slot number of the unit and slot to which the module is inserted, and the module ID. If the module ID of the I/O module inserted at the position specified by the unit number and slot number does not match the module ID in the setting file, the settings of the module are ignored.

Note

The unit number and slot number of an I/O module are numbers that indicate where the module is inserted. For details, refer to “e-RT3 CPU Module (SFRD□2) BSP Common Function Manual” (IM 34M06M52-02E).

- **File path**

The setting file should be stored in the following path:
/usr/local/etc/ert3/ert3io.conf

3.2.2 Working with the daemon

With the systemctl command, you can work with the ert3ioconfd daemon. It performs the following actions on the systemd commands.

Note

The user that has root privilege can use “systemctl” command. When you use it, use “sudo” command or switch to root user with “su” command.

● Start configuration

With the start command, you can run configuration manually. The command configures the I/O module according to the setting file.

```
# systemctl start ert3ioconfd
```

● Stop configuration

With the stop command, you can stop the daemon. When the daemon is stopped, the I/O module is not accessed.

```
# systemctl stop ert3ioconfd
```

● Restart configuration

With the restart command, you can stop and start the daemon. Use this command when you modify the setting file and then reapply it to the I/O module.

```
# systemctl restart ert3ioconfd
```

● Enable or disable configuration on startup

With the enable or disable command, you can enable or disable the execution of the daemon on startup. If the daemon is enabled on startup, the I/O module is configured when the power is turned on according to the setting file.
Similarly, the disable command is used to disable the execution of it on startup.

```
# systemctl enable ert3ioconfd
# systemctl disable ert3ioconfd
```

● Check the configuration status

With the status command, you can check the running status of the daemon.

```
# systemctl status ert3ioconfd -n 40
```

The settings and setting errors in the loaded JSON file are displayed when you run a command. The n option can be used to change the maximum value for the lines to be displayed.
You can check whether the setting file contains proper information by comparing the information displayed with the information in the setting file. If expected information is not displayed, check the settings to see if the JSON file is written in the proper format or the unit number, slot number, and module ID are correct.

Note

The JSON information is sorted for display. Note that it is different from the order of the information in the setting file.

If an error occurs during a parameter setup, the information of the setting failure is displayed.

3.3 Setting file in detail

3.3.1 Digital input module

The setting format for a digital input module is shown below. Specify the input setting on a 16-bit basis.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "X01-X16":{"sampling":"input sampling period", "intr":"interrupt edge"},
  "X17-X32":{"sampling":"input sampling period", "intr":"interrupt edge"},
  "X33-X48":{"sampling":"input sampling period", "intr":"interrupt edge" },
  "X49-X64":{"sampling":"input sampling period", "intr":"interrupt edge" }
}
```

Table 3.2 shows the settings for digital input modules.
Enclose a string in "" and specify a number directly.

Table 3.2 Settings for digital input modules (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"XD08" "XD16" "XD32" "XD64"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3XD□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"X01-X16"	--		Sets an object consisting of sampling and intr for bits 1 to 16.
"X17-X32"	--		Sets an object consisting of sampling and intr for bits 17 to 32.
"X33-X48"	--		Sets an object consisting of sampling and intr for bits 33 to 48.
"X49-X64"	--		Sets an object consisting of sampling and intr for bits 49 to 64.
"sampling"	"always" "62.5us" "250us" "1ms" "16ms"		Specifies the sampling period as a string. By default, it is set to "16ms".
"intr"	"up" "down"		Specifies an interrupt edge. By default, it is set to "up". up: An interrupt occurs at the rising edge. down: An interrupt occurs at the falling edge.

* Required key

- Setting example

```
{
  "unit":0, "slot":2,"modid":"XD32",
  "X01-X16":{"sampling":"16ms"},
  "X17-X32":{"sampling":"1ms"}
}
```

3.3.2 Digital output module

The setting format for a digital output module is shown below. Specify the output setting on a 16-bit basis.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "Y01-Y16":{"fail":"CPU failure output"},
  "Y17-Y32":{"fail":"CPU failure output"},
  "Y33-Y48":{"fail":"CPU failure output"},
  "Y49-Y64":{"fail":"CPU failure output"}
}
```

Table 3.3 shows the settings for digital output modules.
Enclose a string in "" and specify a number directly.

Table 3.3 Settings for digital output modules (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"YD04" "YD08" "YD14" "YD32" "YD64"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3YD□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"Y01-Y16"	--		Sets an object consisting of fail for bits 1 to 16.
"Y17-Y32"	--		Sets an object consisting of fail for bits 17 to 32.
"Y33-Y48"	--		Sets an object consisting of fail for bits 33 to 48.
"Y49-Y64"	--		Sets an object consisting of fail for bits 49 to 64.
"fail"	"hold" "reset"		Specifies the CPU failure output. By default, it is set to "hold". hold: Tells the module to continue to output the last value. reset: Tells the module to set the output value to 0.

* Required key

- Setting example

```
{
  "unit":0, "slot":3,"modid":"YD32",
  "Y01-Y16":{"fail":"reset"},
  "Y17-Y32":{"fail":"reset"}
}
```

3.3.3 Analog input module

The setting format for an analog input module is shown below.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "cycle":"conversion cycle", "drift":"drift correction",
  "ch1 ":{"range":"input signal range", "skip":"channel skip",
          "scaleup":digital output value corresponding to the upper limit of input signals,
          "scalelow":digital output value corresponding to the lower limit of input signals,
           "offset":offset value,
           "mslag":first-order lag filter time constant, "avepoint":moving average points},
  ...,
  "ch8":{}
}
```

Table 3.4 shows the settings for analog input modules.
Enclose a string in "" and specify a number directly.

Table 3.4 Settings for analog input modules (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"AD04" "AD08"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3AD□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"cycle"	"50us" "100us" "250us" "500us" "1ms" "16.6ms" "20ms" "100ms"		Specifies the A/D conversion cycle. The default value is "1ms".
"drift"	"enable" "disable"		Specifies whether the drift correction feature is enabled or disabled. The default value is "enable".
"ch1" to "ch8"	--		Sets an object for the channel.
"range"	"-10-10v" "0-10v" "0-5v" "1-5v" "0-20ma" "4-20ma"		Specifies the input range. The default value is: F3AD08-4R: "0-20ma" Other modules: "-10-10v"
"skip"	"yes" "no"		Specifies whether A/D conversion is skipped. The default value is "no". yes: No A/D conversion is performed. no: A/D conversion is performed.
"scaleup"	N (-30000≤N≤30000)		Digital output value that corresponds to the upper limit of input signals. The default value is 0 (no scaling).
"scalelow"	N (-30000≤N≤30000)		Digital output value that corresponds to the lower limit of input signals. The default value is 0 (no scaling).
"offset"	N (-5000≤N≤5000)		Offset value. The default is 0 (no offset).

"mslag"	0 to 30000		First-order lag filter [ms]. The default value is 0 (disabled).
"avepoint"	2^n (1≤n≤5)		Moving average points. The default value is 0 (disabled). It is enabled only when the first-lag filter is set to 0.

* Required key

- Setting example

```
{
  "unit":0, "slot":4,"modid":"AD08",
  "cycle":"250us", "drift":"enable",
  "ch1":{"range":"4-20ma","scaleup":10000,"scalelow":0,"mslag":1000 },
  "ch2":{"range":"0-5v","avepoint":16},
  "ch3":{"range":"-10-10v"},
  "ch4":{"skip":"yes"},
  "ch5":{"skip":"yes"},
  "ch6":{"skip":"yes"},
  "ch7":{"skip":"yes"},
  "ch8":{"skip":"yes"}
}
```

Table 3.5 shows the digital output values when scaling is disabled.

Table 3.5 Initial scaling settings for analog input modules

Input signal range	Digital output value
-10 to 10 V	-20000 to 20000
0 to 10 V	0 to 20000
0 to 5 V	0 to 10000
1 to 5 V	2000 to 10000
0 to 20 mA	0 to 10000
4 to 20 mA	2000 to 10000

Note

For details on the module specifications, refer to “Analog Input Modules” (IM 34M06H11-02E).

3.3.4 Analog output module

The setting format for an analog output module is shown below.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "ch":"output synchronization channel",
  "ch1 ":{"range":"output signal range", "fail":"CPU failure output",
          "scaleup":digital input value corresponding to the upper limit of output signals,
          "scalelow":digital input value corresponding to the lower limit of output signals},
           ...,
  "ch8":{}
}
```

Table 3.6 shows the settings for analog output modules.
Enclose a string in "" and specify a number directly.

Table 3.6 Settings for analog output modules (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"DA04" "DA08"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3DA□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"ch"	0 to 8		Specifies the channel number for synchronization output. The default value is 0. 0: No synchronization output. Other than above: Output synchronized with the specified channel.
"ch1" to "ch8"	--		Sets an object for the channel.
"range"	"-10-10v" "0-10v" "0-5v" "1-5v" "-20-20ma" "0-20ma" "4-20ma"		Specifies the output range. The default value is "-10-10v".
"fail"	N (-30000≤N≤30000)		Specifies an output value in CPU failure. If this key is not specified, the output value is maintained in a CPU failure.
"scaleup"	N (-30000≤N≤30000)		Digital input value that corresponds to the upper limit of output signals. The default value is 0 (no scaling).
"scalelow"	N (-30000≤N≤30000)		Digital input value that corresponds to the lower limit of output signals. The default value is 0 (no scaling).

* Required key

- Setting example

```
{
  "unit":0, "slot":5,"modid":"DA04",
  "ch1":{"range":"4-20ma","scaleup":10000,"scalelow":0},
  "ch2":{"range":"4-20ma","scaleup":10000,"scalelow":0},
  "ch3":{"range":"1-5v","scaleup":10000,"scalelow":0},
  "ch4":{"range":"1-5v","scaleup":10000, "scalelow":0}
}
```

Table 3.7 shows the digital output values when scaling is disabled.

Table 3.7 Initial scaling settings for analog output modules

Output signal range	Digital input value
-10 to 10 V	-20000 to 20000
0 to 10 V	0 to 20000
0 to 5 V	0 to 10000
1 to 5 V	2000 to 10000
-20 mA to 20 mA	-10000 to 10000
0 to 20 mA	0 to 10000
4 to 20 mA	2000 to 10000

Note

For details on the module specifications, refer to “Analog Output Module” (IM 34M06H11-03E).

3.3.5 High-speed data acquisition module

The setting format for a high-speed data acquisition module is shown below.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "cycle":"data acquisition cycle",
  "ch1 ":{"range":"input signal range", "scale":"enable/disable",
          "scaleup":digital output value corresponding to the upper limit of input signals,
          "scalelow":digital output value corresponding to the lower limit of input signals,
          "offset":offset,
          "filter1":"filter 1 type","filter2":"filter 2 type"
          "cutoff1":cutoff frequency 1,"cutoff2":cutoff frequency 2
          "avep":moving average points},
           ...,
  "ch12":{}
}
```

Table 3.8 shows the settings for high-speed data acquisition modules.
Enclose a string in "" and specify a number directly.

Table 3.8 Settings for high-speed data acquisition modules (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"HA06" "HA12"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3HA□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"cycle"	5 to 1000		Specifies the data acquisition cycle [microsecond]. The setting value is rounded to the multiple of 5.
"ch1" to "ch12"	--		Sets an object for the channel. Data is collected only for the channel where this key is specified.
"range"	"-10-10v" "0-10v" "1-5v" "-5-5v" "-2.5-2.5v"		Specifies the output range. The default value is "-10-10v".
"scale"	"enable" "disable"		Specifies whether scaling is enabled or disabled. The default value is "disable".
"scaleup"	N (-29000≤N≤30000)		Digital output value that corresponds to the upper limit of input signals. The default value is 0.
"scalelow"	N (-29000≤N≤30000)		Digital output value that corresponds to the lower limit of input signals. The default value is 0.
"offset"	N (-2500≤N≤2500)		Offset value. The default value is 0.
"filter1"	"none" "multi" "average" "lpf_butterworth " "lpf_chebyshev "		Filter type. The default value is "none". none: No filtering multi: Multi-sampling average: Moving average lpf_butterworth: Low-pass (Butterworth) filter lpf_chebyshev: Low-pass (Chebyshev) filter
"filter2"	"none" "lpf_butterworth " "lpf_chebyshev " "hpf_butterworth " "hpf_chebyshev "		Filter type. The default value is "none". none: No filtering lpf_butterworth: Low-pass (Butterworth) filter lpf_chebyshev: Low-pass (Chebyshev) filter hpf_butterworth: High-pass (Butterworth) filter hpf_chebyshev: High-pass (Chebyshev) filter
"cutoff1"	400 to 40000		Specifies the cutoff frequency for filter1.
"cutoff2"	400 to 40000		Specifies the cutoff frequency for filter2.
"avep"	2^n		Specifies the population for multi-sampling or the moving average points. In multi-sampling: 1≤n≤4 In moving average: 1≤n≤11

* Required key

- Setting example

```
{
  "unit":0, "slot":6,"modid":"HA12",
  "cycle":100,
  "ch1":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
    "filter1":"lpf-butterworth","filter2":"none",
    "cutoff1":10000,"cutoff2":0},
  "ch2":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
```

```
      "filter1":"lpf-butterworth","filter2":"none",
      "cutoff1":10000,"cutoff2":0},
    "ch3":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"lpf-butterworth","filter2":"none",
      "cutoff1":10000,"cutoff2":0},
    "ch4":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"lpf-butterworth","filter2":"none",
      "cutoff1":10000,"cutoff2":0},
    "ch5":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"lpf-butterworth","filter2":"none",
      "cutoff1":10000,"cutoff2":0},
    "ch6":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"lpf-butterworth","filter2":"none",
      "cutoff1":10000,"cutoff2":0},
    "ch7":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"},
    "ch8":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"},
    "ch9":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"},
    "ch10":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"},
    "ch11":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"},
    "ch12":{"range":"-10-10v","scaleup":30000, "scalelow":-30000, "offset":0,
      "filter1":"none", "filter2":"none"}
}
```

Table 3.9 shows the digital output values when scaling is disabled.

Table 3.9 Initial scaling settings for high-speed data acquisition modules

Input signal range	Digital output value
-10 to 10 V	-20000 to 20000
0 to 10 V	0 to 20000
1 to 5 V	2000 to 10000
-5 to 5 V	-10000 to 10000
-2.5 to 2.5 V	-5000 to 5000

Note

For details on the high-speed data acquisition module, refer to “High-speed Data Acquisition Module (F3HA06-1R, F3HA12-1R)” (IM 34M06G02-02E).

3.3.6 Temperature monitoring module

The setting format for a temperature monitoring module is shown below.

```
{
  "unit":unit number, "slot":slot number, "modid":"module ID",
  "freq":"output synchronization channel",
  "ch1 ":{"in":"input signal type","rh":upper limit of the measurement range,"rl":lower limit of the
measurement range,
          "sh":scaling upper limit,"sl":scaling lower limit,"sdp":scaling decimal point position},
           ...,
  "ch4":{}
}
```

Table 3.10 shows the settings for the temperature monitoring module.
Enclose a string in "" and specify a number directly.

Table 3.10 Settings for the temperature monitoring module (JSON)

Key (string)	Value	Required*	Remarks
"unit"	0 to 7	Yes	Specifies the position of insertion.
"slot"	1 to 16	Yes	Specifies the position of insertion.
"modid"	"CX04"	Yes	Specifies four (uppercase alphabetic) letters of the model name of an I/O module (F3CX□□), with the string F3 removed. It is used, together with the unit and slot keys, to check the module.
"freq"	"50hz" "60hz"		Specifies the frequency of the power supply. The default value is "50hz".
"ch1" to "ch4"	--		Sets an object for the channel.
"in"	"k-200-1370c" "k-200-1000c" "k-200-500c" "j-200-1200c" "j-200-500c" "t-270-400c" "b0-1600c" "s0-1600c" "r0-1600c" "n-200-1300c" "e-270-1000c" "l-200-900c" "u-200-400c" "w0-1600c" "p0-1390c" "jpt-200-500c" "jpt-200-200c" "jpt0-300c" "jpt0-150c" "pt-200-850c" "pt-200-500c" "pt-200-200c" "pt0-300c" "pt0-150c" "0-10mv" "0-100mv" "0-1v" "0-5v" "1-5v" "0-10v"		Specifies the output range. The default value is "k-200-1370c". Thermocouple K (-200 to 1370) Thermocouple K (-200 to 1000) Thermocouple K (-200 to 500) Thermocouple J (-200 to 1200) Thermocouple J (-200 to 500) Thermocouple T (-270 to 400) Thermocouple B (0 to 1600) Thermocouple S (0 to 1600) Thermocouple R (0 to 1600) Thermocouple N (-200 to 1300) Thermocouple E (-270 to 1000) Thermocouple L (-200 to 900) Thermocouple U (-200 to 400) Thermocouple W (0 to 1600) Thermocouple Platinel (0 to 1390) RTD JPt (-200 to 500) RTD JPt (-200 to 200) RTD JPt (0 to 300) RTD JPt (0 to 150) RTD Pt (-200 to 850) RTD Pt (-200 to 500) RTD Pt (-200 to 200) RTD Pt (0 to 300) RTD Pt (0 to 150) 0 to 10 mV 0 to 100 mV 0 to 1 V 0 to 5 V 1 to 5 V 0 to 10 V

* Required key

- Setting example

```
{
  "unit":0, "slot":7,"modid":"CX04",
  "freq":"50hz",
  "ch1":{"in":"k-200-500c"},
  "ch2":{"in":"k-200-1000c"},
  "ch3":{"in":"k-200-500c"},
  "ch4":{"in":"k-200-1000c"}
}
```

Table 3.11 shows the input ranges and digital output values of the temperature monitoring module.

Table 3.11 Output values of the temperature monitoring module

Input signal range	Digital output value
Thermocouple K (-200 to 1370)	-2000 to 13700
Thermocouple K (-200 to 1000)	-2000 to 10000
Thermocouple K (-200 to 500)	-2000 to 5000
Thermocouple J (-200 to 1200)	-2000 to 12000
Thermocouple J (-200 to 500)	-2000 to 5000
Thermocouple T (-270 to 400)	-2700 to 4000
Thermocouple B (0 to 1600)	0 to 16000
Thermocouple S (0 to 1600)	0 to 16000
Thermocouple R (0 to 1600)	0 to 16000
Thermocouple N (-200 to 1300)	-2000 to 13000
Thermocouple E (-270 to 1000)	-2700 to 10000
Thermocouple L (-200 to 900)	-2000 to 9000
Thermocouple U (-200 to 400)	-2000 to 4000
Thermocouple W (0 to 1600)	0 to 16000
Thermocouple Platinel (0 to 1390)	0 to 13900
RTD JPt (-200 to 500)	-2000 to 5000
RTD JPt (-200 to 200)	-2000 to 2000
RTD JPt (0 to 300)	0 to 3000
RTD JPt (0 to 150)	0 to 15000
RTD Pt (-200 to 850)	-2000 to 8500
RTD Pt (-200 to 500)	-2000 to 5000
RTD Pt (-200 to 200)	-2000 to 2000
RTD Pt (0 to 300)	0 to 3000
RTD Pt (0 to 150)	0 to 15000
0-10mv	0 to 1000
0-100mv	0 to 1000
0-1v	0 to 1000
0-5v	0 to 5000
1-5v	1000 to 5000
0-10v	0 to 1000

Note

For details on the module specifications, refer to “Temperature Monitoring Module” (IM 34M06H63-01E).

4. F3HA12 data acquisition service

This chapter describes the features and usage of the F3HA12 data acquisition service.

4.1 Functional overview

The F3HA12 data acquisition service runs the data acquisition feature of a high-speed data acquisition module (F3HA06/F3HA12) in the background. In general, data acquisition with the high-speed data acquisition module requires monitoring data being accumulated in the module, reading the accumulated data from the module, and keeping on doing the previous steps periodically. The F3HA12 data acquisition service is fully responsible for accessing the high-speed data acquisition module and provides users with the acquired data.

Note

For the details on the module specifications, refer to “High-speed Data Acquisition Module (F3HA06-1R, F3HA12-1R)” (IM 34M06G02-02E).

4.2 Usage

The F3HA12 data acquisition service is provided as a daemon managed by systemd. Systemd is a utility platform for daemon management designed for Linux. Using the systemctl command, you can configure services to start or stop and whether they are run automatically on startup.

4.2.1 Working with the daemon

With the systemctl command, you can work with the ert3dgsd daemon. It performs the following actions on the systemd commands.

Note

The user that has root privilege can use “systemctl” command. When you use it, use “sudo” command or switch to root user with “su” command.

- **Start the data acquisition daemon**

  With the start command, you can start the data acquisition daemon manually. Start the daemon for data acquisition to prepare for it. Start the data acquisition itself.

```
# systemctl start ert3dgsd
```

- **Stop the data acquisition daemon**

  With the stop command, you can stop the daemon.

```
# systemctl stop ert3dgsd
```

- **Restart the data acquisition daemon**

  With the restart command, you can stop and start the daemon. Use this command when you modify the setting file and then reapply it to the I/O module.

```
# systemctl restart ert3dgsd
```

- **Enable or disable the data acquisition daemon on startup**

  With the enable or disable command, you can enable or disable the execution of the daemon on startup. If the daemon is enabled on startup with the enable command, the I/O module is configured when the power is turned on according to the setting file.
  Similarly, the disable command is used to disable the execution of it on startup.

```
# systemctl enable ert3dgsd
```

```
# systemctl disable ert3dgsd
```

- **Check the status of the data acquisition daemon**

  With the status command, you can view the log output from the daemon.

```
# systemctl status ert3dgsd
```

4.2.2 Data acquisition

This subsection provides an overview of data acquisition.
After starting the F3HA12 data acquisition service, you configure F3HA12, start data acquisition, and then obtain the acquired data.

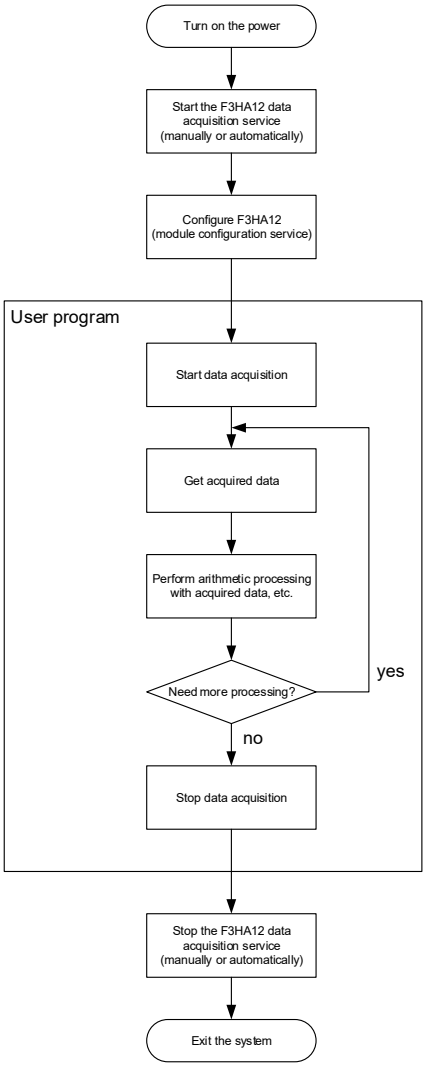


Figure 4.1 Flowchart of data acquisition

The F3HA12 data acquisition service consists of a thread to acquire data from the high-speed data acquisition module and a data server to provide acquired data for the user.

Once started, the service performs the initial operations for the high-speed data acquisition module and the data server and waits for a data acquisition start command from the user.
The user can configure F3HA12 by using the IO module configuration service described in Chapter 3. With this configuration, the user can specify the data acquisition cycle, the channel from which the data is acquired, and analog input settings (such as the range, scale, and whether filters are used).

The user starts or stops data acquisition and obtains the acquired data through the API.
When data acquisition is started, the service accumulates the data in the internal buffer and assigns a data number (1 origin) on a scan basis. A scan is a unit of data acquired by an F3HA□□ module. Acquired data is stored in the internal buffer tightly on a single scan basis. For example, if channels 1, 2, and 6 are active, channel 1 data, channel 2 data, and channel 6 data are stored and then channel 1 data with the next data number is stored. The size of data for one channel is 2 bytes.
The internal buffer is a ring buffer of which size is 100000 scans.

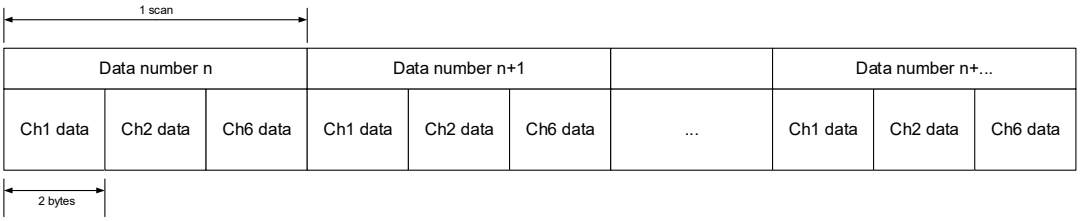


Figure 4.2 Stored data when channels 1, 2, and 6 are enabled

Once data acquisition is stopped, the data number is assigned from 1 when it is started again.

Using the API to obtain acquired data, the user gets the data held by the service from the data server. The acquisition buffer stores the data tightly as shown in the figure above. Use an offset value based on the data number and the number of data acquisition targets to access the necessary data.

Note

For details on the configuration of F3HA12, refer to “3. e-RT3 I/O module configuration service”.

4.3 API

This section shows the information of the API functions as a user interface provided by the F3HA12 data acquisition service.

Table 4.1 API list

Class	Feature	Function name
Management	Initialize API resources	LEDG_open
	Release API resources	LEDG_close
Configuration	Get a data acquisition target	LEDG_getHaGathering
Control	Start data acquisition	LEDG_startHaDataGathering
	Stop data acquisition	LEDG_stopHaDataGathering
Data acquisition	Get the data number of acquired data	LEDG_getHaDataNo
	Get acquired data	LEDG_getHaData

■ Management

● LEDG_open

Feature: Initialize API resources

Format: bool LEDG_open(LEDG_OPEN_MODE mode, int unit, int slot);

Description: This function initializes resources used internally by the API functions.
All the API functions become available by specifying "LEDG_OPEN_MODE_READWRITE" for the "mode" argument.
The API functions for getting configuration and getting acquired data become available by specifying "LEDG_OPEN_MODE_READ" for the "mode" argument.
(The API functions related to configuration change and control are not available.)

Argument	mode	Open mode LEDG_OPEN_MODE_READWRITE: Readable and writable LEDG_OPEN_MODE_READ: Readable
	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	true	Successful
	false	Failed

● LEDG_close

Feature: Release API resources

Format: void LEDG_close(void);

Description: This function releases resources used internally by the API functions.

■ Configuration

● LEDG_getHaGathering

Feature: Get a data acquisition target

Format: unsigned long
LEDG_getHaGathering(bool enableChannels [12], bool* enableCounter);

Description: This function gets whether or not analog input channels and the counter are the data acquisition target (active/inactive).

enableChannels[0]: Whether or not channel 1 is the data acquisition target
enableChannels[1]: Whether or not channel 2 is the data acquisition target
...
enableChannels[11]: Whether or not channel 12 is the data acquisition target
The total number of bytes of data acquisition target data is returned as a return value.

One point of an analog input channel is 2 bytes of data and the counter is 4 bytes.
For example, the return value when the analog input channels for five points are active is 10.

Argument	enableChannels	A pointer to store the Boolean array that indicates whether the acquisition channel is active or inactive. true: Active false: Inactive
	enableCounter	A pointer to store the Boolean variable that indicates whether the counter acquisition is active or inactive. true: Active false: Inactive
Return value	ULONG_MAX	Error
	Other than the above	The number of bytes per scan (0 to 28)

■ Control

● LEDG_ startHaDataGathering

Feature: Start data acquisition

Format: bool LEDG_startHaDataGathering(void);

Description: This function starts data acquisition.
When data acquisition is started, the function accumulates the data in the internal buffer and assigns a data number (1 origin) on a scan basis. A scan is a unit of data acquired by an F3HA□□ module. The size of data per scan varies depending on the number of data acquisition targets.
Once data acquisition is stopped, the data number is assigned from 1 when it is started again.

Return value	true	Successful
	false	Failed

● LEDG_ stopHaDataGathering

Feature: Stop data acquisition

Format: bool LEDG_stopHaDataGathering(void);

Description: This function stops data acquisition.

■ Data acquisition

● LEDG_getHaDataNo

Feature: Get the data number of acquired data

Format:
```
bool LEDG_getHaDataNo (long long* oldestNo, long long* newestNo,
                       HA_ERR_STS* acqLastErr);
```

Description: This function gets the data number (1 origin) of the data being acquired.
It gets the oldest and latest data numbers of data held by the service when the API function is called.

Argument	oldestNo	The oldest data number of the data to be acquired. If the data does not exist, -1 is returned. (If it is unnecessary, NULL is passed.)
	newestNo	The latest data number of the data to be acquired. If the data does not exist, -1 is returned. (If it is unnecessary, NULL is passed.)
	acqLastErr	Final data acquisition error status
Return value	true	Successful
	false	Failed

● LEDG_getHaData

Feature: Get acquired data

Format:
```
bool LEDG_getHaData (long long reqFromNo, long long reqToNo,
                     unsigned char* buf, unsigned long numOfBuff,
                     long long* realFromNo, long long* realToNo,
                     HA_ERR_STS* acqLastErr);
```

Description:
This function gets acquired data from reqFromNo to reqToNo in the buffer.
You need to ensure that the data acquisition buffer has space larger than the size obtained by multiplying the number of scans acquired by LEDG_getHaGathering by numOfBuff.
Acquired data is stored tightly on a single scan basis. For example, if channels 1, 2, and 6 are active, channel 1 data with the realFromNo number is stored in the 0th byte of the offset in the buffer, channel 2 data with the same number in the 2nd byte of the offset, channel 6 data with the same number in the 4th byte of the offset, and then channel 1 data with the next data number in the 8th byte of the offset, and so on.
The data numbers actually acquired are stored in realFromNo and realToNo depending on the number of buffers and the status of the data held by the service.

Argument	reqFromNo	The start number of data to be requested. If the specified data number does not exist, the data from the oldest is returned.
	reqToNo	The last number of data to be requested. If the specified data number does not exist, the data up to the latest is returned. (The data up to the latest is returned if LLONG_MAX is specified.)
	buf	Buffer for data acquisition
	numOfBuff	The number of buffers ready
	realFromNo	The start data number of the data actually acquired is returned.
	realToNo	The latest data number of the data actually acquired is returned.
	acqLastErr	Final data acquisition error status
Return value	ULONG_MAX	Error
	Other than the above	The number of scans of acquired data

5. Application development with Python

This chapter describes how to create a development environment of Python applications with Visual Studio Code and Jupyter Notebook.

Visual Studio Code is a free source code editor with development tools of code completion, debugging, and more. It allows you to easily add or remove source files on F3RP70-2L and edit various settings such as debug configurations, making it possible to carry out flexible development.

Jupyter Notebook is an open source application in which you can view, run, and edit document files called a notebook through web browser.
It provides features, such as stepwise execution in the unit of operations called a cell and easy drawing of graphs, helping you develop your applications quickly.

5.1 Development method

Figure 5.1 shows the configuration of the development environment for Python applications.

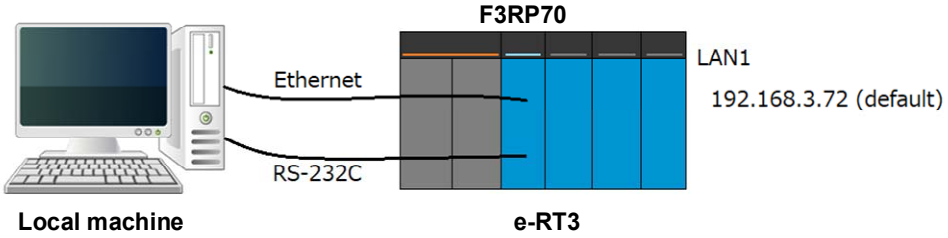


Figure 5.1 Configuration of the application development environment

In the development of your application, the local machine communicates with e-RT3 through a browser or SSH connection. Therefore, you need to connect your machine to F3RP70-2L with an Ethernet cable.

The COM port on F3RP70-2L is used for the Linux console. Using Linux shell commands, you can create files, modify F3RP70-2L settings, check the operating status, and more.
Table 5.1 lists serial setting of F3RP70-2L

Table 5.1 Serial setting of F3RP70-2L

Item	value
Baud rate	115,200bps
Data length	8bit
Stop bit	1bit
Parity bit	None
Flow control	none

5.2 Remote development with Visual Studio Code

5.2.1 Overview

Visual Studio Code is a free source code editor with development tools of code completion, debugging, and more. It allows you to easily add or remove source files on F3RP70-2L and edit various settings such as debug configurations, making it possible to carry out flexible development.

This section describes how to create the remote development environment with Visual Studio Code and how to use it. In application development, you (1) upload source code, (2) run a program on F3RP70-2L, and then (3) attach to a process and debug it. The details of each step are as follows:

(1) Upload source code
Using an SFTP extension of Visual Studio Code, upload source code on the local machine to F3RP70-2L.

(2) Run the program
Connect to F3RP70-2L from Visual Studio Code through an SSH connection and run the program on F3RP70-2L.

(3) Attach to a process and debug it
Attach to the process that is running on F3RP70-2L from Visual Studio Code and perform debug on the local machine.

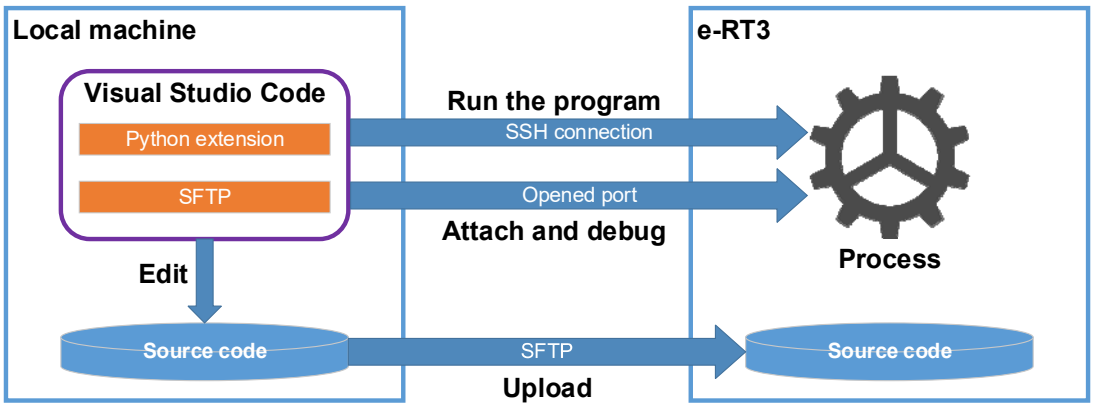


Figure5.2 Remote development with Visual Studio Code

5.2.2 Environment creation procedure

- **Installing Visual Studio Code**

  Install Visual Studio Code in your local machine.

1. Access the following URL to download Visual Studio Code:
https://code.visualstudio.com/

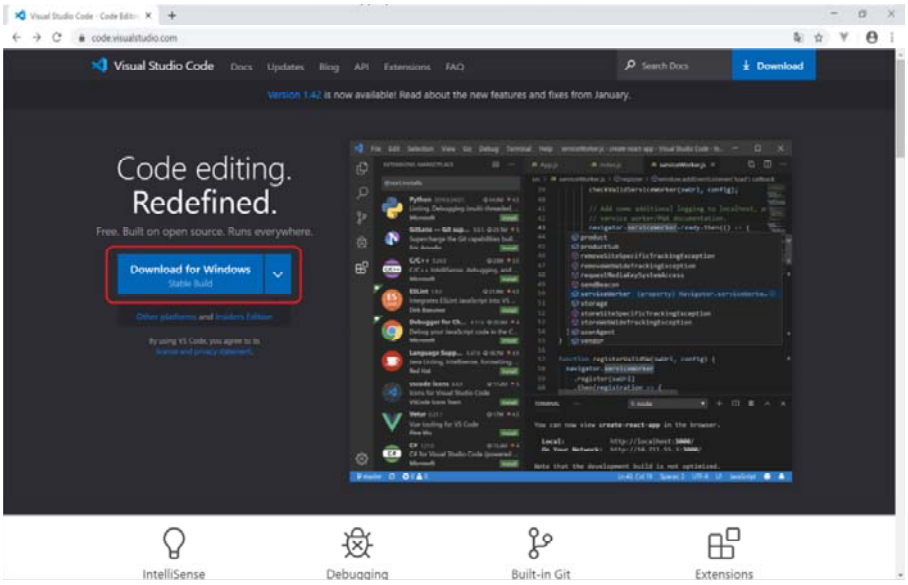


Figure 5.3 Installation of Visual Studio code -1

2. Installing Visual Studio Code

Run the exe file you downloaded to install Visual Studio Code.
Select [I accept the agreement] and then click [Next].

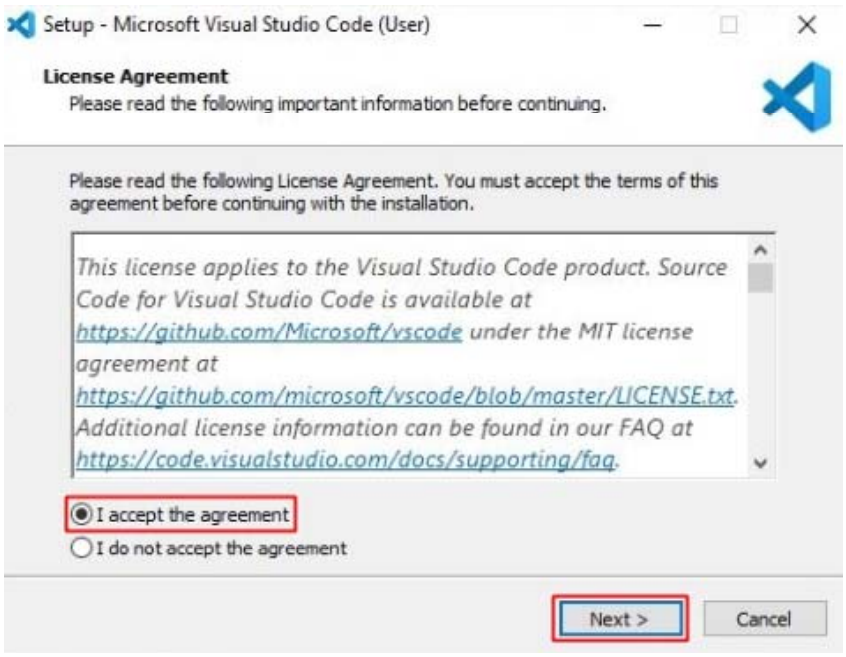


Figure 5.4 Installation of Visual Studio code -2

Click [Next].

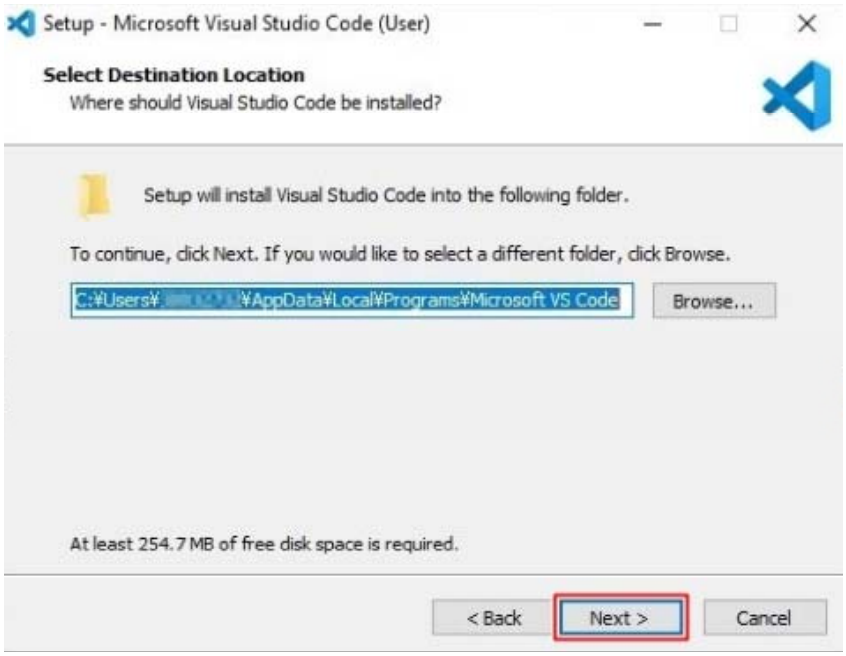


Figure 5.5 Installation of Visual Studio code -3

Click [Next].

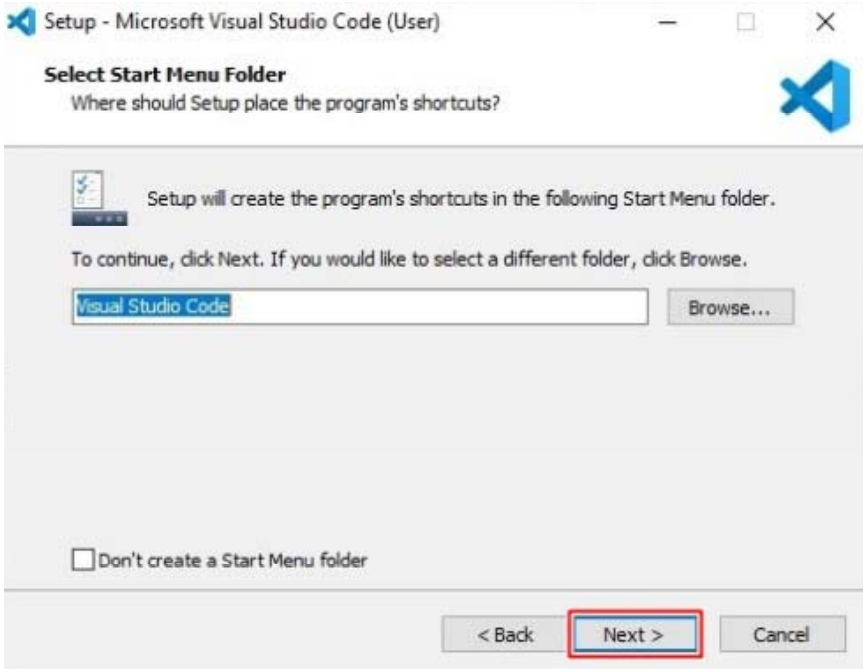


Figure 5.6 Installation of Visual Studio code -4

Select the [Create a desktop icon] check box and then click [Next].

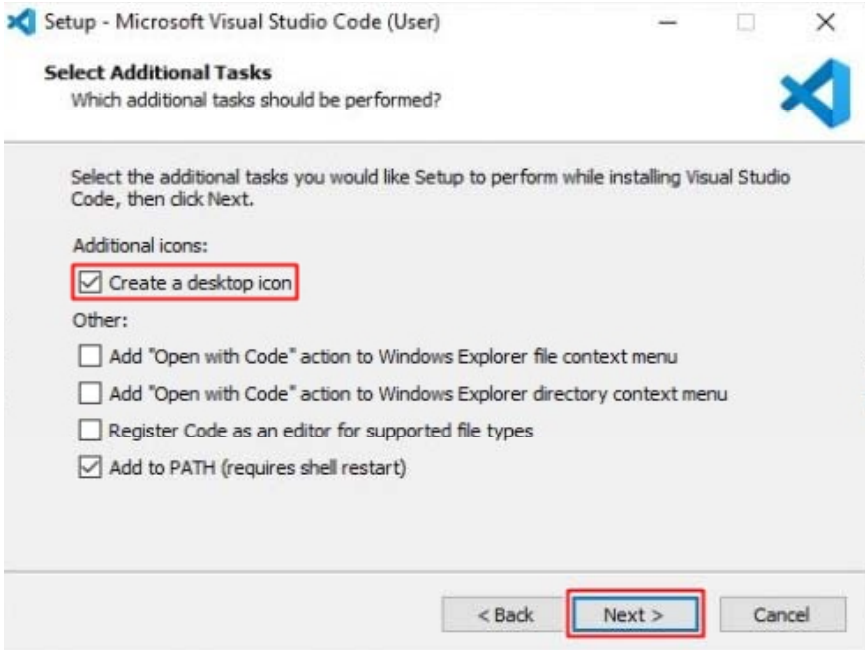


Figure 5.7 Installation of Visual Studio code -5

Click [Install].

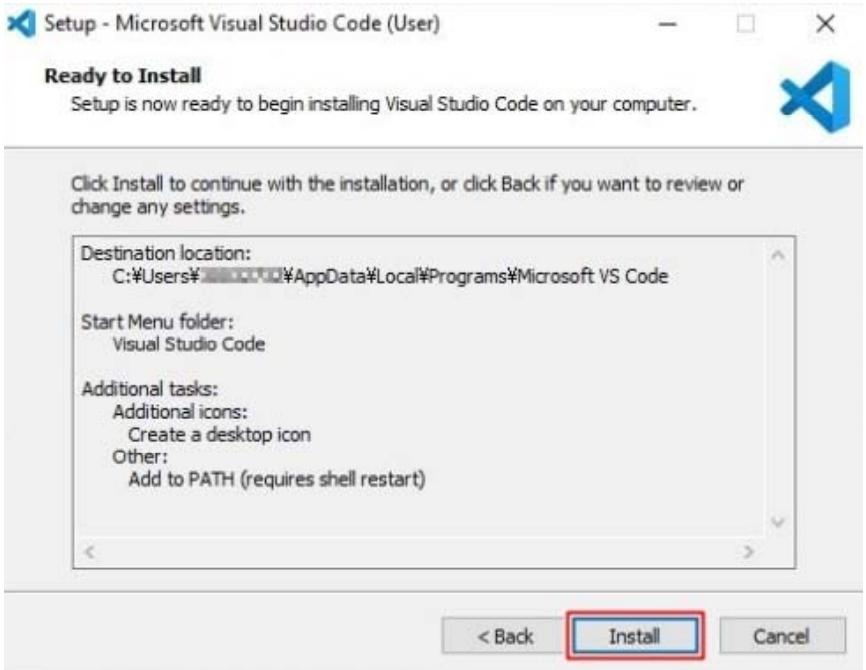


Figure 5.8 Installation of Visual Studio code -6

Click [Finish] to finish the installation.

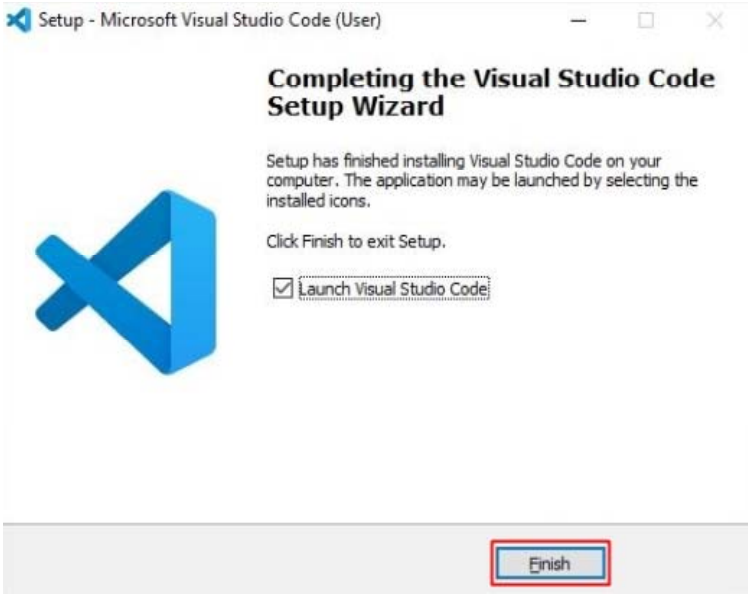


Figure 5.9 Installation of Visual Studio code -7

● Installing the extensions

Install two extensions: the Python extension, which is used to debug Python applications on the local machine, and the SFTP extension, which is used to upload source code on the local machine to F3RP70-2L.

From the menu on the left, select the [Extensions] icon and type “python” in the search field. From the search results, select [Python] and install it.

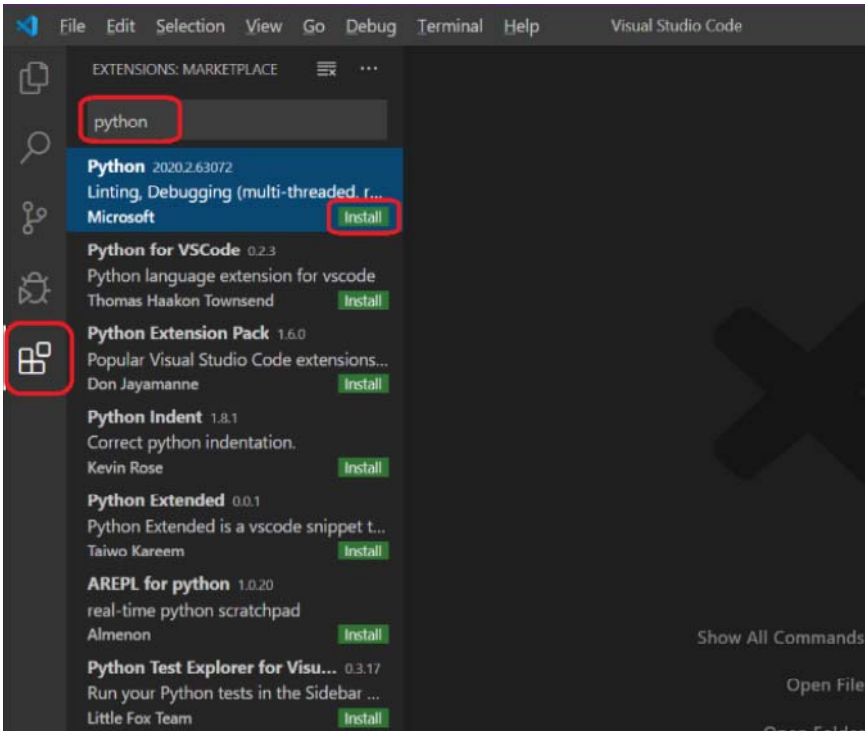


Figure 5.10 Installation of extension -1

Similarly, type “sftp” in the search field and install the SFTP extension.

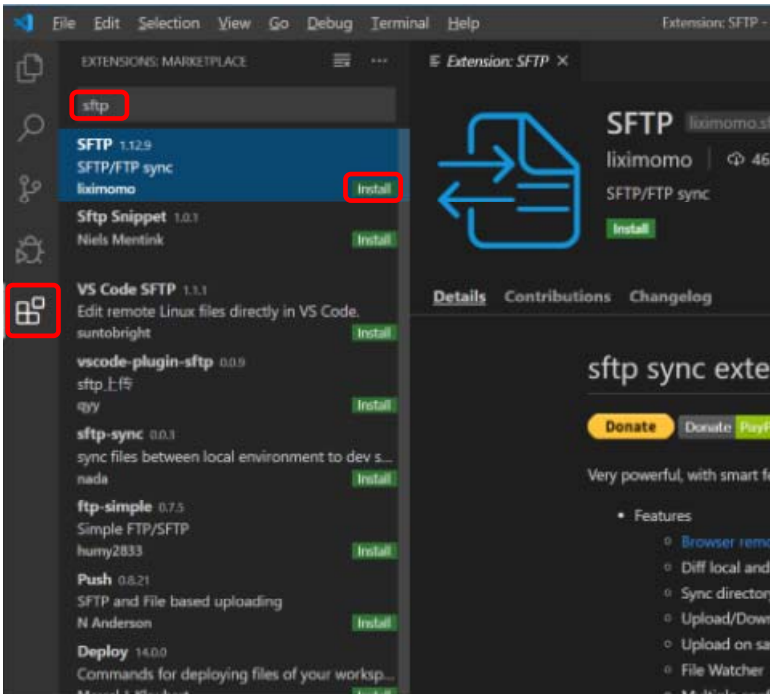


Figure 5.11 Installation of extension -2

After the installation is complete, restart Visual Studio Code for the settings to take effect.

● Creating a workspace folder

1. Creating a workspace folder on F3RP70-2L.

Create workspace folder on F3RP70-2L.

Starts the Visual Studio Code, then Click [Terminal] - [New Terminal] and open a terminal.

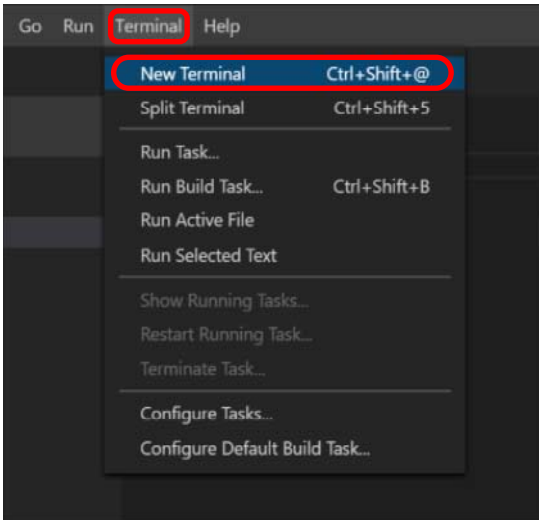


Figure 5.12 Creating workspace folder -1

Terminal is shown in bottom pane.

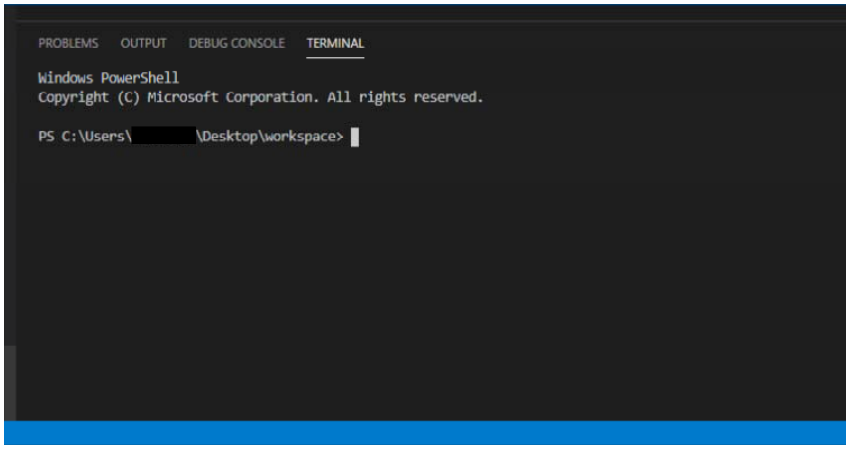


Figure 5.13 Creating workspace folder -2

Connect to F3RP70-2L as “ert3” user.

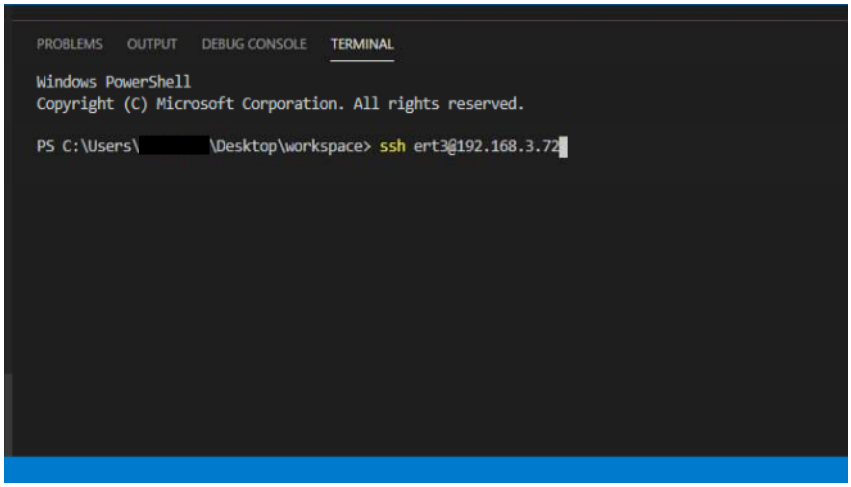


Figure 5.14 Creating workspace folder -3

When you connect to F3RP70-2L at first time, following message is displayed. Input “Yes” and press “Enter” key.

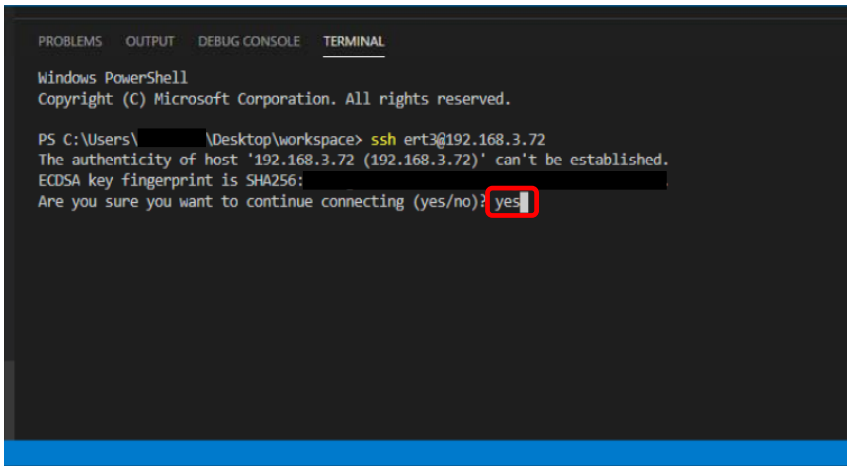


Figure 5.15 Creating workspace folder -4

Input password of “ert3” user, and log in.

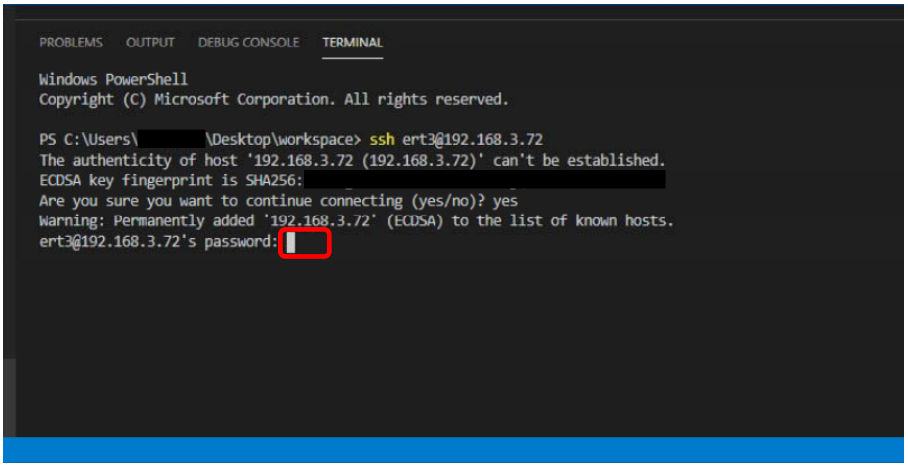


Figure 5.16 Creating workspace folder -5

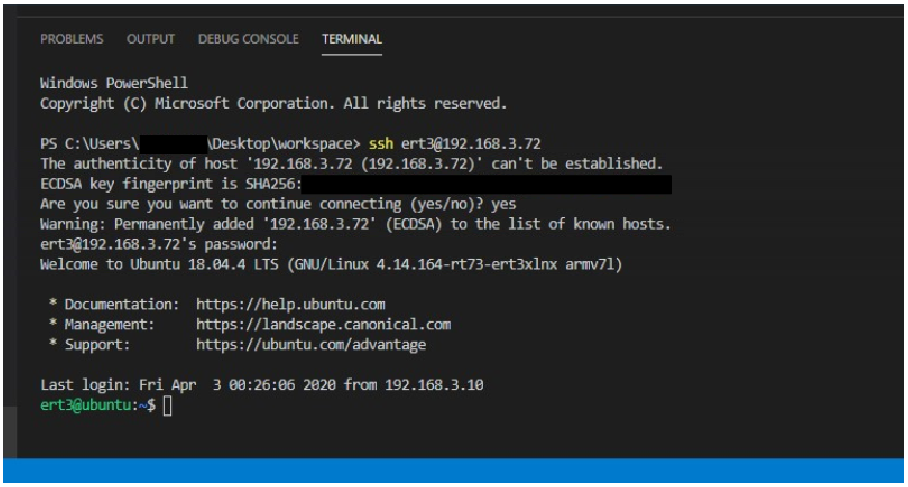


Figure 5.17 Creating workspace folder -6

Create workspace folder in any directory. In this manual as an example, input below command and create workspace folder in home directory of “ert3”.

```
$ mkdir /home/ert3/workspace
```

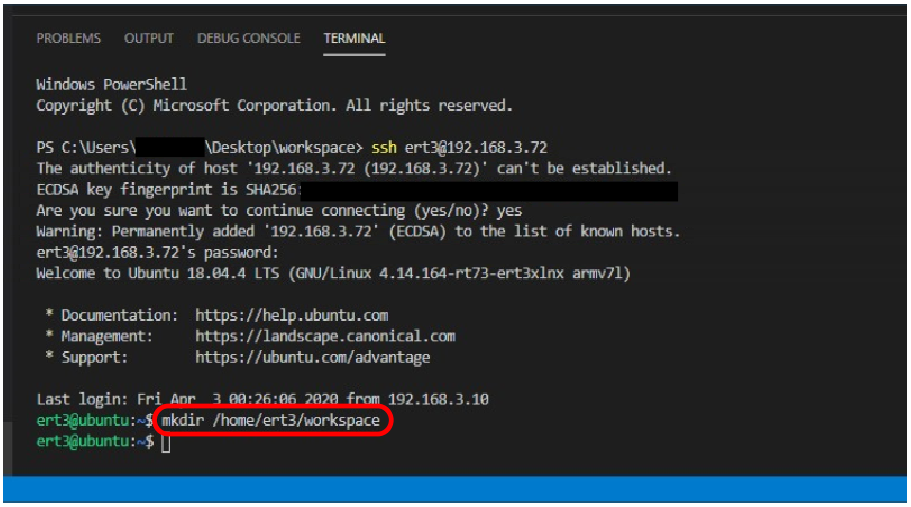


Figure 5.18 Creating workspace folder -7

2. Creating a workspace folder on local machine.

Create workspace folder in any directory of local machine. In this manual as an example, create workspace folder in desktop.

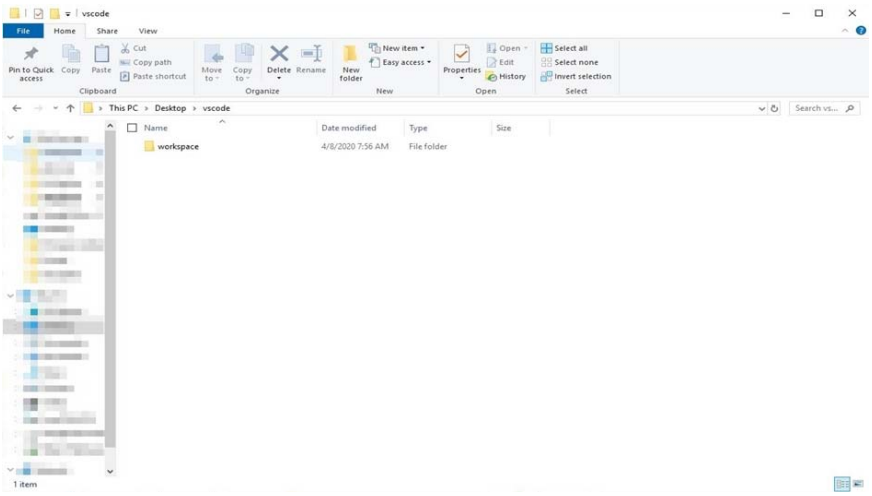


Figure 5.19 Creating workspace folder -8

● Configuring SFTP

Configure the settings for uploading source code on the local machine to e-RT3.

1. Open the workspace folder to Visual Studio Code.

Click [File] - [Add Folder to Workspace].

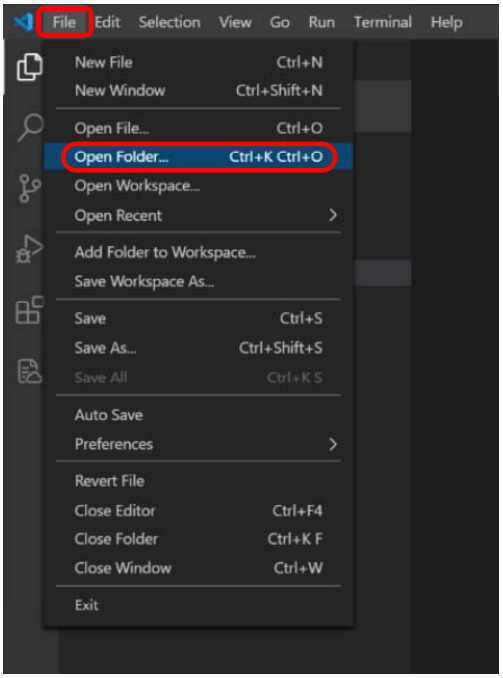


Figure 5.20 Setting of SFTP -1

Open the workspace folder you created on the local machine.

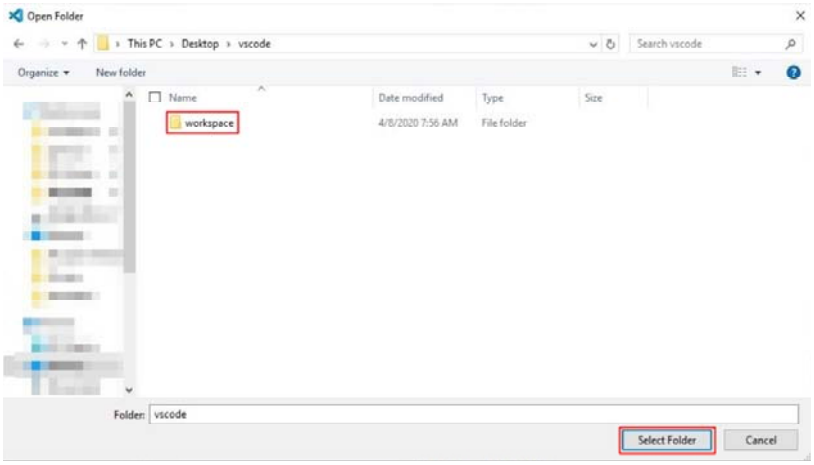


Figure 5.21 Setting of SFTP -2

2. Configure SFTP.

Click [View] - [Command Palette] to open the command palette. In the search field, type “sftp config” and then click [SFTP: Config].

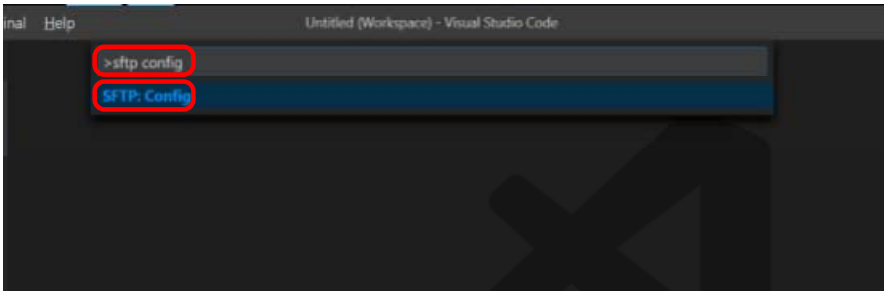


Figure 5.21 Setting of SFTP -3

Fill out the following items and save the settings.

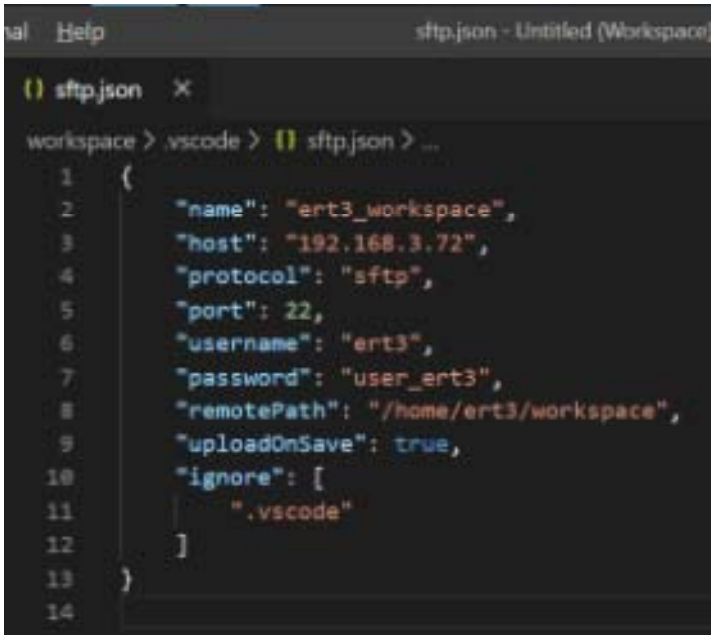


Figure 5.23 Setting of SFTP -4

The descriptions of the items and values are as follows:

- name
  Specifies the connection name displayed on Visual Studio Code.

- host
  Specifies the IP address of F3RP70-2L.

- protocol
  Specifies the protocol of file transferring.

- port
  Specifies the port number of F3RP70-2L used for file transferring.

- username
  Specifies the user name of F3RP70-2L.

- password
  Specifies the password of F3RP70-2L user.

- remote Path
  Specifies a workspace folder of F3RP70-2L.

- uploadOnSave
  Specify whether to automatically upload when saving the file.

- ignore
  Specify a file and folder not to upload.

In the menu on the left, click the [SFTP] icon and check the folder in e-RT3. Check that “ert3_workspace” is displayed.

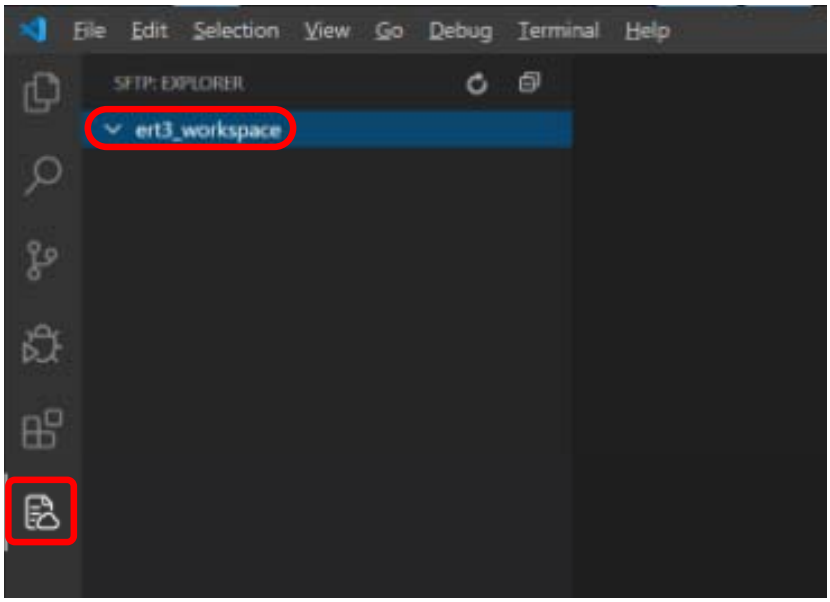


Figure 5.24 Setting of SFTP -5

● Configuring Launch

Edit “Launch.json” to set up the debug configuration.

Select the “.vscode” folder and click the [New File] button to create the “launch.json” file.

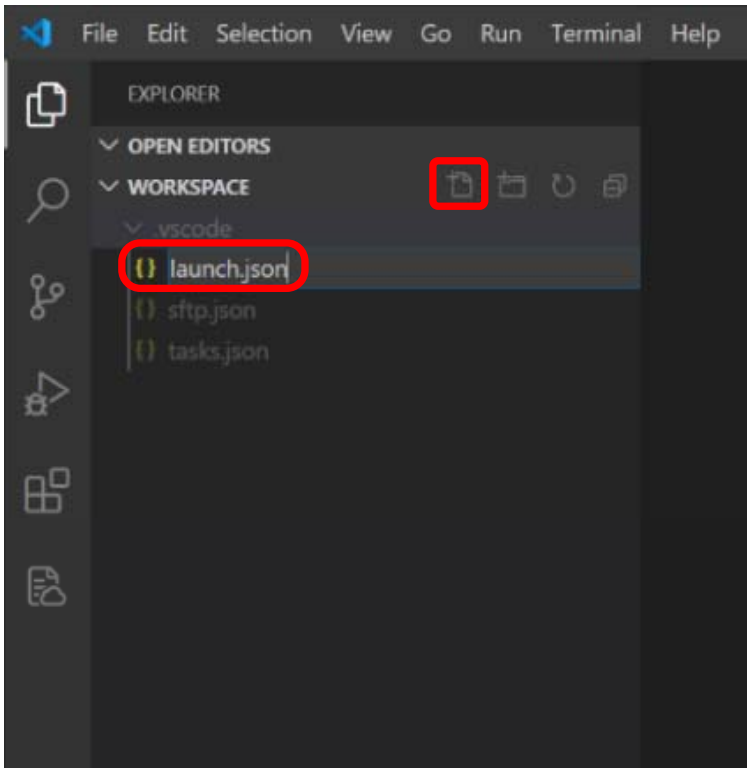


Figure 5.25 Setting of launch -1

Fill out the following items and save the settings.

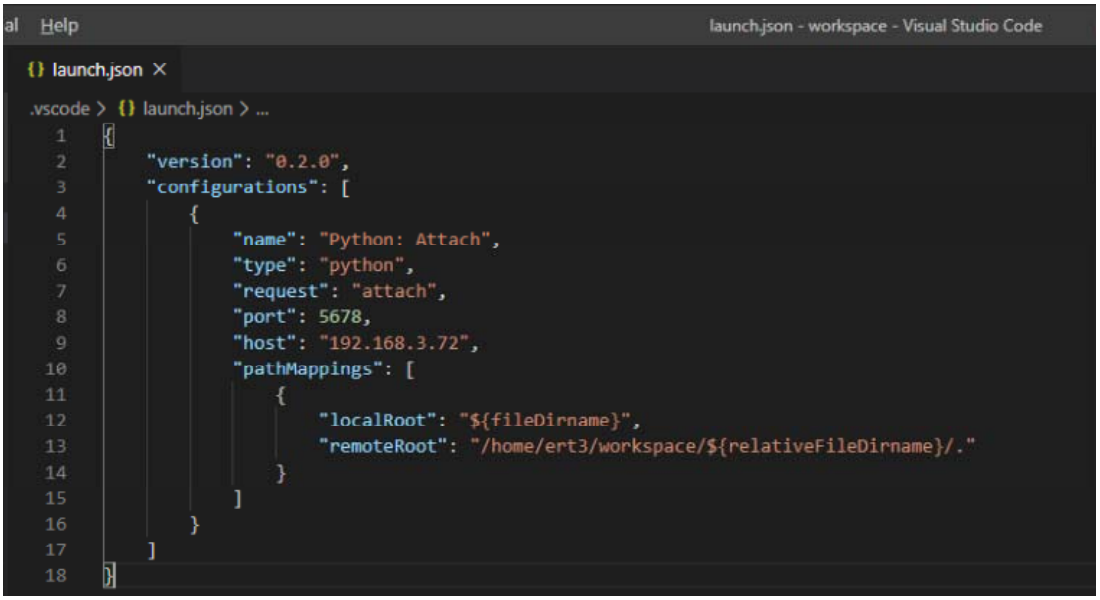


Figure 5.26 Setting of Launch -2

The descriptions of the items and values are as follows:

- name
  Specifies the name of setting.

- type
  Specifies the type of setting.

- request
  Specifies the request of setting.

- port
  Specifies the port number of F3RP70-2L used for communication.

- host
  Specifies the IP address of F3RP70-2L.

- localRoot
  Specifies the workspace of the local machine. ${fileDirname} is a variable that indicates the path to the file currently open in the editor.

- remoteRoot
  Specifies the workspace of F3RP70-2L. ${relativeFileDirname} is a variable that indicates the relative path to the directory of the file currently open in the editor. The starting point of the relative path is the workspace folder of the local machine.

5.2.3 Usage

- **Creating the project folder and source file**

Select [workspace] and click the [New Folder] button to create the “project” folder.

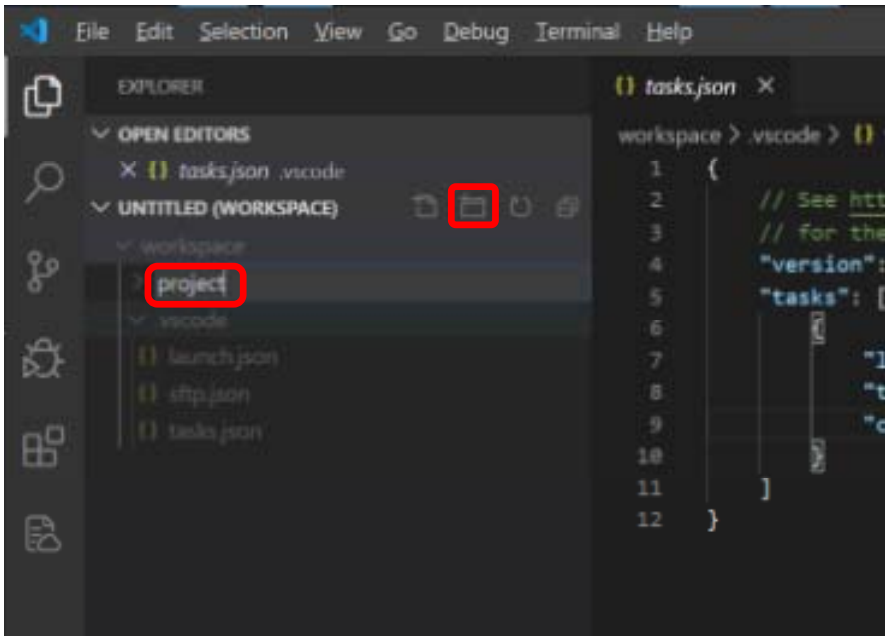


Figure 5.27 Creating project folder and source file -1

Select the “project” folder and click the [New File] button to create the “test.py” file.

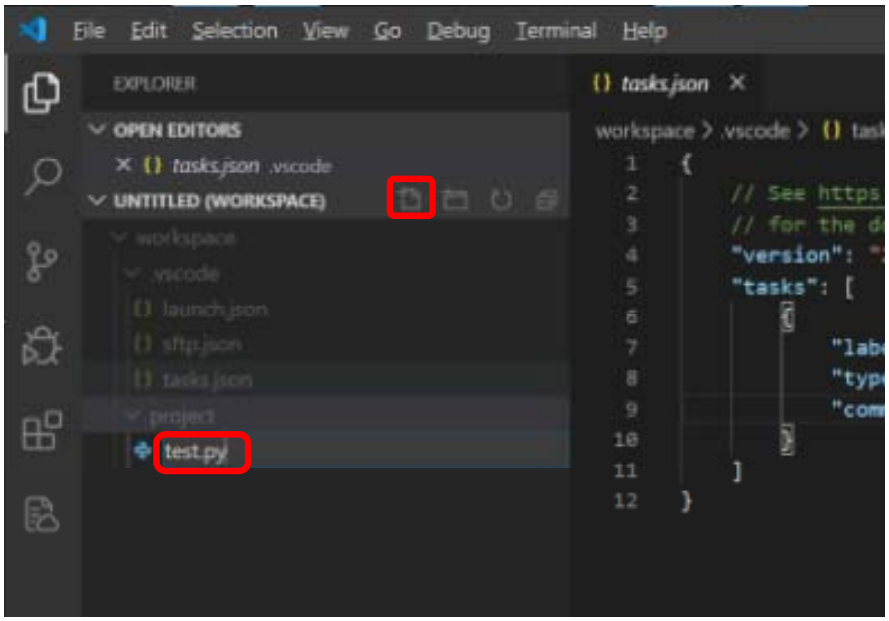


Figure 5.28 Creating project folder and source file -2

Note

Place the source file directly under the project folder.

Creating and uploading a program

1. Create a program.

Open “test.py” and write the following source code in it.

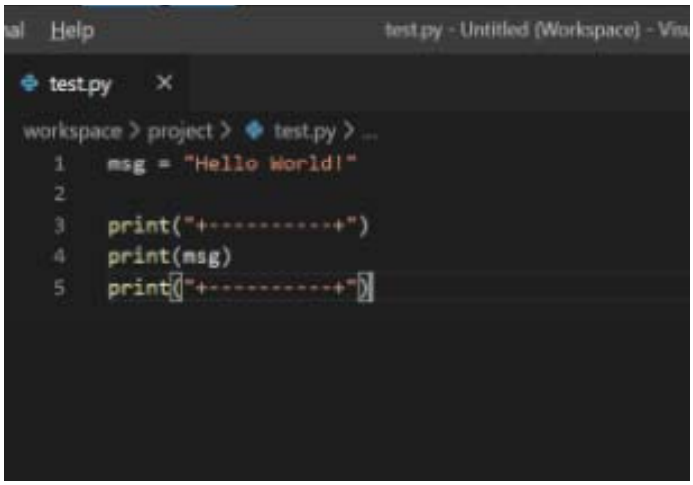


Figure 5.29 Creating and uploading a program -1

2. Upload the source code.
Right-click the “project” folder and click [Sync Local -> Remote] to upload the source code to the workspace of F3RP70-2L.

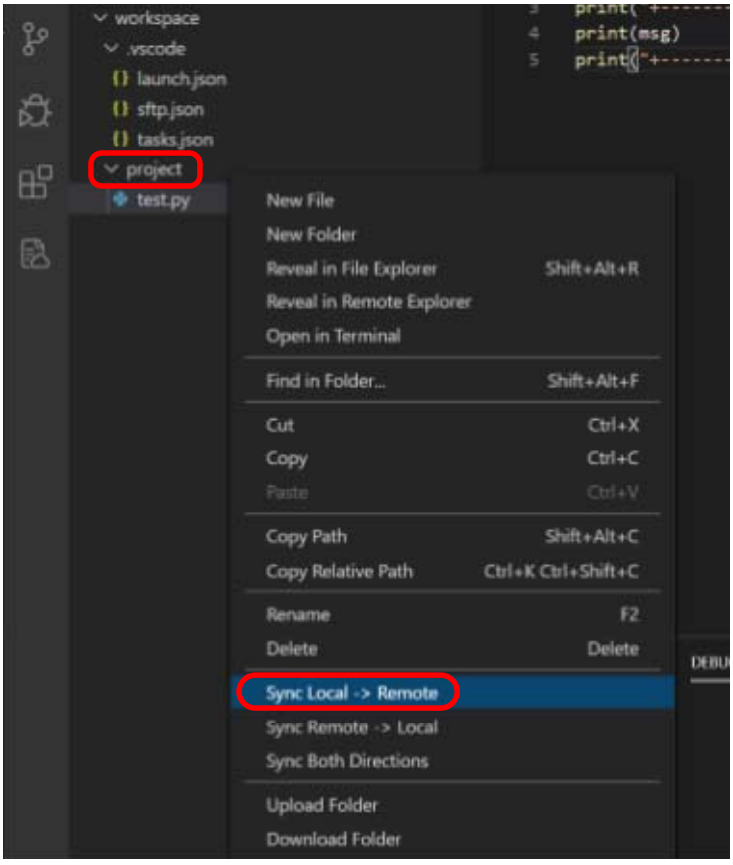


Figure 5.30 Creating and uploading a program -2

Click the [SFTP] icon, select the “ert3_workspace” folder, and click the [Refresh] button. Check that the source code is added to the workspace of F3RP70-2L.

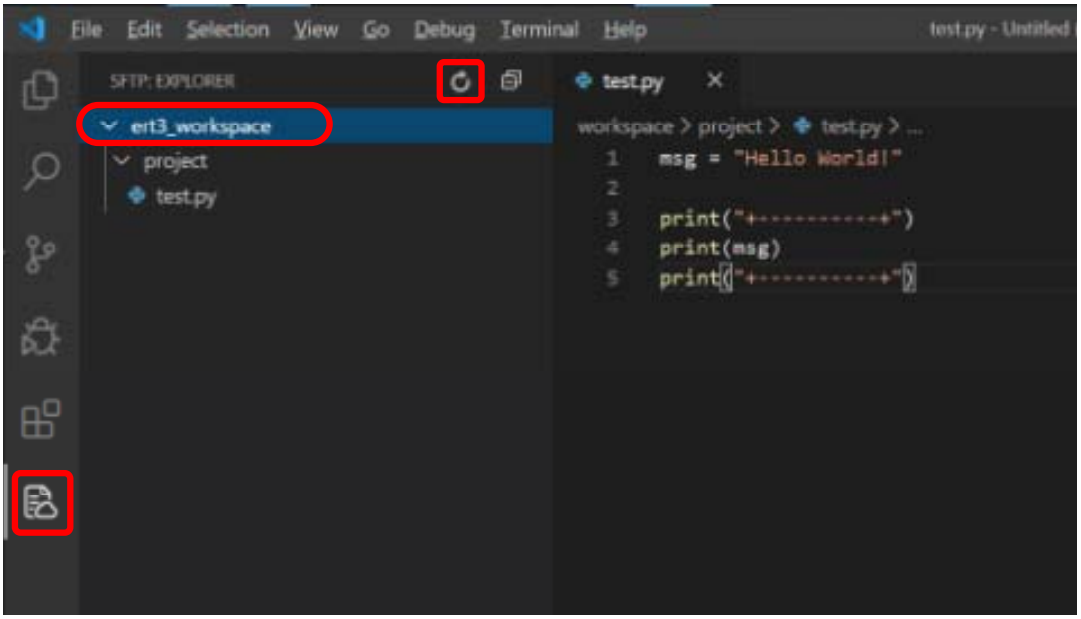


Figure 5.31 Creating and uploading a program -3

● Running and debugging the program

1. Running the program on F3RP70-2L.

Connect to F3RP70-2L with SSH, execute the program, and wait for attachment from the debugger on the local machine.

Click [Terminal] – [New Terminal], open the terminal and connect to F3RP70-2L as “ert3” user using SSH.

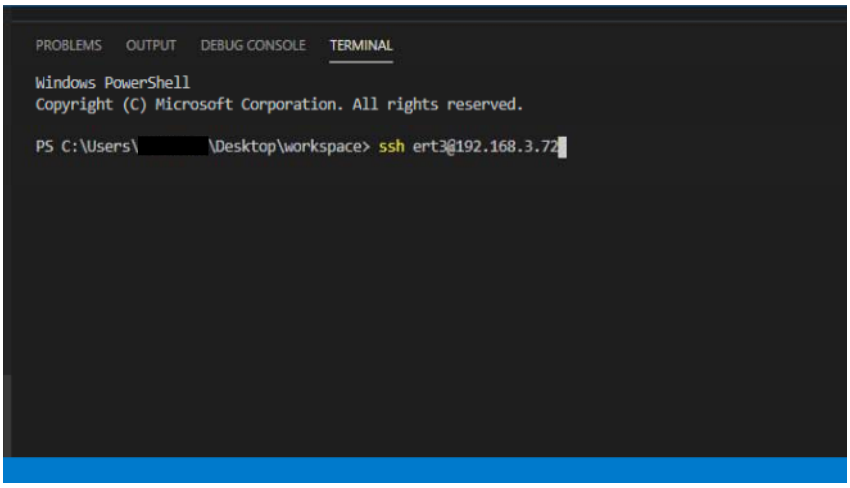


Figure 5.32 Running and debugging the program -1

Move to the directory containing the source file “test.py” to execute.

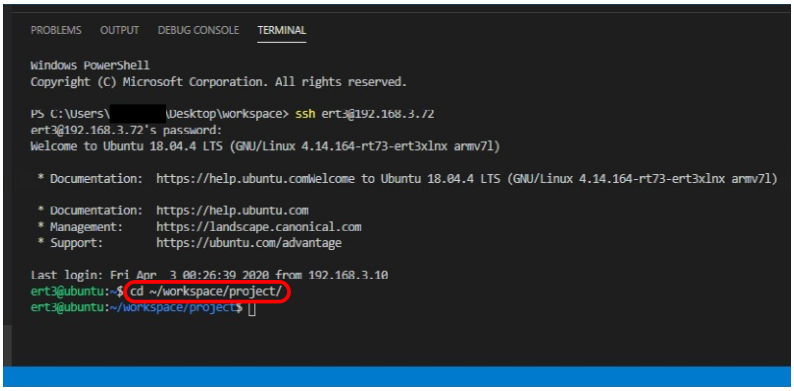


Figure 5.33 Running and debugging the program -2

Input the command below and execute “test.py” and wait for attachment from the debugger on the local machine.

```
$ python3 -m ptvsd --host 0.0.0.0 --port 5678 --wait test.py
```

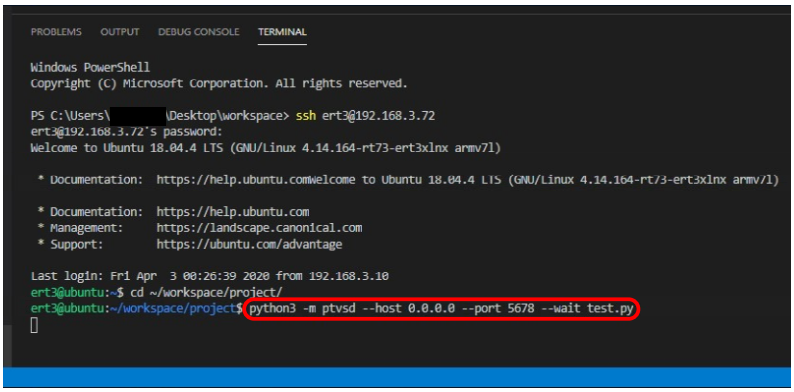


Figure 5.34 Running and debugging the program -3

2. Attaching from debugger of local machine.

Attach the process running on F3RP70-2L from debugger of local machine.
Open the “test.py” source file on the local machine and put a breakpoint at any place.
Click the [Debug] icon and then click the [Start Debugging] button.

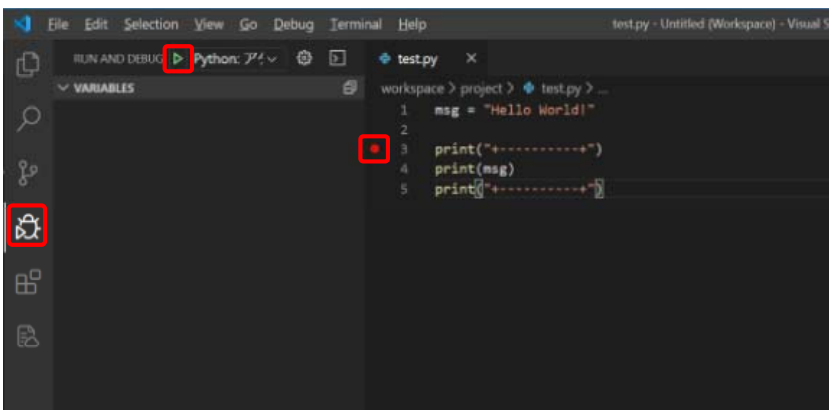


Figure 5.35 Running and debugging the program -4

The program stops at the breakpoint after attaching the program running on F3RP70-2L.

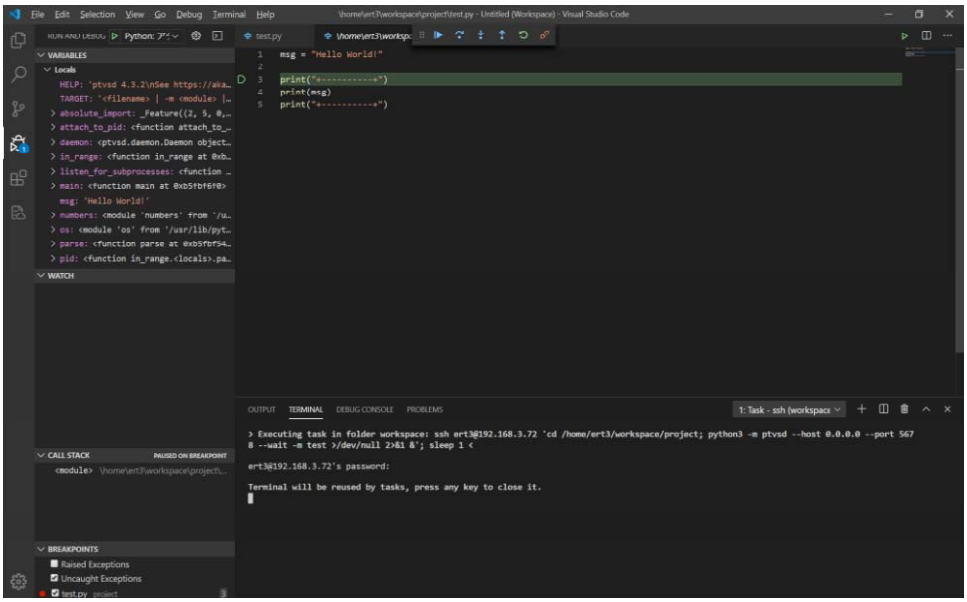


Figure 5.36 Running and debugging the program -5

You can see the output in the terminal.

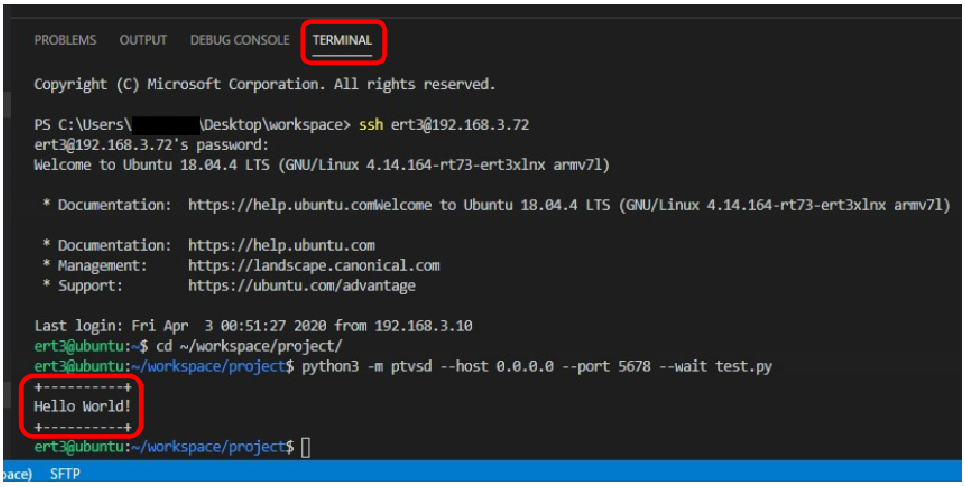


Figure 5.37 Running and debugging the program -6

● Removing the file

Remove the file on F3RP70-2L and local machine.

- Removing the file from local machin
  Click the file icon to view the file on local machine. And then right click on the file and click [Delete].

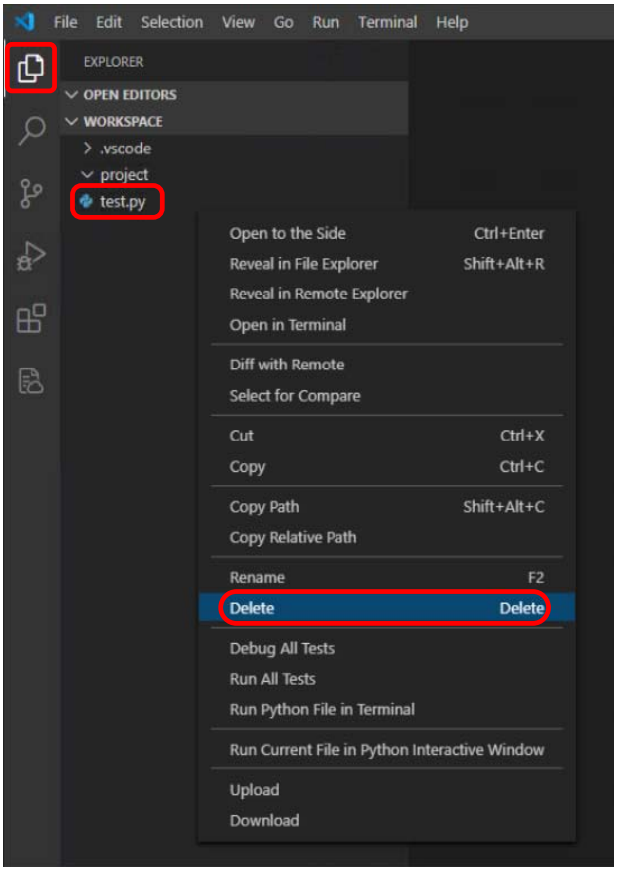


Figure 5.38 Removing the file -1

・ Removing the file from F3RP70-2L
Click the file icon to view the file on F3RP70-2L. And then right click on the file and click [Delete].

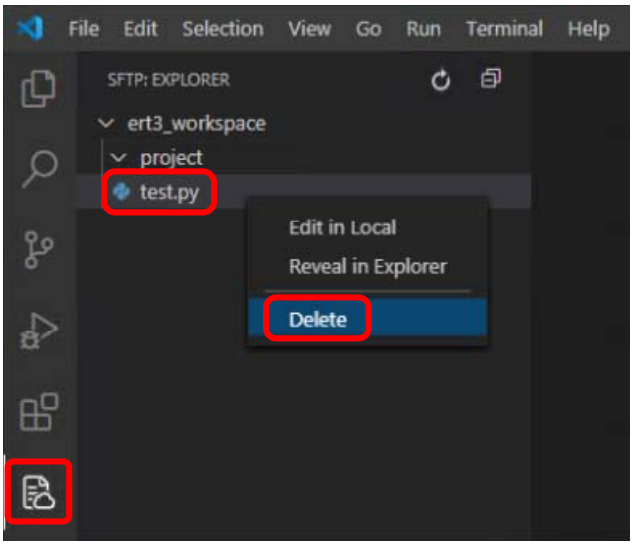


Figure 5.39 Removing the file -2

5.3 Remote development with Jupyter Notebook

5.3.1 Overview

Jupyter Notebook is an open source application in which you can view, run, and edit document files called a notebook.
It provides features, such as stepwise execution in the unit of operations called a cell and easy drawing of graphs, helping you develop your applications quickly.

This section describes how to configure and start a Jupyter Notebook server on F3RP70-2L and how to access the Jupyter Notebook server from the local machine through a Web browser. The user edits the notebook in F3RP70-2L through the Web browser on the local machine for development.

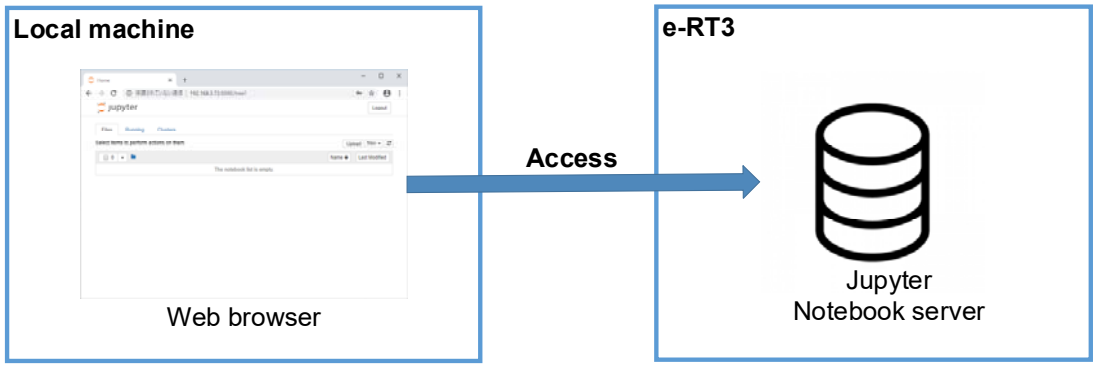


Figure 5.40 Remote development with Jupyter Notebook

5.3.2 Environment creation procedure

Configure Jupyter Notebook that has been installed in e-RT3.

- **Configuring Jupyter Notebook**

  Create a configuration file.
  Example:

```
ert3@ubuntu:~$ jupyter notebook --generate-config
```

  Configure the password required when you access Jupyter Notebook from the local machine.
  Example:

```
ert3@ubuntu:~$ jupyter notebook password
Enter password:          //Enter the password.
Verify password:         //Enter the password again.
```

5.3.3 Usage

- **Starting Jupyter Notebook and accessing it from a Web browser**

1. Start Jupyter Notebook.

Example:

```
ert3@ubuntu:~$ jupyter notebook --ip='*' --no-browser
```

2. Access the following URL in the Web browser of the local machine:
https://192.168.3.72:8888/

Note

Secure communication can be performed using SSL. Please refer to the Jupyter Notebook official document for details.
https://jupyter-notebook.readthedocs.io/en/stable/public_server.html

A login screen appears.
Enter the password you specified to log in.

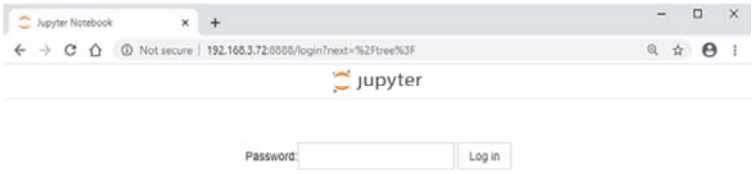


Figure 5.41 Starting Jupyter Notebook and access from web browser -1

Note

When you can't log in, try to delete your browser cache.

The following screen appears.

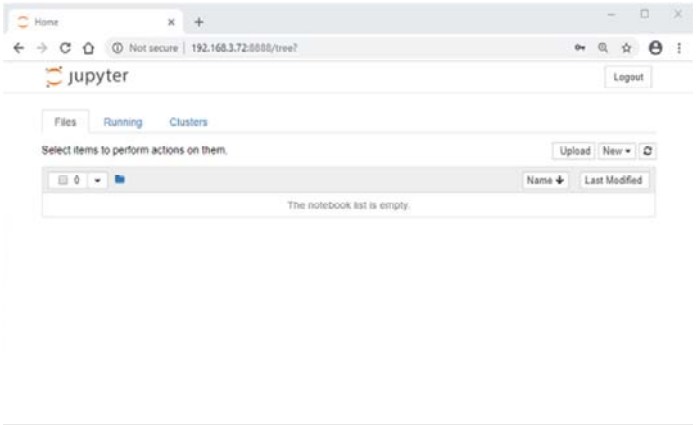


Figure 5.42 Starting Jupyter Notebook and access from web browser -2

● Creating a folder

Click the [New] button to show the drop-down list and click [Folder].

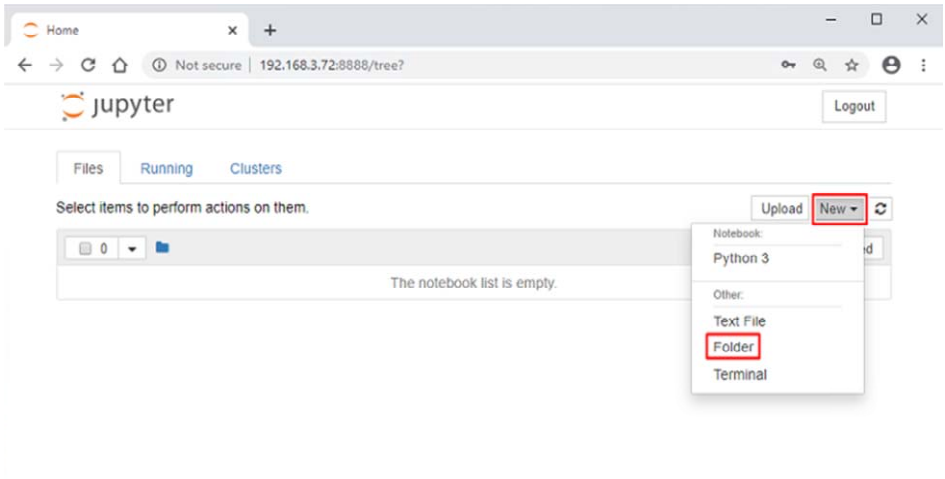


Figure 5.43 Creating folder -1

Select the check box for [Untitled Folder] you created and then click [Rename] displayed above it.

Figure 5.44 Creating folder -2

Type a given folder name and click the [Rename] button.

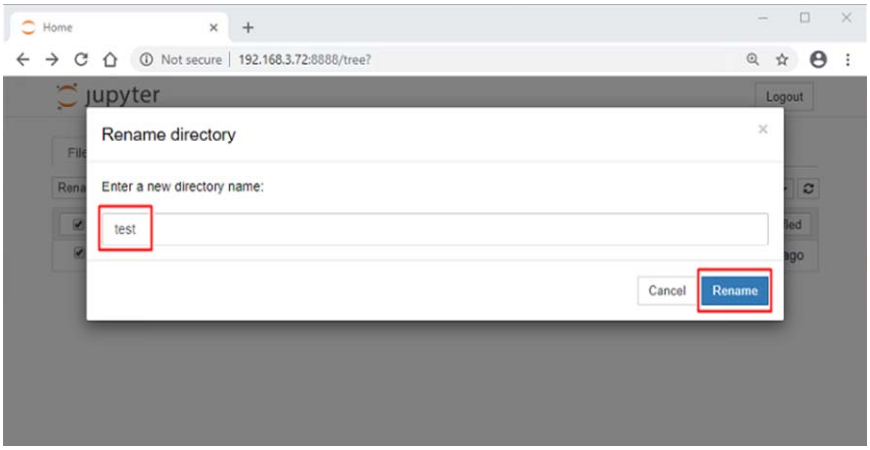


Figure 5.45 Creating folder -3

● Creating a notebook

Click the [New] button to show the drop-down list and click [Python3].

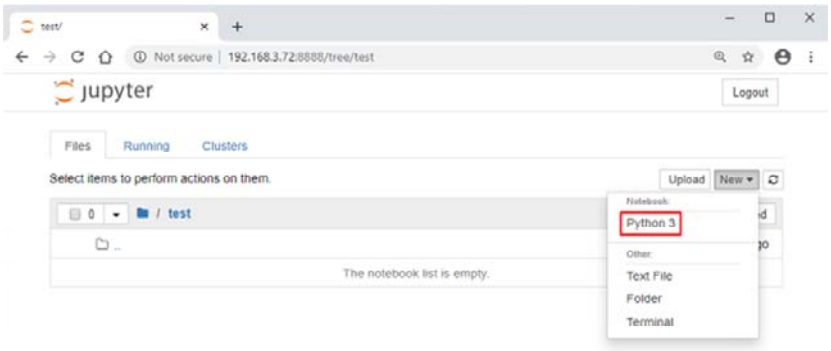


Figure 5.46 Creating notebook -1

A new notebook appears in the browser.

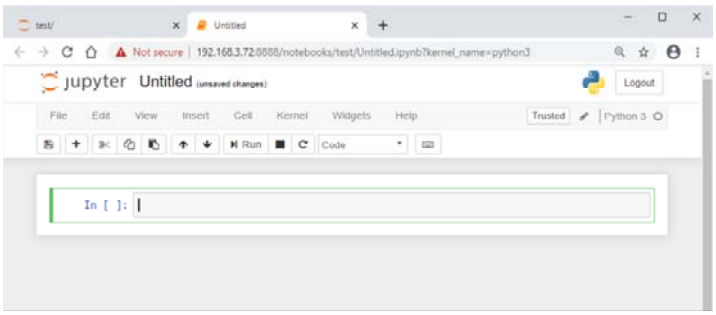


Figure 5.42 Creating notebook -2

Coding

In this subsection as an example, you create a program that shows the elements of a list and draws a scatter diagram.

Import the “matplotlib” package, which is required to draw the scatter diagram.

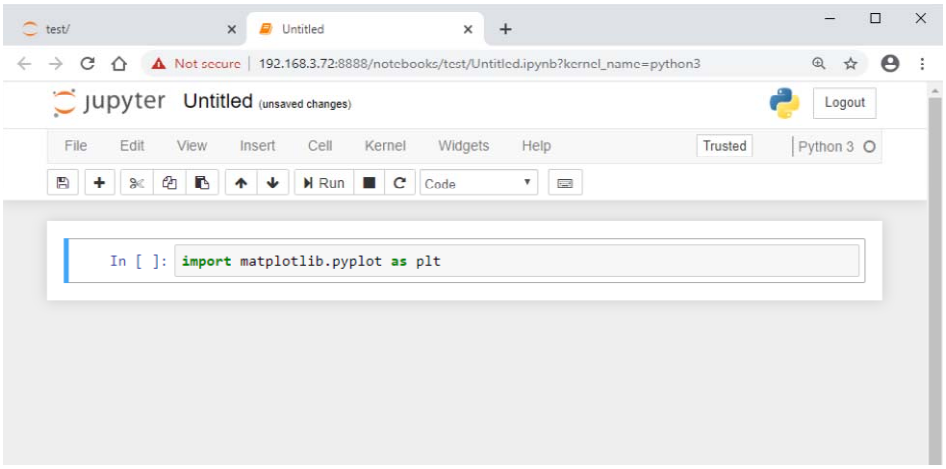


Figure 5.48 Coding -1

Click [insert cell below] and write code to declare and show a list.

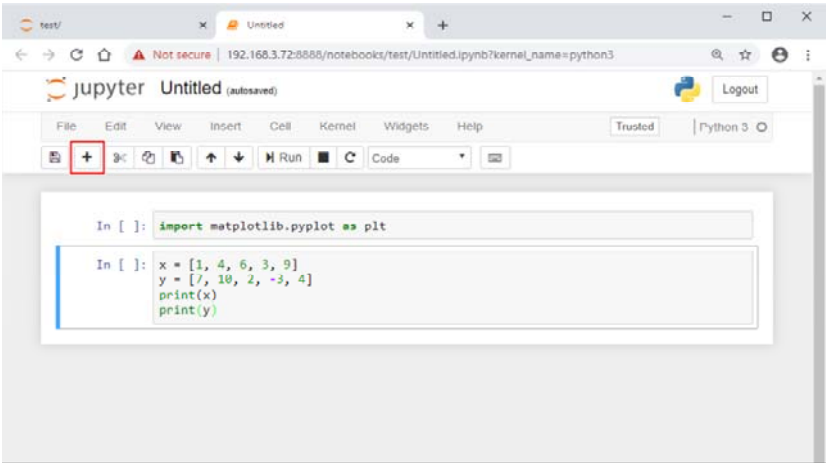


Figure 5.49 Coding -2

Click the [Run] button to run the code in the cell and show the result.

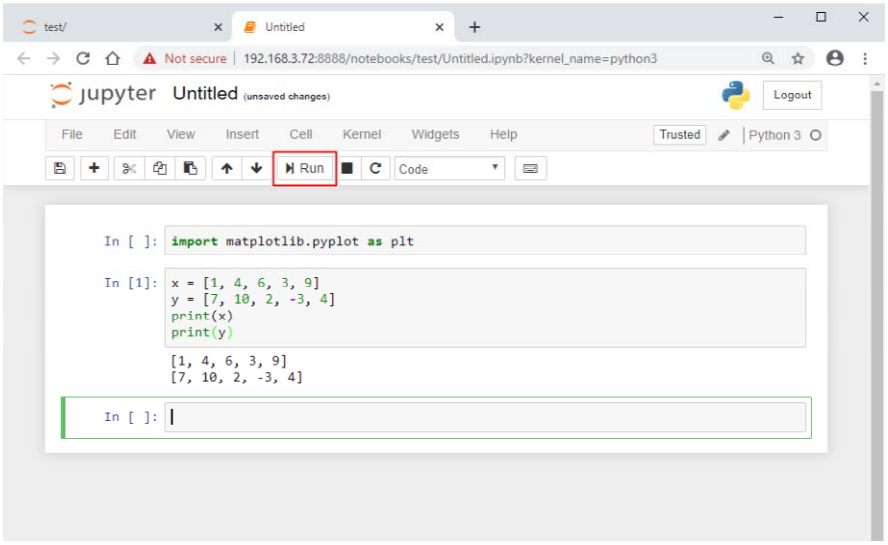


Figure 5.50 Coding -3

In a new cell, write code that draws a scatter diagram.

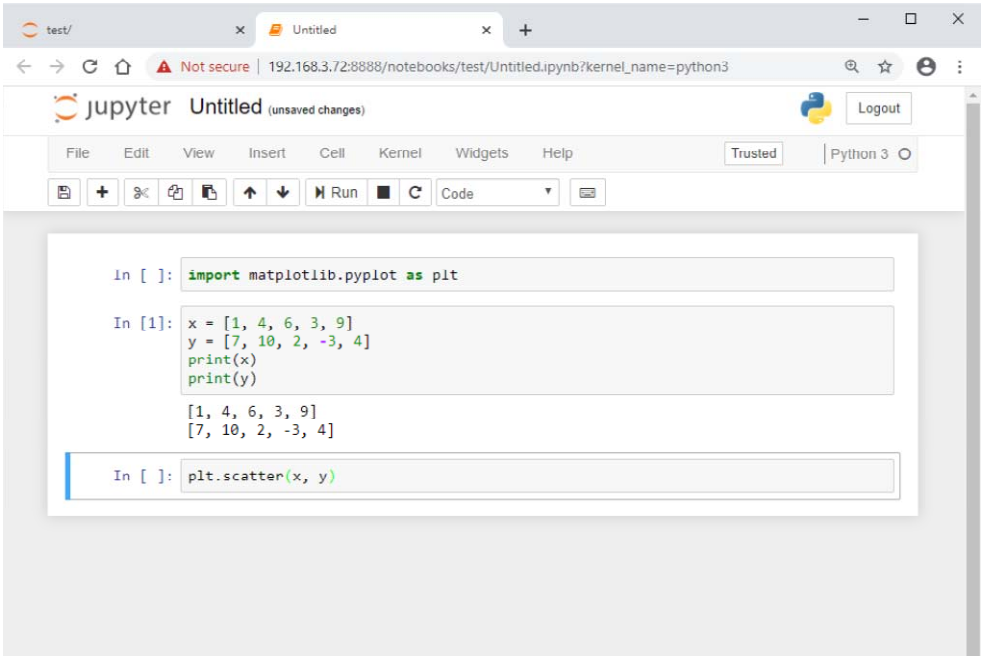


Figure 5.51 Coding -4

Run the top cell and then run the bottom cell to draw the scatter diagram.

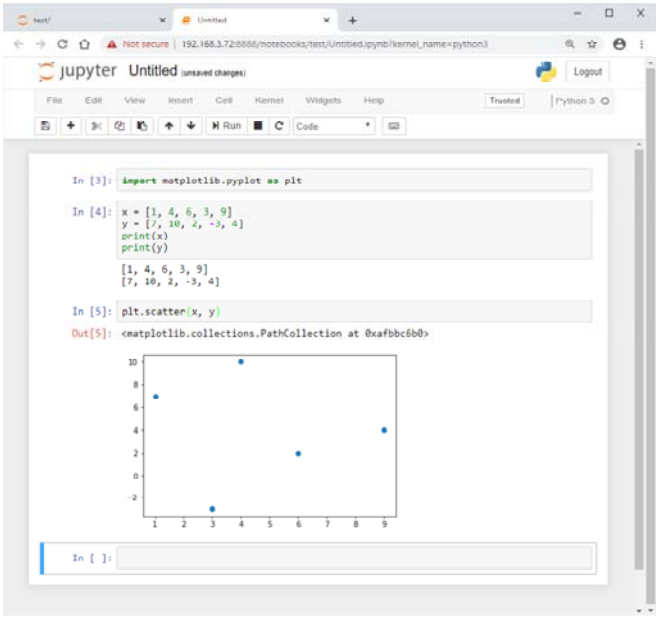


Figure 5.52 Coding -5

Note

You must run the top cell first as the bottom cell uses matplotlib.pyplot.

Note

If you want to run all cells from top to bottom, select the [Cell] menu to show the drop-down list and click [Run All].

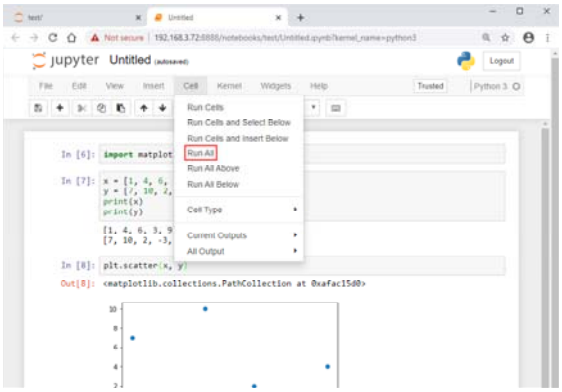


Figure 5.53 Coding -6

- **Exiting Jupyter Notebook**

  In the F3RP70-2L console, press the [Ctrl] + [C] keys to exit Jupyter Notebook.

5.4 How to access the M3IO module

You can configure the M3IO module and perform data I/O by running C/C++ library functions. This section describes how to call C/C++ library functions from Python.

5.4.1 Input output data of IO module

Input / output data of the IO module is read /written from /to the device (relay, register) of the module. The position of the device is specified by the unit, slot and device number in the module. A unit is the smallest unit of the system. The usual unit number is 0. When expanding IO, up to 7 unit (unit numbers 1 to 7) can be added. The slot number represents the position of the module within the unit. From the right next to the power supply module, take a value from 1 to 16. The device number in the module start from 1 and the number varies depending on the module. Generally, input / output data is allocated from device number 1. For example, the data of channel 1 of the analog input module reads and writes the device number 1, and the data of channel 4 reads and writes the device number 4.

Note

- For details on the specifications of the system configuration, refer to “e-RT3 CPU Module (SFRD□2) BSP Common Function Manual (IM 34M06M52-02E)”.
- For details on each module, refer to the manual of each modules.

For modules supported by the IO Module Configuration Service, the service executes the settings in the module so that the user program can be completed only by reading and writing I / O data. The module devices and library functions used are shown below.

Table 5.2 Device of module and access API

Module type	Module model	Device type	IO Data Number	API
Digital input	F3XD08-□□	relay	1～8	Read by 1bit：readM3InRelayP Read by 16bit：readM3InRelay
	F3XD16-□□		1~16	
	F3XD32-□□		1~32	
	F3XD64-□□		1~64	
Digital outpu	F3YD04-□□		1~4	Write by 1bit：writeM3OutRelayP Write by 16bit：writeM3OutRelay Read by 16 bit：readM3OutRelay
	F3YD08-□□		1~8	
	F3YD14-□□		1~14	
	F3YD32-□□		1~32	
	F3YD64-□□		1~64	
Analog input	F3AD04-5R	register	1~4	Read by 16bit：readM3IoRegister
	F3AD08-5R		1~8	
	F3AD08-6R		1~8	
	F3AD08-4R		1~8	
Analog output	F3DA04-6R		1~4	Write by 16bit：writeM3IoRegister Read by 16bit：readM3IoRegister
	F3DA08-5R		1~8	
High speed data acquisition	F3HA06-1R		1~6	Use API described in chapter4 For reading of immediate data: Read by 16bit：readM3IoRegister
	F3HA12-1R		1~12	
Temperature monitor	F3CX04-0N		1~4	1 点単位読出：readM3IoRegister

5.4.2 Calling C/C++ library functions from Python

ctypes is a software module that provides C-compatible data types. To access the IO module from Python, use ctypes to convert Python variables into appropriate types and call functions in the library. The following shows some examples.

Note

For details on the API functions for e-RT3 I/O module access, refer to “Appendix1 I/O Module Access Library” of this document.

- **Reading data from a XD module (1bit)**

The program below reads data from relay number 1 of the XD module inserted into slot 2 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(2)
# Relay number
c_pos = ctypes.c_int(1)

# Create a short-type array for the buffer to store read data
# short-type array with 1 element
short_arr= ctypes.c_uint16 * 1
# create arry
c_data = short_arr()

# Call the library function
libc.readM3InRelayP(c_unit, c_slot, c_pos, c_data)
```

The specification of the readM3InputRelayP function is:
int readM3InputRelayP (int unit, int slot, int pos, unsigned short *data);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

Reading data from a XD module (16bit)

The program below reads data from relay number 1 to 32 of the XD module inserted into slot 2 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(2)
# Relay number
c_pos = ctypes.c_int(1)
# Number of read block (1 block equals to 16 points)
c_num = ctypes.c_int(2)
# Create a short-type array for the buffer to store read data
# short-type array with 4 element
short_arr = ctypes.c_uint16 * 4
# create array
c_data = short_arr()

# Call the library function
libc.readM3InRelay(c_unit, c_slot, c_pos, c_num, c_data)
```

The specification of the readM3IuptRelay function is:
int readM3InputRelay (int unit, int slot, int pos, int num, unsigned short data[4]);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

Writing data to a YD module (1bit)

The program below write data from relay number 1 of the YD module inserted into slot 3 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
```

```
# Slot
c_slot = ctypes.c_int(3)
# Relay number
c_pos = ctypes.c_int(1)
# data for writing
c_data = ctypes.c_uint16(1)

# Call the library function
libc.writeM3OutRelayP(c_unit, c_slot, c_pos, c_data)
```

The specification of the writeM3OutRelayP function is:
int writeM3OutRelay (int unit, int slot, int pos, unsigned short *data);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

- **Writing data to a YD module (16bit)**

  The program below write data from relay number 1 to 32 of the YD module inserted into slot 3 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Slot
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(3)
# Relay number
c_pos = ctypes.c_int(1)
# Write blocks （1 block equals to 16 points）
c_num = ctypes.c_int(2)
# Create a short-type array for the buffer to store write data
short_arr = ctypes.c_uint16 * 4
# Writing data
data = [0xffff, 0xffff]
c_data = short_arr(*data)
# Data mask
mask = [0xffff, 0xffff]
c_mask = short_arr(*mask)

# Call the library function
libc.writeM3OutRelay(c_unit, c_slot, c_pos, c_num, c_data, c_mask)
```

The specification of the writeM3OutRelay function is:
int writeM3OutRelay (int unit, int slot, int pos, int num, unsigned short data[4], unsigned short mask[4]);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

- **Reading data from a YD module (16bit)**

The program below read data from relay number 1 to 32 of the YD module inserted into slot 3 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(3)
# Relay number
c_pos = ctypes.c_int(1)
# Read blocks （1block equals to 16 points）
c_num = ctypes.c_int(2)
# Create a short-type array for the buffer to store read data
# short-type array with 4 element
short_arr = ctypes.c_uint16 * 4
# create array
c_data = short_arr()

# Call the library function
libc.readM3OutRelay(c_unit, c_slot, c_pos, c_num, c_data)
```

The specification of the readM3OutRelay function is:
Int readM3OutRelay (int unit, int slot, int pos, int num, unsigned short data[4]);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

● Writing data to a DA module

The program below writes 6000 into register number 1 of the DA module inserted into slot 4 of unit 0. Channel 1 of the DA module carries a voltage of 3 volts.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")

# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(4)
# register number
c_pos = ctypes.c_int(1)
# Write points
c_num = ctypes.c_int(1)

# Create a short-type array for the buffer to store write data
# python array to be converted
data = [6000]
# short-type array with 1 element
short_arr = ctypes.c_short * 1
# Convert python array into a short-type array
c_data = short_arr(*data)

# Call the library function
libc.writeM3IoRegister(c_unit, c_slot, c_pos, c_num, c_data)
```

The specification of the writeM3IoRegister function is:
int writeM3IoRegister(int unit, int slot, int pos, int num, unsigned short *data);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

● Reading data from the AD module

The program below reads data from register number 1 of the AD module inserted into slot 5 of unit 0.

```
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libm3.so.1")
```

```
# Convert Python variables into the int type
# Unit
c_unit = ctypes.c_int(0)
# Slot
c_slot = ctypes.c_int(5)
# Register number
c_pos = ctypes.c_int(1)
# Read points
c_num = ctypes.c_int(1)

# Create a short-type array for the buffer to store read data
# short-type array with 1 element
short_arr = ctypes.c_short * 1
#create array
c_data = short_arr()

# Call the library function
libc.readM3IoRegister(c_unit, c_slot, c_pos, c_num, c_data)
```

The specification of the readM3IoRegister function is:
int readM3IoRegister(int unit, int slot, int pos, int num, unsigned short *data);
Therefore, Python variables are converted into an appropriate type with ctypes and then the function is called.

- **Reading data from the high-speed data acquisition module**

The program below reads data from register number 1 of the high-speed data acquisition module inserted into slot 3 of unit 0. Store ert3dgc.pu and ha_access_sample.py in the same directory and execute ha_access_sample.py.

Note

Before using high-speed data acquisition module, you have to configure a module using F3HA12 data acquisition service and e-RT3 IO configuration service.
For details of each services, see chapter 3 and 4 of this document.

ert3dgc.py

```
__version__ = '1.1.1-00'
import ctypes

# Load the library
libc = ctypes.cdll.LoadLibrary("/usr/local/lib/libert3dgc.so.1")

LONGLONG_MAX = 0x7fffffffffffffff
```

```
# initialize API resource mode 0:read-write  1:read only
def open_ha(mode=0, unit=0, slot=2):
    # Load library function
    LEDG_open = libc.LEDG_open

    # Specifies type of return value.
    LEDG_open.restype = ctypes.c_bool

    # Convert Python variables into the int type
    c_mode = ctypes.c_int32(mode)
    c_unit = ctypes.c_int32(unit)
    c_slot = ctypes.c_int32(slot)

    # Initialize API resource, return true/false
    return LEDG_open(c_mode, c_unit, c_slot)

# Release API resource
def close_ha():
    LEDG_close = libc.LEDG_close

    # Release API resource
    LEDG_close()

# Get the data number of acquired data
def get_hadatano():
    LEDG_getHaDataNo = libc.LEDG_getHaDataNo
    LEDG_getHaDataNo.restype = ctypes.c_bool

    c_oldestno = ctypes.c_int64(-1)
    c_newestno = ctypes.c_int64(-1)
    c_acq_lasterr = ctypes.c_int32(-1)

    # Get the data number of acquired data
    LEDG_getHaDataNo(ctypes.byref(c_oldestno),          ctypes.byref(c_newestno),
ctypes.byref(c_acq_lasterr))

    # Return oldest, newest data number and error status
    return c_oldestno.value, c_newestno.value, c_acq_lasterr.value

# Create buffer for data acquisition
def create_buffer(bytes_per_scan, num_of_buff):
    buf = ctypes.c_byte * (bytes_per_scan * num_of_buff)
    return buf()

# Get acquired data
def get_hadata(c_buf, num_of_buff, fromno=0, tono=LONGLONG_MAX):
```

```
    c_req_fromno = ctypes.c_int64(fromno)
    c_req_tono = ctypes.c_int64(tono)
    c_num_of_buff = ctypes.c_int32(num_of_buff)
    c_real_fromno = ctypes.c_int64(-1)
    c_real_tono = ctypes.c_int64(-1)
    c_acq_lasterr = ctypes.c_int32(-1)

    LEDG_getHaData = libc.LEDG_getHaData
    LEDG_getHaData.restype = ctypes.c_uint32

    # Get acquired data
    scan_num = LEDG_getHaData(c_req_fromno, c_req_tono, c_buf, c_num_of_buff,
ctypes.byref(c_real_fromno),                          ctypes.byref(c_real_tono),
ctypes.byref(c_acq_lasterr))

    # Return number of scan, start data number, end data number, error status
    return scan_num, c_real_fromno.value, c_real_tono.value, c_acq_lasterr.value

# Get data acquisition target
def get_hagathering():
    # Create a boot-type srray with 12 elements
    boolarr12 = ctypes.c_bool * 12
    c_enable_channels = boolarr12()
    # Convert python variable into bool-type
    c_enable_counter = ctypes.c_bool(False)

    LEDG_getHaGathering = libc.LEDG_getHaGathering
    LEDG_getHaGathering.restype = ctypes.c_uint32

    #Get data acquisition target
    bytes_per_scan          =          LEDG_getHaGathering(c_enable_channels,
ctypes.byref(c_enable_counter))

    # Return number of bytes per a scan, valid of acquisition channel, valid of
counter
    return    bytes_per_scan,    [ch    for    ch    in    c_enable_channels],
c_enable_counter.value

# Start data acquisition
def start_ha():
    LEDG_startHaDataGathering = libc.LEDG_startHaDataGathering
    LEDG_startHaDataGathering.restype = ctypes.c_bool

    # Start data acquisition, return true/false
    return LEDG_startHaDataGathering()
```

```
# Stop data acquisition
def stop_ha():
    LEDG_stopHaDataGathering = libc.LEDG_stopHaDataGathering

    # Stop data acquisition
    LEDG_stopHaDataGathering()
```

ha_access_sample.py

```
import time
import numpy as np

import ert3dgc

def main():
    # Open as read mode
    ert3dgc.open_ha(mode=0, unit=0, slot=3)

    # get number of bytes per a scan and valid of chnannel
    bytes_per_scan, channels, _ = ert3dgc.get_hagathering()
    # Get the number of valid channel
    ch_num = channels.count(True)

    # Definition of buffer for data acquisition
    num_of_buff = 100000
    buf = ert3dgc.create_buffer(bytes_per_scan, num_of_buff)

    # Start acquisition
    ert3dgc.start_ha()

    # Wait for data
    time.sleep(1)

    # Get oldest, newest data number
    oldestno, newestno, _ = ert3dgc.get_hadatano()

    # Get acquisition data from oldest to newest
    scan_num, _, _, _ = ert3dgc.get_hadata(buf, num_of_buff, fromno=oldestno,
tono=newestno)

    # Get data form buffer
    data = np.frombuffer(buf, dtype='int16', count=scan_num*ch_num)

    # Stop acquisition
    ert3dgc.stop_ha()
```

```
    # Close
    ert3dgc.close_ha()

    return

if __name__ == "__main__":
    main()
```

5.5 Sample program

This is a sample to detect anomalies using One Class SVM. Use the data collected from CH1 and CH2 of HA as input. In learning, features are extracted from the collected data to create a model. In prediction, the feature amount is extracted from the collected data and compared with the model to determine whether the data is normal or abnormal.

MLsample
├ert3dgc.py A program to access an I/O module
├feature.py A program to extract features
├drawGraph.py A program to draw a graph
├training.y A program to perform training
├prediction.py A program to make predictions
└prediction_continuous.py A program to make predictions continuously

Each program is as follows:

- ert3dgc.py
  Program for access high-speed data acquisition module.

- feature.py
  It is a program that extracts the feature amount from the collected data. In this sample, the average value of the size of the collected data is used as the feature value.

- draw_graph.py
  It is a program that draws the features used for learning and the boundaries that distinguish between normal and abnormal values. Figure 5.54 shows an example of the drawn graph.

- training.py
  This is a program that uses the collected data for learning. The collected data is divided into a fixed number of pieces, feature extraction is performed for each, and learning is performed based on the obtained feature quantities to determine the discrimination boundaries that determine whether normal or abnormal.

- prediction.py
  It is a program that collects data at regular intervals and makes predictions. The collected data is divided into fixed numbers and feature extraction is performed for each, and the obtained feature values are compared with the model obtained by learning to determine whether they are normal or abnormal.

・ predicition_continuous.py

It is a program that continuously collects data and makes predictions. As soon as data for one prediction is accumulated, feature extraction is performed, and the obtained feature amount is compared with the model obtained by learning to determine whether it is normal or abnormal.

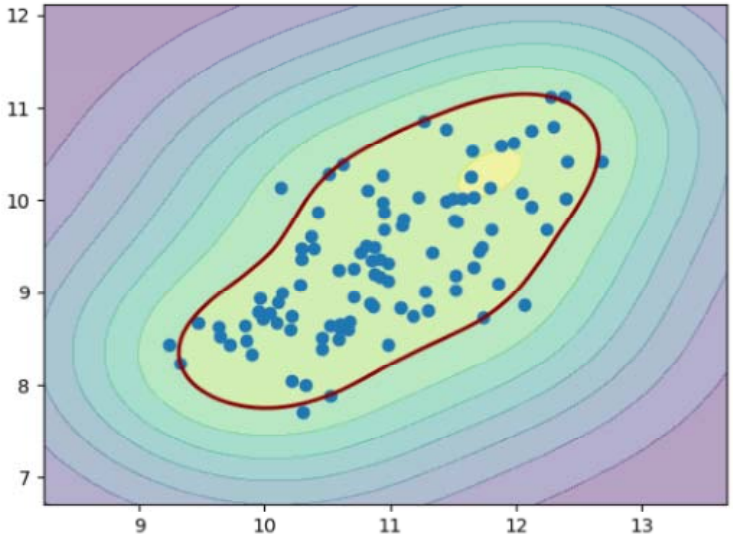


Figure 5.54 example, graph of model (blue point: feature, red line: discrimination boundaries)

Note

Before using high-speed data acquisition module, you have to configure a module using F3HA12 data acquisition service and e-RT3 IO configuration service.
For details of each services, see chapter 3 and 4 of this document.

● Usage

1. Place the “MLsample” folder in a given directory in F3RP70-2L
2. Execute training.py for learning.

```
ert3@ubuntu:~/workspace/MLsample$ python3 training.py
```

3. Execute prediction.py for prediction.
   For stop the program, press “Ctrl” + “C” key.

```
ert3@ubuntu:~/workspace/MLsample$ python3 prediction.py
```

Execute prediction_continuous.py for prediction for prediction of continuous

data.

```
ert3@ubuntu:~/workspace/MLsample$ python3 prediction_continuous.py
```

- **Source code**

ert3dgc.py
See chapter 5.4.2

feature.py

```
#MLsample version 1.1.1-00
import numpy as np

# Feature extraction  rawdata: acquired data, ch_num: number of channel, data_size:
numberof data for learning or prediction

def feature_extraction(rawdata, ch_num, data_size):
    # Divide acquired data into data of channel every DATA_SIZE
    # chdataset:3D array(number of learning orprediction) × (number of channel) ×
(number of data for learning or predition)
    chdataset = rawdata.reshape(-1, data_size, ch_num).transpose(0, 2, 1)

    # Calculate the average value of the data of learning or prediction data every
channel
    #feature: 2D array(number of data for learning or prediction)  ×  (number of
channels)
    feature = [[calc_aveamp(ch) for ch in data] for data in chdataset]

    return feature

# Average of absolute values of array values
def calc_aveamp(data):
    data = np.array(data)
    data = list(np.abs(data))
    return sum(data)/len(data)
```

draw_graph.py

```
#MLsample version 1.1.1-00
import numpy as np
import matplotlib
matplotlib.use("Agg")
import matplotlib.pyplot as plt

# Draw graph
def draw_graph(data, model):
```

```
    data=np.array(data)
    # create mesh grid
    x1min, x1max = data[:, 0].min()-1, data[:, 0].max()+1
    x2min, x2max = data[:, 1].min()-1, data[:, 1].max()+1
    xx1,  xx2  =  np.meshgrid(np.linspace(x1min,  x1max,  500),np.linspace(x2min,
x2max, 500))

    Z = model.decision_function(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)

    # Draw contour lines
    plt.contourf(xx1, xx2, Z, alpha=0.4)

    # Draw discrimination boundary
    plt.contour(xx1, xx2, Z, levels=[0], linewidths=2, colors='darkred')

    # Draw data
    plt.scatter(data[:, 0], data[:, 1])

    # Save picture
    plt.savefig("graph.png")
```

training.py

```
#MLsample version 1.1.1-00
import time
import pickle

import numpy as np
from sklearn import svm

import ert3dgc
from feature import feature_extraction
from draw_graph import draw_graph

DATA_NUM = 100 # Number of learning data
DATA_SIZE = 1000 # Number of learning data
GATHER_NUM = DATA_SIZE * DATA_NUM # Number of acquired data

def main():
    # Open read-write mode
    print("open = ", ert3dgc.open_ha(mode=0, unit=0, slot=3))

    # Get number of bytes per a scan
    bytes_per_scan, _, _ = ert3dgc.get_hagathering()

    # Get number of valid channels
```

```
    ch_num = bytes_per_scan // 2

    # Definition of buffer for acquired data
    buf = ert3dgc.create_buffer(bytes_per_scan, GATHER_NUM)

    # Start data acquisition
    print("start = ", ert3dgc.start_ha())

    # Get newest data number
    _, newestno, _ = ert3dgc.get_hadatano()

    # Wait for data
    while newestno + 1 < GATHER_NUM:
        time.sleep(1)
        _, newestno, _ = ert3dgc.get_hadatano()

    # Get data of which size is GATHER_NUM
    scan_num, _, _, _ = ert3dgc.get_hadata(buf, GATHER_NUM, fromno=newestno-
GATHER_NUM+1)

    # Get acquired data from buffer
    data = np.frombuffer(buf, dtype='int16', count=scan_num*ch_num)

    # learning
    training(data, ch_num)

    # Stop data acquisition
    ert3dgc.stop_ha()
    print("stop")

    # Close
    ert3dgc.close_ha()
    print("close")

def training(rawdata, ch_num):
    # feature extraction
    feature = feature_extraction(rawdata, ch_num, DATA_SIZE)

    # learning
    clf = svm.OneClassSVM(nu=0.1, kernel="rbf")
    clf.fit(feature)

    # Save model
    pickle.dump(clf, open("model.pickle", 'wb'))

    # Draw graph
```

```
    draw_graph(feature, clf)

if __name__ == "__main__":
    main()
```

prediction.py

```
#MLsample version 1.1.1-00
import time
import pickle

import numpy as np

import ert3dgc
from feature import feature_extraction

CLF = pickle.load(open("model.pickle", 'rb'))# Read model
DATA_NUM = 10 # Number of prediction data for 1 batch
DATA_SIZE = 1000 # Number of acquired data for 1 prediction
GATHER_NUM = DATA_SIZE * DATA_NUM # Number of acquired data

def main():
    # Open as read-write mode
    print("open = ", ert3dgc.open_ha(mode=0, unit=0, slot=3))

    # Get number of bytes per a scan
    bytes_per_scan, _, _ = ert3dgc.get_hagathering()

    # Get number of valid channels
    ch_num = bytes_per_scan // 2

    # Definition of buffer for acquired data
    buf = ert3dgc.create_buffer(bytes_per_scan, GATHER_NUM)

    # Start data acquisition
    print("start = ", ert3dgc.start_ha())

    # Get newest data number
    _, newestno, _ = ert3dgc.get_hadatano()

    # Wit for data
    while newestno + 1 < GATHER_NUM:
        time.sleep(1)
        _, newestno, _ = ert3dgc.get_hadatano()

    try:
        while True:
```

```
            # Get newest data number
            _, newestno, _ = ert3dgc.get_hadatano()

            # Get data of which size is GATHER_NUM
            scan_num,   _,   _,   _   =   ert3dgc.get_hadata(buf,   GATHER_NUM,
fromno=newestno-GATHER_NUM+1)

            # Get acquired data from buffer
            data = np.frombuffer(buf, dtype='int16', count=scan_num*ch_num)

            # Prediction
            prediction(data, ch_num)

            # interval for prediction
            time.sleep(1)
    except KeyboardInterrupt:
        # Stop data acquisition
        ert3dgc.stop_ha()
        print("stop")

        # Close
        ert3dgc.close_ha()
        print("close")

    return

def prediction(rawdata, ch_num):
    # feature extraction
    feature = feature_extraction(rawdata, ch_num, DATA_SIZE)

    # Prediction
    y_pred_test = CLF.predict(feature)
    n_outlier_test = y_pred_test[y_pred_test == -1].size
    # Number of prediction data
    print("number of test data: ", len(y_pred_test))
    # Number of errors
    print("number of outliers: ", n_outlier_test)
    print("")

if __name__ == "__main__":
    main()
```

prediction_continuous.py

```
#MLsample version 1.1.1-00
import pickle
```

```
import numpy as np

import ert3dgc
from feature import feature_extraction

CLF = pickle.load(open("model.pickle", 'rb'))# Read model
DATA_SIZE = 1000 # Number of acquired data for 1 prediction
NUM_OF_BUFF = 100000 # Size of buffer for acquired data

def main():
    # Open as read-write mode
    print("open = ", ert3dgc.open_ha(mode=0, unit=0, slot=3))

    # Get number of bytes per a scan
    bytes_per_scan, _, _ = ert3dgc.get_hagathering()

    # Get number of valid channels
    ch_num = bytes_per_scan // 2

    # Definition of buffer for acquired data
    buf = ert3dgc.create_buffer(bytes_per_scan, NUM_OF_BUFF)

    # Start data acquisition
    print("start = ", ert3dgc.start_ha())

    try:
        # Start data acquisition number
        req_fromno = 0
        # FIFO
        queue = np.array([])
        while True:
            # Get data from “req_from” to latest
            scan_num,   real_fromno,   real_tono,   _   =   ert3dgc.get_hadata(buf,
NUM_OF_BUFF, fromno=req_fromno)

            # If you cannot get expected data
            if req_fromno < real_fromno:
                # Initialize queue
                queue = np.array([])
                print("WARNING: some data lost")

            # Get data from buffer
            queue    =    np.append(queue,    np.frombuffer(buf,    dtype='int16',
count=scan_num*ch_num))

            # Enough data is stored
```

```
            while len(queue) >= (DATA_SIZE*ch_num):
                # Get the top element and remove from the queue
                data = queue[:DATA_SIZE*ch_num]
                queue = queue[DATA_SIZE*ch_num:]
                # Prediction
                prediction(data, ch_num)

            # When you can get some data
            if scan_num > 0:
                # add 1 to real_tono for next request number
                req_fromno = real_tono + 1

    except KeyboardInterrupt:
        # Stop data acquisition
        ert3dgc.stop_ha()
        print("stop")

        # Close
        ert3dgc.close_ha()
        print("close")

    return

def prediction(rawdata, ch_num):
    # Feature extraction
    feature = feature_extraction(rawdata, ch_num, DATA_SIZE)

    # Prediction
    y_pred_test = CLF.predict(feature)
    n_outlier_test = y_pred_test[y_pred_test == -1].size
    # Number of prediction data
    print("number of test data: ", len(y_pred_test))
    # Number errors
    print("number of outliers: ", n_outlier_test)
    print("")

if __name__ == "__main__":
    main()
```

6. Application development with C/C++

6.1 Host development with F3RP70-2L

This Ubuntu image has the build-essential package for the armhf architecture installed as a build toolchain for C/C++ programs.
This section describes how to build and execute a program on F3RP70-2L using this toolchain.

6.1.1 Usage

- **Preparations**

To transfer source files to F3RP70-2L, install WinSCP on the local machine.

1. Access the following URL to download WinSCP:
https://winscp.net/eng/download.php

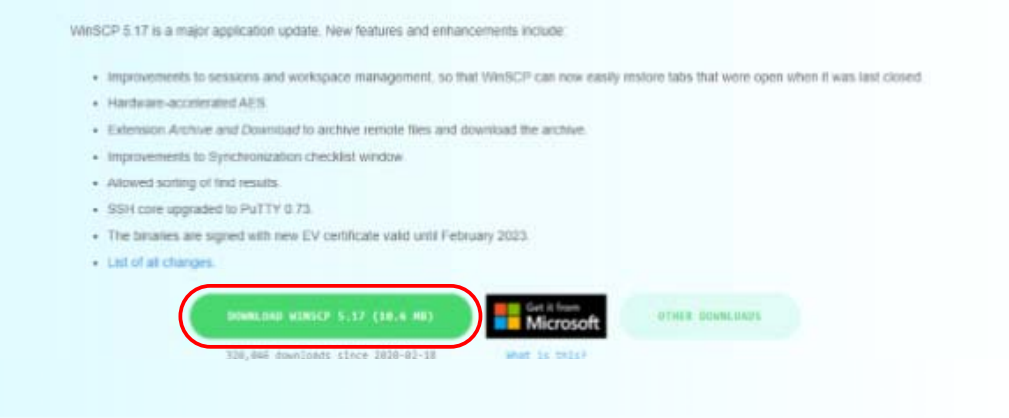


Figure 6.1 Installation of WinSCP -1

2. Run the file you downloaded to install WinSCP.

Click [Install for all users].

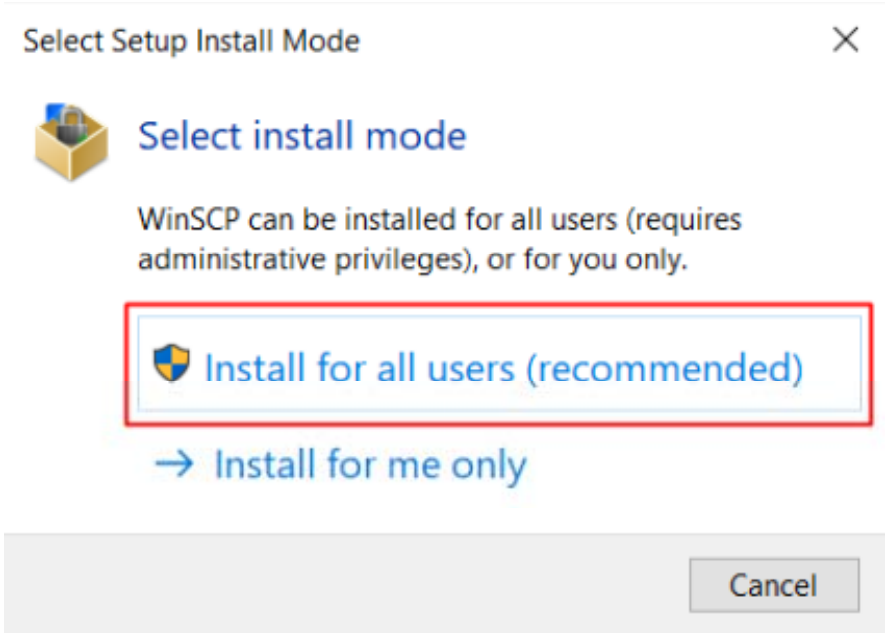


Figure 6.2 Installation of WinSCP -2

Click [Yes].

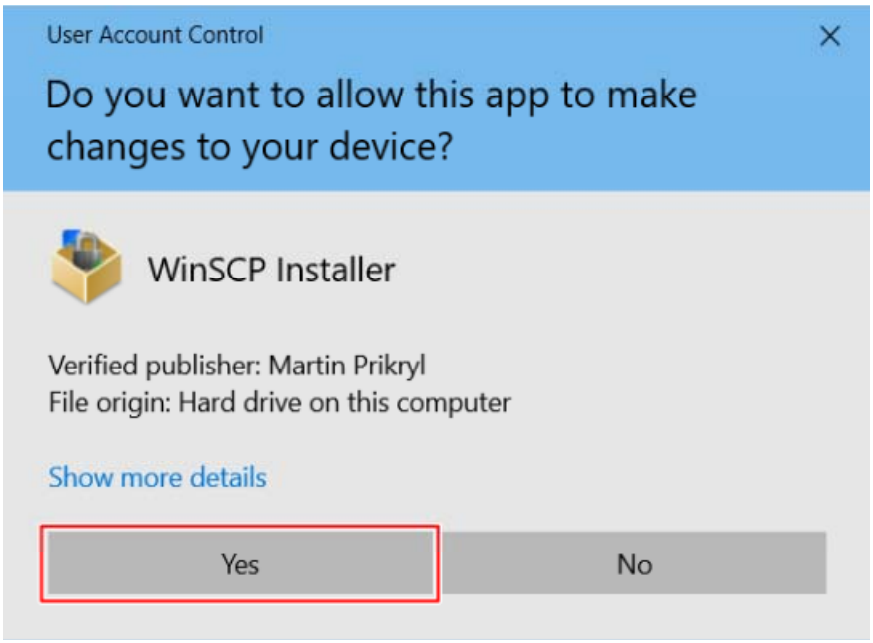


Figure 6.3 Installation of WinSCP -3

Click [Accept].

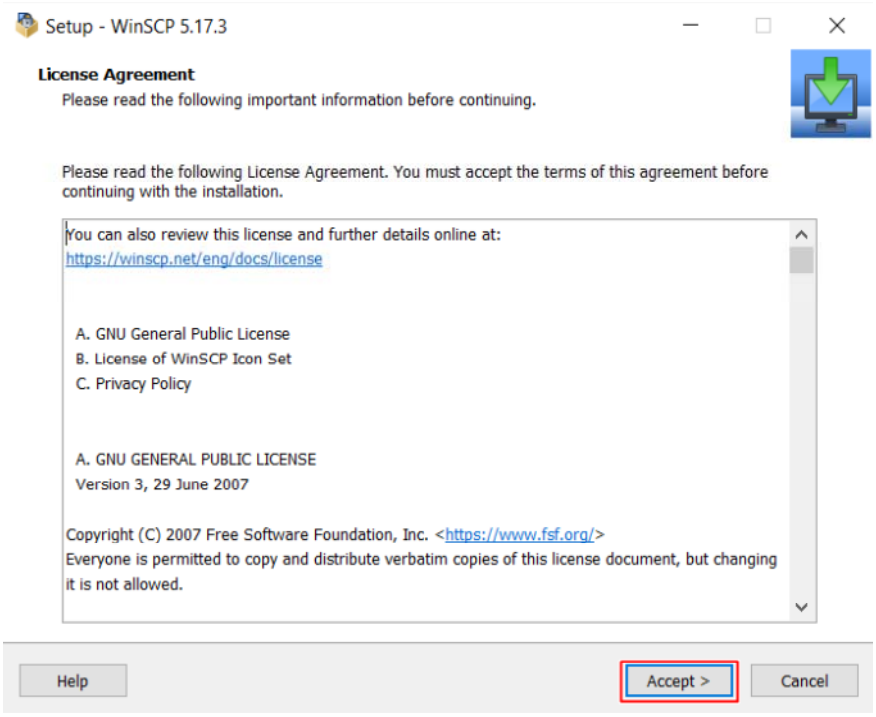


Figure 6.4 Installation of WinSCP -4

Select [Typical installation] and click [Next].

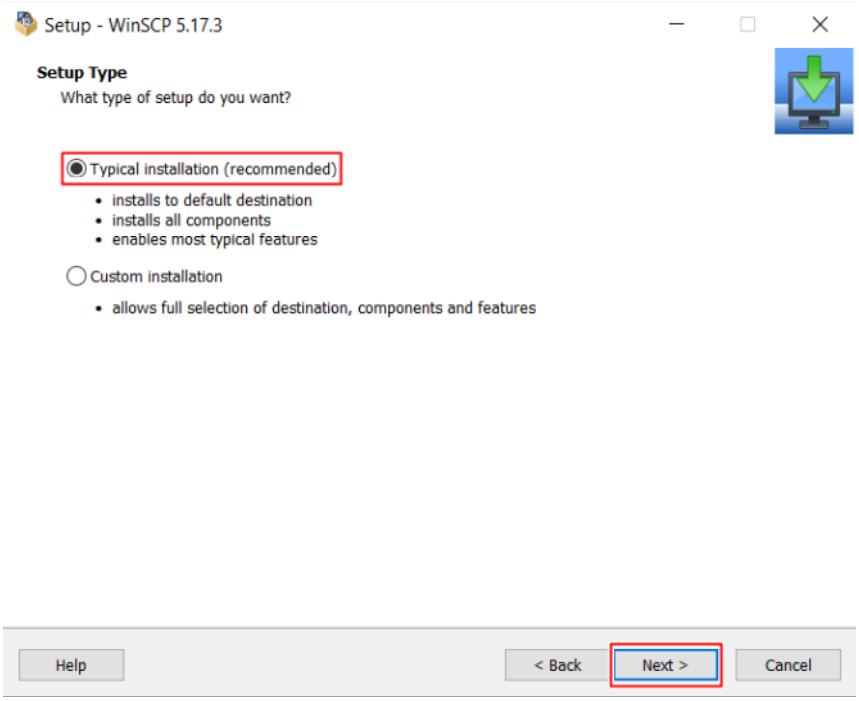


Figure 6.5 Installation of WinSCP -5

Click [Next].

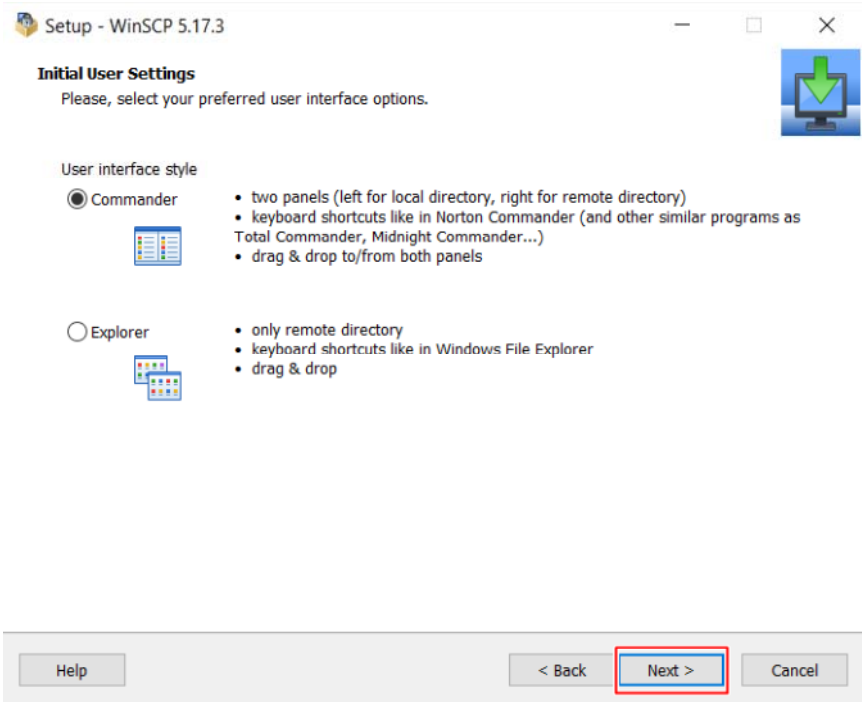


Figure 6.6 Installation of WinSCP -6

Click [Install].

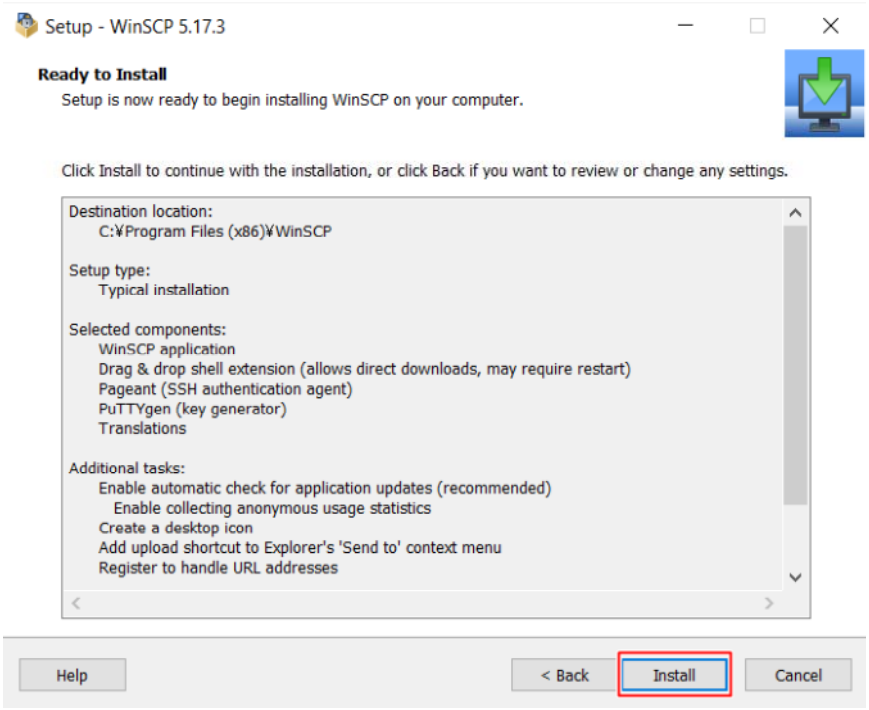


Figure 6.7 Installation of WinSCP -7

The installation is now started.

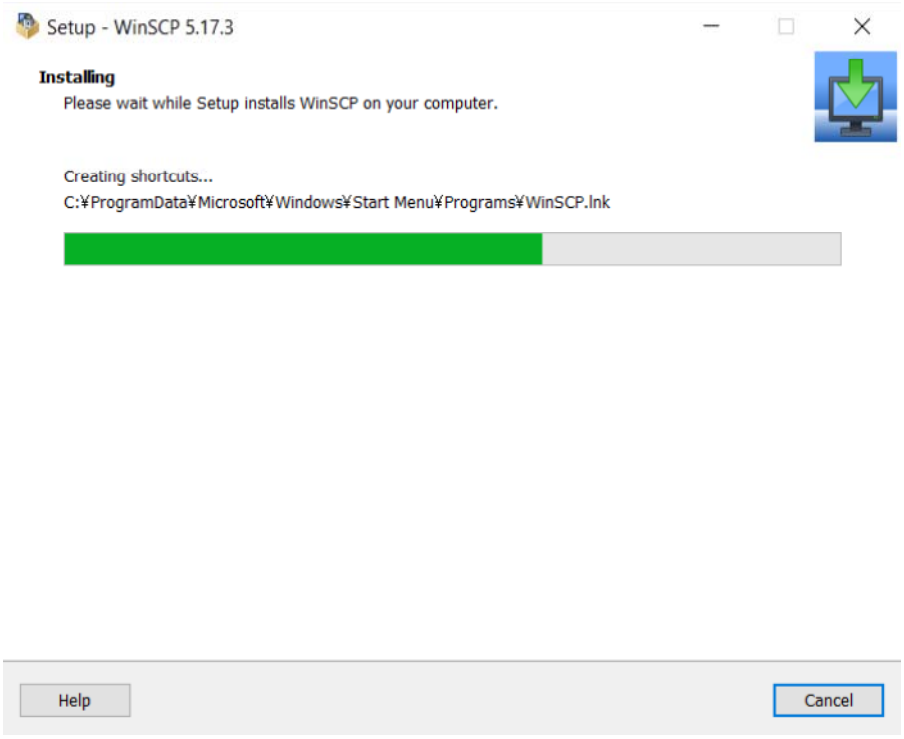


Figure 6.8 Installation of WinSCP -8

Click [Finish].

Figure 6.9 Installation of WinSCP -9

3. Configure the connection settings.
Fill out the [Host name], [User name], and [Password] fields and click [Login].

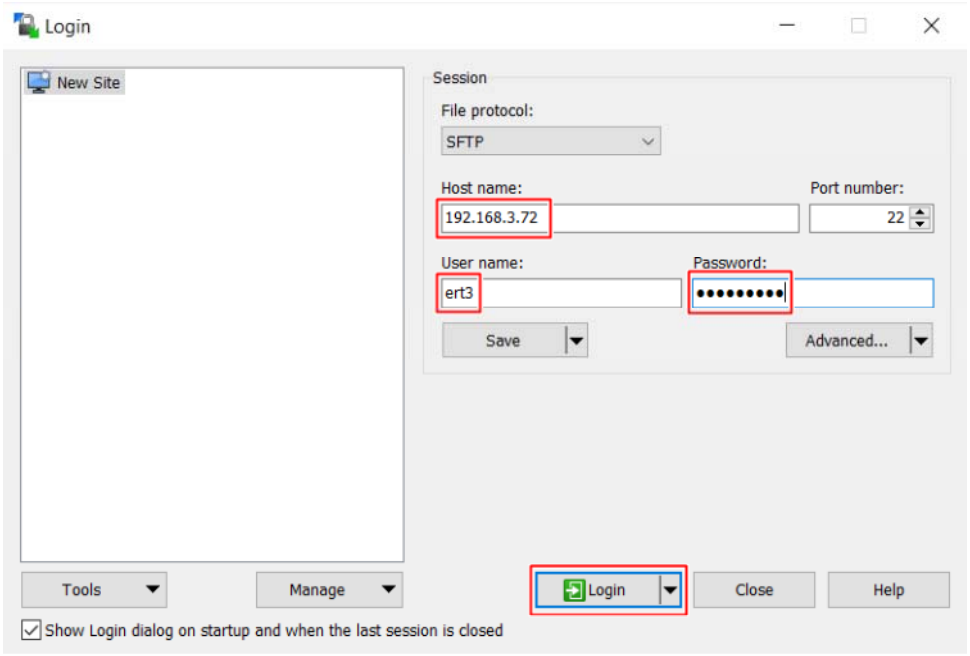


Figure 6.10 Installation of WinSCP -10

Click [Yes].

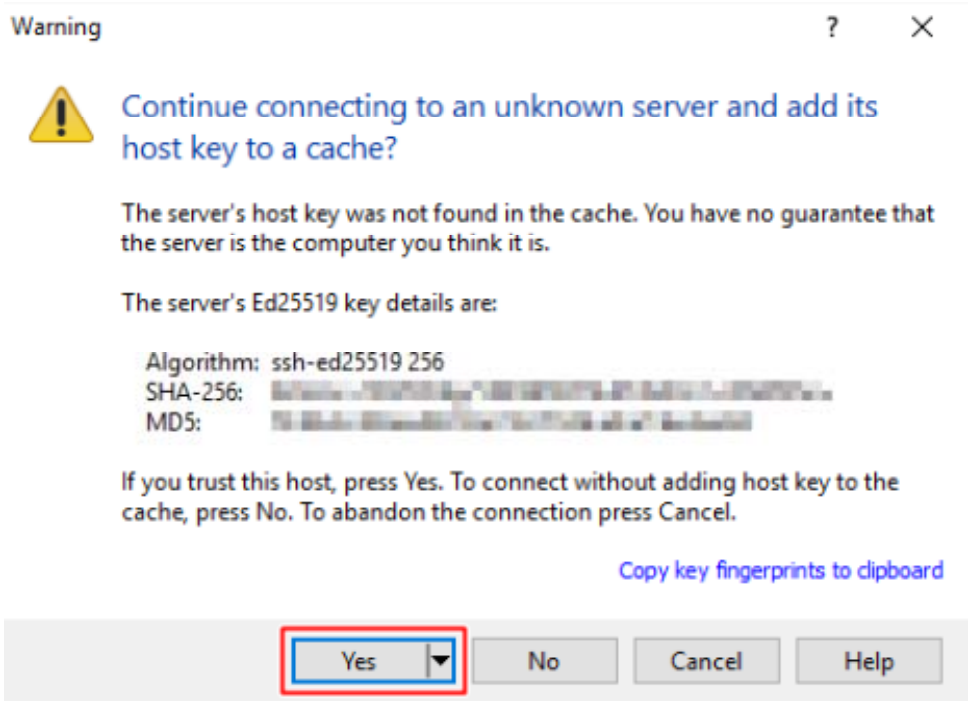


Figure 6.11 Installation of WinSCP -11

The directories on the local machine are displayed on the left side and the directories in F3RP70-2L on the right side.

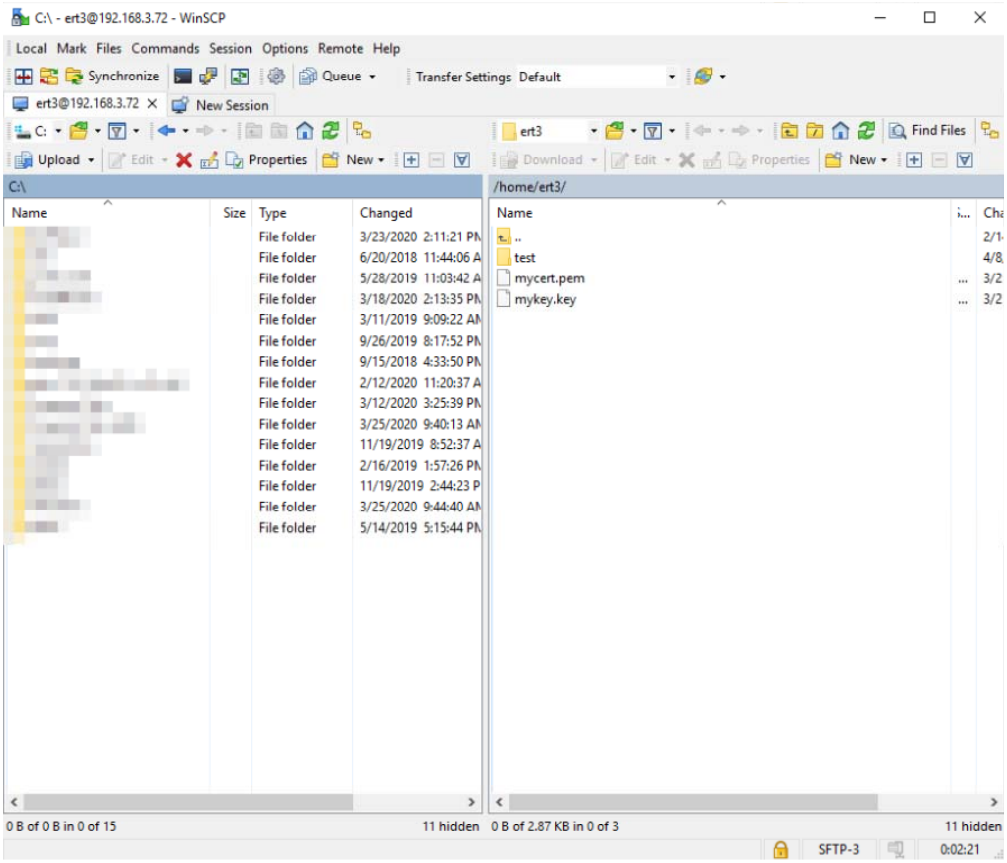


Figure 6.12 Installation of WinSCP -12

● Creating source code

Create “HelloWorld.c” in a given directory on the local machine and write the following code:

```
#include <stdio.h>

int main(void)
{
  printf("Hello World!\n");
  return 0;
}
```

In WinSCP, drag “HelloWorld.c” and drop it into the home directory (“/home/ert3/”) of the ert3 user, and click [OK].

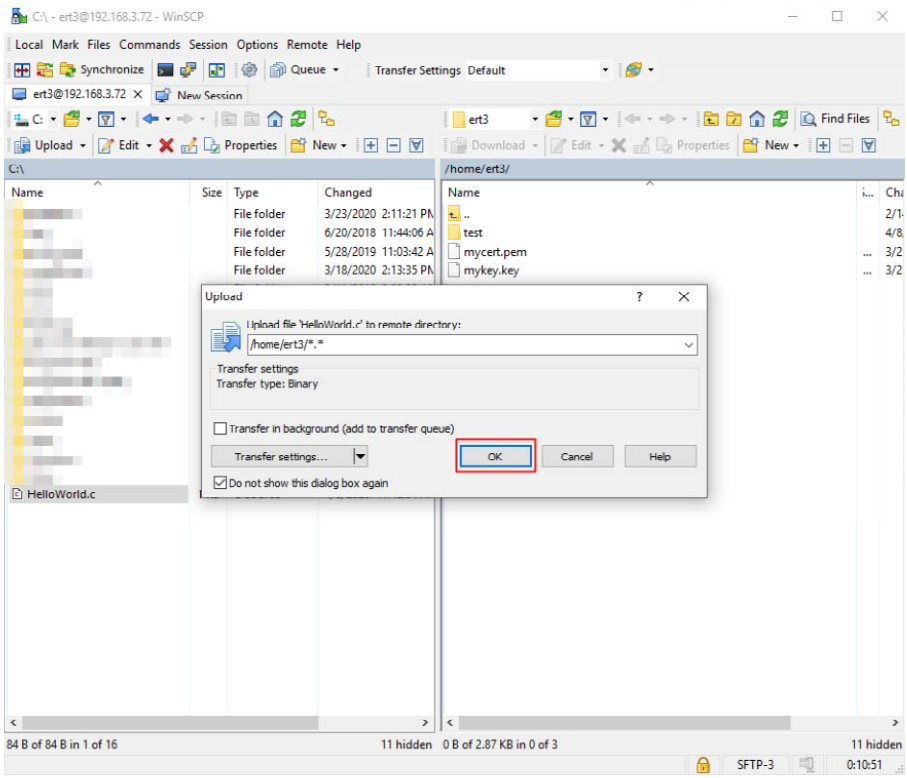


Figure 6.13 Installation of WinSCP -13

Check that “HelloWorld.c” is added to the home directory.

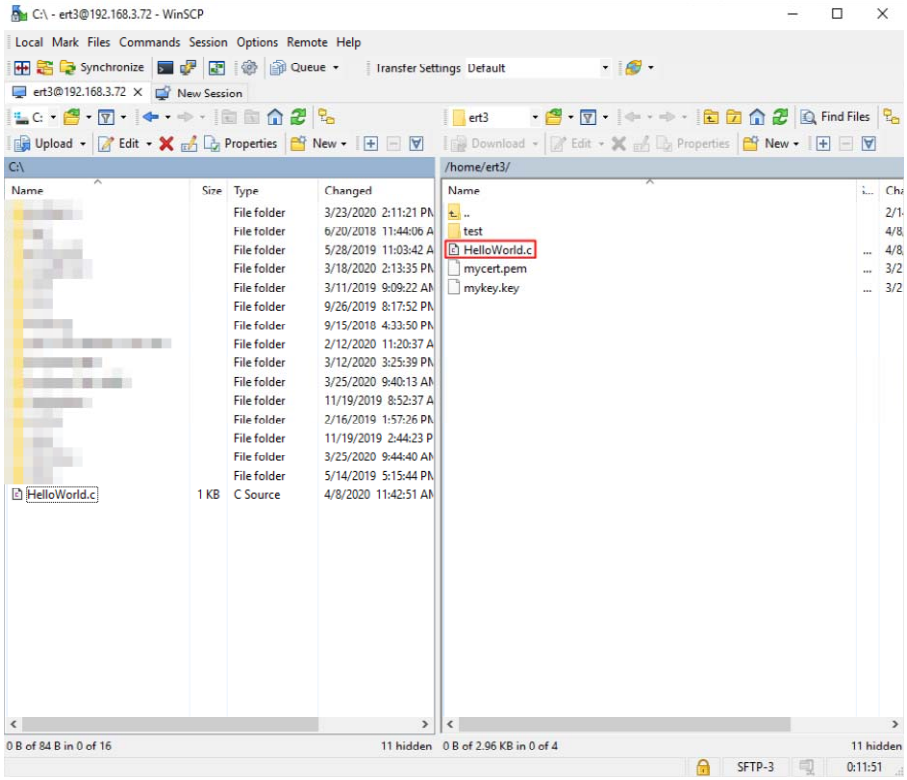


Figure 6.14 Installation of WinSCP -14

- **Building and running the source code**

  In the console, go to the home directory of the ert3 user and run the following command to build the code:

```
ert3@ubuntu:~$ arm-linux-gnueabihf-gcc -o HelloWorld HelloWorld.c
```

  Run the following command to execute the program and check the output.

```
ert3@ubuntu:~$ ./HelloWorld
Hello World!
```

6.1.2 Using the e-RT3 -specific API functions

API functions to access the e-RT3 I/O module are provided.
The library is stored in “/usr/local/lib” and the header files are “/usr/local/include/ert3”.

Note

For details on the API functions for e-RT3 I/O module access, refer to “Appendix1 I/O Module Access Library” of this document.

7. Overlay Filesystem

This chapter describes the functions and usage of the Overlay Filesystem (OverlayFS) addon software option for Ubuntu. By working with OverlayFS enabled, you can reduce negative effects on the system of unexpected power failures.

Note

This function is included in the F3RP70 Ubuntu image R1.2.1 and later. To find the revision of your Ubuntu image, see 2.2.1, “Specifications of the Ubuntu image” in this document.

7.1 Overview

7.1.1 OverlayFS overview

OverlayFS consists of 3 files systems: a lower-layer, upper-layer, and merged upper-lower layer file system. The lower layer is a read-only file system that resides on the F3RP70-2L's SD memory card. This is the file system on which Ubuntu is loaded, and from which Ubuntu reads.

The upper layer and merged upper-lower layer file systems reside on a RAM disk on the F3RP70, and any changes made in the file system (e.g. file add/edit/delete) after bootup of Ubuntu are made to the upper layers only, leaving the lower layer unmodified. That is why it is protected from power interruptions. For a more detailed explanation, see section 7.2.

- **Target**

  OverlayFS is useful for the following.
  - To prevent negative effects on the system of unexpected power interruptions
  - To prevent changes of the default operating environment

  See also the operating precautions in section 7.4.

7.1.2 Overview of procedures

The following are the major steps involved in preparing to use OverlayFS. For details, see section 7.3.

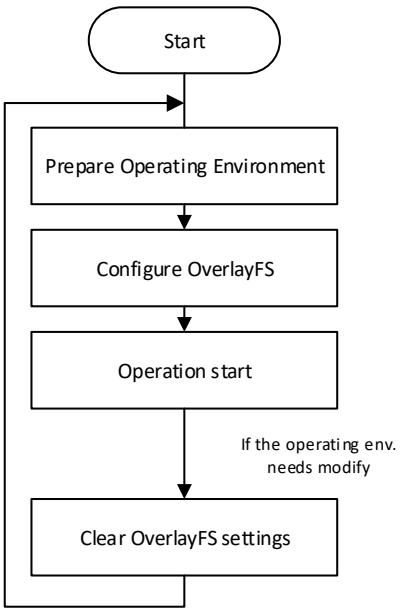


Figure. 7.1 Preparation for, and use of OverlayFS

7.2 Description of OverlayFS

Features

- Power interruption resistance
  Prevents negative effects on the system due to unexpected power interruptions that the shutdown process cannot address.
- Same operation as usual
  Enabling OverlayFS does not affect normal operations.
- Simple
  Enabling/disabling setting can be done with just a parameter change.

Description

The following describes each of the three file systems.

- Lower layer
  This is the core OverlayFS file system. The contents prepared when OverlayFS disabled (i.e. normal startup) are configured as the lower layer.
- Upper layer
  This file system records the difference from the lower layer after startup.
- Merged upper-lower layer
  This file system merges the lower and upper layers, and in normal operation the user sees the merged directory.

OverlayFS places these three layers from bottom to top in the order: lower, upper, merged. Thus, the user sees the merged file system at the top, and operates the F3RP70 without being aware of the layering. Changes to files or directories (e.g. add/edit/delete) are made in the upper level that is visible to the user, but the original lower level remains unchanged. Therefore, turning off the power or restarting Ubuntu will return the system to the state prior to the change.

Note that in the case of the F3RP70-2L, the lower, read-only layer resides on the SD memory card, whereas the upper and merged upper-lower file systems are loaded into the RAM disk. Because of this, no unexpected writing to the SD card occurs. This is how negative effects on the original system from unexpected power interruptions are prevented. However, if there is information that you want to save, you need to create a separate storage location.

Note

Operation with OverlayFS is optional and can be selected according to the customer's environment. By default, OverlayFS is disabled and Ubuntu starts as normal.

7.3 Enter settings

This section describes the OverlayFS setting procedure in detail.

■ Development environment

Items required for development and equipment configuration are the same as for serial console connection in "2.4.2 Procedure of log in to Ubuntu" in this document. Before starting Ubuntu, there are tasks to perform from the bootloader.

As some work requires root privileges, enable the sudo command according to the procedure in 2.4.3, “Enable the sudo command."

7.3.1 Preparing the operating environment

As explained in section 7.1, OverlayFS is built based on the lower layer file system. First, we will prepare the environment that we want to be the base.

1. Start Ubuntu
   Follow the procedure in chapter 2 to create the SD memory card for startup, and start Ubuntu on the F3RP70-2L.

2. Prepare functions required for operation
   OverlayFS is disabled upon initial startup, so you can proceed with your development work as usual (installing packages, creating and placing executables, and so on).

3. Update initramfs
   initramfs is a file that is loaded when launching Ubuntu, and is required for startup. When each function is ready, run the following command and update **initrd.img-xxxxxx** in the **/boot** directory. The placeholder "**xxxxxx**" corresponds to the kernel version and is referenced by the **uname -r** command.

   ```
   $ sudo update-initramfs -u
   ```

4. Update initrd.img
   Convert **initramfs** created in step 3 to **initrd.img** in a format that can be read by U-Boot installed in F3RP70-2L.

   To create **initrd.img** from **initramfs**, regenerate **initrd.img** from the new **initramfs** as shown in the following command. This completes preparation of the operating environment.

   ```
   $ sudo mkimage -A arm -O linux -T ramdisk -C none -n "/boot/initrd.img-$(uname -
   r)" -d /boot/initrd.img-$(uname -r) /boot/initrd.img
   ```

7.3.2 Configuring OverlayFS

When the operating environment is ready, enable OverlayFS and configure it for startup.

1. Edit /boot/ert3Env
   Edit the settings file for u-boot environmental variables **/boot/ert3Env**. Open the settings file, add the last line as **"kuopt='overlayroot=tmpfs'"**, and then save and close the file.

```
$ sudo vi /boot/ert3Env

# kernel image filename
kfile=vmlinuz-4.14.164-rt73-ert3xlnx
...
autorun=yes
kuopt='overlayroot=tmpfs'
```

2. Restart Ubuntu.
   If you restart and log in, a message appears indicating that OverlayFS is enabled.

```
tmpfs-root /media/root-rw tmpfs rw,relatime 0 0
overlayroot / overlay rw,relatime,lowerdir=/media/root-ro,upperdir=/media/root-
rw/overlay,workdir=/media/root-rw/overlay-workdir/_ 0 0
/dev/mmcblk0p2 /media/root-ro ext4 ro,relatime,data=ordered 0 0
```

   You can also use other commands to display mount information.

Note

Some service errors may appear in the Ubuntu startup message, but there is no significant operational impact. For details, see the operational precautions in section 7.4.

7.3.3 Clearing OverlayFS settings

When you want to disable OverlayFS to return to normal startup, use the following procedure.

1. Disable OverlayFS temporally.
   Execute "overlayroot-chroot" command to enable writing to the SD memory card.

```
$ sudo overlayroot-chroot
INFO: Chrooting into [/media/root-ro]
#
```

2. Edit /boot/ert3Env
   Edit the settings file for u-boot environmental variables **/boot/ert3Env**. Open the settings file, remove or comment out the last line **"kuopt='overlayroot=tmpfs'"** added in the previous section 7.3.2 and then save and close the file.

```
$ sudo vi /boot/ert3Env

# kernel image filename
kfile=vmlinuz-4.14.164-rt73-ert3xlnx
...
autorun=yes
#kuopt='overlayroot=tmpfs'
```

3. Return to enable OverlayFS.
   Execute "exit" command to return to enable OverlayFS.

```
# exit
```

4. Restart Ubuntu.
   If you restart and log in, OverlayFS is disabled and starts as normal.

7.4 Usage precautions

The following are precautions when operating with OverlayFS. Please read this document before use.

- **Saving data**

While operating with OverlayFS, data cannot be saved to the SD memory card containing the Ubuntu image. If you acquire data during operation that you want to save after turning off or restarting, you must prepare another device, such as a separate SD memory card. When the SD memory card is inserted into the F3RP70-2L, slot 1 is recognized as **/dev/mmcblk0**, and slot 2 is recognized as **/dev/mmcblk1**. Mount it to the appropriate directories. For example, format the SD card inserted into slot 2 in ext4 format and mount it as follows.

```
$ sudo mkdir /media/sd
$ sudo mkfs -t ext4 /dev/mmcblk1
$ sudo mount /dev/mmcblk1 /media/sd
```

Remove the mount before removing the SD card or turning off the F3RP70-2L.

```
$ sudo umount /media/sd
```

Note

OverlayFS does not affect any data storage area that you prepare for retaining when the power is turned off or the system is restarted. Therefore, be sure not to turn off the power when writing to this storage area. Also, format the SD memory card only for the first time.

- **Capacity**

When starting with OverlayFS enabled, the merged upper-lower layer file system and the upper layer file system are created in tmpfs format on the RAM disk. For the F3RP70-2L, the maximum capacity of the tmpfs format is 512 MB by default, and the total volume of the tmpfs file system is 512 MB. You can check the currently used capacity with the **df** command as follows. Ensure that the totals of “Used” of tmpfs and tmpfs-root at “Filesystem” do not exceed 512MB.

```
$ df
Filesystem     1K-blocks    Used Available Use% Mounted on
udev              500820       0    500820   0% /dev
tmpfs             102732    4132     98600   5% /run
/dev/mmcblk0p2   3444992 1449116   1801164  45% /media/root-ro
tmpfs-root        513644  103920    409724  21% /media/root-rw
overlayroot       513644  103920    409724  21% /
```

```
tmpfs             513644      20    513624   1% /dev/shm
tmpfs               5120       0      5120   0% /run/lock
tmpfs             513644       0    513644   0% /sys/fs/cgroup
tmpfs             102728       0    102728   0% /run/user/1000
```

- **Starting systemd-tmpfiles-setup.service**

  When starting with OverlayFS enabled, the systemd-tmpfiles-setup.service fails to start, and the storage location of the **journald** log changes from its normal **/var/log/journal** and its subdirectories to **/run/log/journal** and its subdirectories. The following errors are displayed at startup.

  ```
  [FAILED] Failed to start Create Volatile Files and Directories.
  See 'systemctl status systemd-tmpfiles-setup.service' for details.
  ```

  Failure to start systemd-tmpfiles-setup.service can be avoided by renaming or deleting /var/log/journal. Remove of delete it after clearing OverlayFS settings, if you need. Below is an example of a rename.

  ```
  mv /var/log/journal /var/log/journal.old
  ```

  Note that, regardless of enabling/disabling countermeasures, starting with OverlayFS enabled, the journald log is saved within **/run/log/journal**. Note, however, that if starting with OverlayFS disabled, if there is no **/var/log/journal**, the log is saved in **/run/log/journal**, and log disappears if the power is turned off.

Appendix1. I/O Module Access Library

This section contains information on user interface APIs for accessing e-RT3 IO module.

A1.1 List of APIs

Table A1.1 List of APIs

Category	Subcategory	Feature	Function name
I/O module	Device access	Read from the input relay in blocks	readM3InRelay
		Read from the input relay	readM3InRelayP
		Read from the output relay in blocks	readM3OutRelay
		Write to the output relay in blocks	writeM3OutRelay
		Write to the output relay	writeM3OutRelayP
		Read 16-bit data from the I/O register	readM3IoRegister
		Read 8-bit data from the I/O register	readM3IoRegisterB
		Read 32-bit data from the I/O register	readM3IoRegisterL
		Write 16-bit data to the I/O register	writeM3IoRegister
		Write 8-bit data to the I/O register	writeM3IoRegisterB
		Write 32-bit data to the I/O register	writeM3IoRegisterL
	Mode configuration	Read from the mode register	readM3IoModeRegister
		Write to the mode register	writeM3IoModeRegister
	Input relay interrupt	Enable interrupts (in all points)	enableM3IoIrq
		Enable interrupts (in one point)	enableM3IoIrqP
		Disable interrupts (in one point)	disableM3IoIrqP
	Module information	Get the module ID	getM3IoName
		Get the mapping address of the I/O space	getM3IoMapAdr
		Get the mapping size of the I/O space	getM3IoMapSize
		Get the offset address of the I/O space in the I/O register	getM3IoDRegAdr
		Get the size of the I/O space in the I/O register	getM3IoDRegSize
		Get the offset address of the I/O space in the input relay	getM3IoXAdr
		Get the size of the I/O space in the input relay	getM3IoXSize
		Get the offset address of the I/O space in the output relay	getM3IoYAdr
		Get the size of the I/O space in the output relay	getM3IoYSize
CPU module	Device access	Read from the CPU device	readM3CpuDevice
		Read from the CPU relay device	readM3CpuDeviceP
		Write to the CPU device	writeM3CpuDevice
		Write to the CPU relay device	writeM3CpuDeviceP
	Signal notification	Enable signal reception	enableM3CpuSignal
		Disable signal reception	disableM3CpuSignal
		Send signals	sendM3CpuSignal
	CPU information	Get the CPU number	getM3CpuNumber
		Get the CPU type	getM3CpuType
		Read from the CPU-shared memory	readM3CpuMemory
		Write to the CPU-shared memory	writeM3CpuMemory
PLC device	Local device	Set local device assignment information	setM3InternalDataTable
		Get local device assignment information	referM3InternalDataTable
		Read from the internal relay in blocks	readM3InternalRelay
		Read from the internal relay	readM3InternalRelayB
		Write to the internal relay in blocks	writeM3InternalRelay
		Write to the internal relay	writeM3InternalRelayB
		Read 16-bit data from the data register	readM3InternalRegister
		Write 16-bit data to the data register	writeM3InternalRegister
	Shared device	Set shared device assignment information	setM3SharedDataConfig
		Get shared device assignment information	referM3SharedDataConfig
		Read from the (extended) shared relay in blocks	readM3SharedRelay
		Read from the (extended) shared relay	readM3SharedRelayB

		Write to the (extended) shared relay in blocks	writeM3SharedRelay
		Write to the (extended) shared relay	writeM3SharedRelayB
		Read 16-bit data from the (extended) shared register	readM3SharedRegister
		Write 16-bit data to the (extended) shared register	writeM3SharedRegister
	Link device	Set link device assignment information	setM3LinkDeviceConfig
		Get link device assignment information	referM3LinkDeviceConfig
		Read from the link relay in blocks	readM3LinkRelay
		Read from the link relay	readM3LinkRelayB
		Write to the link relay in blocks	writeM3LinkRelay
		Write to the link relay	writeM3LinkRelayB
		Read 16-bit data from the link register	readM3LinkRegister
		Write 16-bit data to the link register	writeM3LinkRegister
System administration	Library management	Get the library version	getM3LibVersion
	LED indicator	Set the state of the RUN LED	setM3RunLed
		Get the state of the RUN LED	getM3RunLed
		Set the state of the ALM LED	setM3AlmLed
		Get the state of the ALM LED	getM3AlmLed
		Set the state of the ERR LED	setM3ErrLed
		Get the state of the ERR LED	getM3ErrLed
		Set the state of the U1 LED	setM3U1Led
		Get the state of the U1 LED	getM3U1Led
		Set the state of the U2 LED	setM3U2Led
		Get the state of the U2 LED	getM3U2Led
		Set the state of the U3 LED	setM3U3Led
		Get the state of the U3 LED	getM3U3Led
		Get the state of the MODE switch	getM3ModeSwitch
		Get the battery level	getM3BatteryPower
	Logging	Write a system log message	writeM3log
		Clear all system logs	cleanM3log
RAS	System operation	System reset	setM3Reset
		Failure output	setM3FailOutput
	System monitoring	Sub-unit transmission route diagnosis	getM3FailSubunit
		CPU module diagnosis	getM3FailCpu
	Alarm notification	Enable high CPU temperature detection	enableM3HeatIrq
		Enable momentary power failure detection	enableM3PowerIrq
WDT	Timer operation	Get the WDT	bindM3Wdt
		Release the WDT	releaseM3Wdt
		Clear the WDT	cleanM3Wdt
		Start the WDT	startM3Wdt
		Stop the WDT	stopM3Wdt
	Mode configuration	Set the WDT timeout period	setM3WdtTimeout
		Set the WDT operating mode	setM3WdtMode
		Get the WDT operating mode	getM3WdtMode

A1.2 List of API error codes

This section contains information on error codes specific to APIs.

Table 4.2 List of error codes

Macro name	Error code	Description
S_m3io_MODULE_NOT_FOUND	257	No module is mounted in the specified slot.
S_m3io_INVALID_UNIT	258	The specified unit number is out of range.
S_m3io_INVALID_SLOT	259	The specified slot number is out of range.
S_m3io_INVALID_NUMBER	260	The specified parameter is out of range.
S_m3io_INVALID_FUNC	261	An unexpected IOCTL code was specified.
S_m3io_INVALID_MODULE	262	An unsupported IOCTL code was specified.
S_m3io_DMA_ERROR	263	DMA communication failed.
S_m3io_BUS_ERROR_NR	264	An I/O bus failure occurred.
S_m3io_BUS_ERROR_BR	265	An I/O bus failure occurred.
S_m3io_BUS_ERROR_RDP	266	An I/O bus failure occurred.
S_m3io_BUS_ERROR_MF	267	An I/O bus failure occurred.
S_m3io_BUS_ERROR_DT	268	An I/O bus failure occurred.
S_m3io_INTERNAL_ERROR	357	An internal error occurred.
S_m3cpu_MODULE_NOT_FOUND	157	No module is mounted in the specified slot.
S_m3cpu_INVALID_UNIT	158	The specified unit number is out of range.
S_m3cpu_INVALID_SLOT	159	The specified slot number is out of range.
S_m3cpu_INVALID_NUMBER	160	The specified parameter is out of range.
S_m3cpu_INVALID_FUNC	161	An unexpected IOCTL code was specified.
S_m3cpu_INVALID_MODULE	162	An unsupported IOCTL code was specified.
S_m3cpu_TIMEOUT_ERROR	163	A CPU module does not respond.
S_m3cpu_INTERNAL_ERROR	166	An internal error occurred.
S_m3dev_INVALID_NUMBER	392	The specified parameter is out of range.
S_m3dev_DEVICE_NOT_FOUND	393	The specified device is not found.
S_m3dev_BOUNDARY_ERROR	396	The device alignment is ignored.
S_m3dev_INVALID_FUNC	397	The specified parameter is out of range.
S_m3dev_INTERNAL_ERROR	398	An internal error occurred.
S_m3ras_LRCHK_ERROR	449	An error was found in the sub-unit transmission route.
S_m3ras_CPUCHK_ERROR	450	An error was found in other CPU.
S_m3ras_BUS_ERROR	459	An I/O bus failure occurred.
S_m3ras_INTERNAL_ERROR	460	An internal error occurred.

When an API function returns an error, an error code is stored in the global variable errno. To see the error code, include the errno.h header file in the source code of your application.

A1.3 Receiving interrupts and alarms

Interrupts and alarms work by making use of message queuing (inter-process communication) on Linux. This section describes how this message queuing is implemented.

The following table lists the features that use the message queue.

Table 4.3 Features that use the message queue

Category	Feature	Function name
Input relay interrupt	Enable interrupts (in all points)	enableM3IoIrq
	Enable interrupts (in one point)	enableM3IoIrqP
Signal notification	Enable signal reception	enableM3CpuSignal
Alarm notification	Enable high CPU temperature detection	enableM3HeatIrq
	Enable momentary power failure detection	enableM3PowerIrq

These API functions require a message queue ID as an argument. To get the message queue ID, use the msgget system call. To set the queue to receive a message when an event occurs, specify the message queue ID obtained by the msgget system call for the argument of an API function.
To receive the message in the message queue, use the msgrcv system call. Specify the type (msgtyp) and data structure (mtext) of the message to be received for the argument of the msgrcv system call. The following table lists the type and data structure for the message.

Table 4.4 Type and data structure for the message

Feature	msgtyp macro name (value)	mtext data structure
Enable interrupts (in all points)	M3IO_MSGTYPE_IO (1)	M3IO_MSG_IO
Enable interrupts (in one point)	M3IO_MSGTYPE_IO (1)	M3IO_MSG_IO
Enable signal reception	M3CPU_MSGTYPE_SEQ_EVENT (2)	M3CPU_MSG_SEQ_EVENT
Enable momentary power failure detection	M3RAS_MSGTYPE_FAIL_EVENT (4)	unsigned short
Enable high CPU temperature detection	M3RAS_MSGTYPE_HEAT_ALARM (5)	not used

Note

High CPU temperature detection enables alarm notification when its API function is called, sending an alarm (message) only once if a failure is detected. It does not repeatedly send alarms for high temperatures.

■ System call

This subsection describes system calls used to receive messages in the message queue. The following contains excerpts from the Linux manual (MAN).

msgget

Feature: Get a message queue identifier

Synopsis:
```
#include <sys/msg.h>
int msgget(key_t key, int msgflg);
```

Description: The msgget system call returns the message queue identifier associated with the value of the key argument. If key has the value IPC_PRIVATE or key is not IPC_PRIVATE when no message queue with the given key exists and IPC_CREAT is specified in msgflg, a new message queue is created.

If msgflg specifies both IPC_CREAT and IPC_EXCL and a message queue already exists for key, then msgget fails with errno set to EEXIST.

Argument			
	key	IPC_PRIVATE	/* Private key. */
	msgflg	IPC_CREAT	/* Create key if key does not exist. */
		IPC_EXCL	/* Fail if key exists. */
		IPC_NOWAIT	/* Return error on wait. */

Return value		
	Non-negative integer	Successful
	-1	Failed

errno	
EACCES	A message queue exists for key, but the calling process does not have permission to access the queue, and does not have the CAP_IPC_OWNER capability.
EEXIST	A message queue exists for key and msgflg specified both IPC_CREAT and IPC_EXCL.
ENOENT	No message queue exists for key and msgflg did not specify IPC_CREAT.
ENOMEM	A message queue has to be created but the system does not have enough memory for the new data structure.
ENOSPC	A message queue has to be created but the system limit for the maximum number of message queues (MSGMNI) would be exceeded.

---

msgrcv

Feature: Operate System V message queues

Synopsis:
```
#include <sys/msg.h>
ssize_t msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp, int msgflg);
```

Description: The msgrcv system call is used to receive messages from a System V message queue. The calling process must have read permission to receive a message.
The msgp argument is a pointer to caller-defined structure of the general form below.
The msgrcv system call removes a message from the queue specified by msqid and places it in the buffer pointed to by msgp.
The argument msgsz specifies the maximum size in bytes for the member mtext of the structure pointed to by the msgp argument. If the message text has a length greater than msgsz, then the behavior depends on whether MSG_NOERROR is specified in msgflg. If MSG_NOERROR is specified, then the message text will be truncated (and the truncated part will be lost); if MSG_NOERROR is not specified, then the message is not removed from the queue and the system call fails returning -1 with errno set to E2BIG.

Structure:
```
struct msgbuf {
   long mtype;      /* message type, must be > 0 */
   char mtext[1];   /* message data */
};
```

The mtext field is an array (or other structure) whose size is specified by msgsz, a non-negative integer value. Messages of zero length (that is, no mtext field) are also permitted. The mtype field must have a strictly positive integer value. This value can be used by the receiving process for message selection.

Argument	
msqid	Message queue ID obtained by msgget()
msgp	Pointer to a struct msgbuf buffer in which a message is stored
msgsz	Specifies the maximum size in bytes for the member mtext of the structure pointed to by the msgp argument.
msgtyp	If it is 0, then the first message in the queue is read. If it is greater than 0, then the first message in the queue of type msgtyp is read, unless MSG_EXCEPT was specified in

		msgflg. If MSG_EXCEPT is specified, the first message in the queue of type other than msgtyp will be read. If it is less than 0, then the first message in the queue with the lowest type less than or equal to the absolute value of msgtyp will be read.
	msgflg	The argument is a bit mask constructed by ORing together with zero or more of the following flags: IPC_NOWAIT /* Return error on wait. */ MSG_NOERROR /* no error if message is too big */ MSG_EXCEPT /* recv any msg except of specified type.*/ MSG_COPY /* copy (not remove) all queue messages */
Return value	Non-negative integer	Successful (the number of bytes actually copied into the mtext array is returned.)
	-1	Failed
errno	E2BIG	The message text length is greater than msgsz and MSG_NOERROR is not specified in msgflg.
	EACCES	The calling process does not have read permission on the message queue, and does not have the CAP_IPC_OWNER capability.
	EAGAIN	No message was available in the queue and IPC_NOWAIT was specified in msgflg.
	EFAULT	The address pointed to by msgp is not accessible.
	EIDRM	While the process was sleeping to receive a message, the message queue was removed.
	EINTR	While the process was sleeping to receive a message, the process caught a signal.
	EINVAL	msgqid was invalid, or msgsz was less than 0. Both MSG_COPY and MSG_EXCEPT were specified. Both MSG_COPY and MSG_EXCEPT were specified in msgflg.
	ENOMSG	IPC_NOWAIT was specified in msgflg and no message of the requested type existed on the message queue. IPC_NOWAIT and MSG_COPY were specified in msgflg and the queue contains less than msgtyp messages.
	ENOSYS	MSG_COPY was specified in msgflg, and this kernel was configured without CONFIG_CHECKPOINT_RESTORE.

■ Sample code

● Receiving signal notifications from multiple CPUs

```
#include <stdio.h>
#include <sys/msg.h>
#include "m3lib.h"

int main(int argc, char *argv[])
{
  int msqid;
  struct msgbuf {
    long mtype;
    M3CPU_MSG_SEQ_EVENT mtext;
  } msgp;

  msqid = msgget(IPC_PRIVATE, 0666);
  if (msqid < 0)     return -1;

  if (enableM3CpuSignal(2, msqid))     return -1;
  if (enableM3CpuSignal(3, msqid))     return -1;
```

```
  if (enableM3CpuSignal(4, msqid))     return -1;

  while(1)
  {
    printf("wait for message\n");
    if (msgrcv(msqid, &msgp, sizeof(M3CPU_MSG_SEQ_EVENT),
        M3CPU_MSGTYPE_SEQ_EVENT, MSG_NOERROR) < 0)
      return -1;

    printf("received message = %lx:  slot=%d
data=%04x %c%c%c%c%c%c%c%c\n",
        msgp.mtype,
        msgp.mtext.slot,
        msgp.mtext.data,
        msgp.mtext.sigName[0], msgp.mtext.sigName[1],
        msgp.mtext.sigName[2], msgp.mtext.sigName[3],
        msgp.mtext.sigName[4], msgp.mtext.sigName[5],
        msgp.mtext.sigName[6], msgp.mtext.sigName[7]);

    if (msgp.mtext.slot == 2 && msgp.mtext.data == 100)
      break;
  }

  return 0;
}
```

- **Batch-processing momentary power failure detection and high CPU temperature detection**

```
#include <stdio.h>
#include <sys/msg.h>
#include "m3lib.h"

int main(int argc, char *argv[])
{
  int msqid;
  struct msgbuf {
    long mtype;
    unsigned short mtext;
  } msgp;

  msqid = msgget(IPC_PRIVATE, 0666);
  if (msqid < 0)     return -1;

  if (enableM3HeatIrq(msqid))          return -1;
  if (enableM3PowerIrq(0, msqid))      return -1;
```

```
  while(1)
  {
    printf("wait for message\n");
    if (msgrcv(msqid, &msgp, sizeof(unsigned short), 0,
MSG_NOERROR) < 0)
      return -1;

    switch (msgp.mtype)
    {
    case M3RAS_MSGTYPE_FAIL_EVENT:
      printf("blackout fail:  starus=%d\n", msgp.mtext);
      break;

    case M3RAS_MSGTYPE_HEAT_ALARM:
      printf("thermal runaway\n");
      break;

    default:
      printf("unknown message\n");
      break;
    }
  }

  return 0;
}
```

A1.4 How to receive signals (inter-process communication)

The watch dog timer (WDT) provided by this library has three types of modes, from which you can select a different timeout operation. One of the modes contains the software WDT capability, which can send a SIGTERM signal, defined for Linux inter-process communication, upon timeout. This section describes how to implement code to receive this signal.

Note

The signaling function for inter-process communication is different from the signal notification. The signal notification can synchronize operations among CPU modules in the multi-CPU configuration.
The signals described in this section are part of the Linux-specific functionality, with which processes running on Linux communicate with each other.

For signal reception, a handler is put in place so that a process can run the handler when receiving a signal. The sigaction system call is used to register a handler with a process.
The sigaction system call associates the signal specified in the argument with the pointer to the handler function. After this registration is completed, the process calls the handler whenever receiving a registered signal. In the hander, implement code to deal with a high CPU load.

Note

For details on WDT mode settings provided by e-RT3, refer to the description for the setM3WdtMode function in "A1.5.6 WDT".

■ System call

This subsection describes system calls used in receiving signals. The following contains excerpts from the Linux manual (MAN).

● sigaction

Feature — Examine and change a signal action

Synopsis
```
#include <signal.h>
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

Description — The sigaction system call is used to change the action taken by a process on receipt of a specific signal.

Structure
```
struct sigaction {
    void     (*sa_handler)(int);
    void     (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t   sa_mask;
    int        sa_flags;
    void     (*sa_restorer)(void);
};
```

	On some architecture a union is involved: do not assign to both sa_handler and sa_sigaction.	
Argument	signum	Specifies the signal and can be any valid signal except SIGKILL and SIGSTOP.
	act	If it is non-null, the new action for signal signum is installed from act.
	oldact	If it is non-null, the previous action is saved in oldact.
Return value	0	Successful
	-1	Failed
errno	EFAULT	act or oldact points to memory which is not a valid part of the process address space.
	EINVAL	An invalid signal was specified. This will also be generated if an attempt is made to change the action for SIGKILL or SIGSTOP, which cannot be caught or ignored.

■ Sample code

● Forcing a WDT timeout to occur and receiving SIGTERM signals

```
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include "m3lib.h"

static void handler(int sig)
{
  if (sig == SIGTERM)
    printf("SIGTERM signal is received\n");
}

int main(int argc, char *argv[])
{
  struct sigaction act;
  int timeout, loop;

  timeout = 1000;
  loop = 10;

  act.sa_handler = handler;
  if (sigaction(SIGTERM, (struct sigaction *)&act, NULL) < 0)
    return -1;

  if (bindM3Wdt())   return -1;
  if (setM3WdtTimeout(timeout))  return -1;
  if (setM3WdtMode(M3WDT_MODE_SIG))    return -1;
  if (startM3Wdt())  return -1;

  while(loop--)
```

```
  {
    usleep(timeout * 900);
    if (cleanM3Wdt())return -1;
  }

  pause();

  if (stopM3Wdt())   return -1;
  if (releaseM3Wdt())      return -1;

  return 0;
}
```

A1.5 API reference

A1.5.1 I/O module

■ Device access

● readM3InRelay

Feature	Read from the input relay in blocks	
Synopsis	int readM3InRelay(int unit, int slot, int pos, int num, unsigned short data[4]);	
Description	The function reads from the input relay in an I/O module in 16 points. num data blocks are read from the input relay with device number pos in the I/O module specified by the arguments unit and slot. The pos value must be 1, 17, 33, or 49. If any other value in the range is specified, the value is rounded to a smaller value (for example, 24 is rounded down to 17). The read data is stored in num elements in data[], starting from the first element. The contact statuses for 16 points are stored in an element, starting from the LSB, in the order of input relay numbers.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the input relay number (1, 17, 33, and 49).
	num	Specifies the number of blocks to be read from (1 to 4).
	data[]	Buffer to store the read data
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no input relay was specified.

● readM3InRelayP

Feature	Read from the input relay	
Synopsis	int readM3InRelayP(int unit, int slot, int pos, unsigned short *data);	
Description	The function reads from the input relay in an I/O module in one point. Only one point is read from the input relay with input relay number pos in the I/O module specified by the arguments unit and slot. The value of 1 is stored in data if the input relay is set to ON, and 0 if set to OFF.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the input relay number (1 to 64).
	data	Buffer to store the read data
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no input relay was specified.

---

readM3OutRelay

Feature: Read from the output relay in blocks

Synopsis: int readM3OutRelay(int unit, int slot, int pos, int num, unsigned short data[4]);

Description: The function reads from the output relay in an I/O module in 16 points.
num data blocks are read from the output relay with device number pos in the I/O module specified by the arguments unit and slot.
The pos value must be 1, 17, 33, or 49. If any other value in the range is specified, the value is rounded to a smaller value (for example, 24 is rounded down to 17). The read data is stored in num elements in data[], starting from the first element. The contact statuses for 16 points are stored in an element, starting from the LSB, in the order of input relay numbers.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the output relay number (1, 17, 33, and 49).
num	Specifies the number of blocks to be read from (1 to 4).
data[]	Buffer to store the read data

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no output relay was specified.

---

writeM3OutRelay

Feature: Write to the output relay in blocks

Synopsis: int writeM3OutRelay(int unit, int slot, int pos, int num, unsigned short data[4], unsigned short mask[4]);

Description: The function writes to the output relay in an I/O module in 16 points.
num data blocks are written to the output relay with device number pos in the I/O module specified by the arguments unit and slot.
The pos value must be 1, 17, 33, or 49. If any other value in the range is specified, the value is rounded to a smaller value (for example, 24 is rounded down to 17). The data is stored in data[], starting from the first element to numth element in the array. The contact statuses for 16 points are set in the order of output relay numbers, starting from the LSB. mask[] is the data for masking. Specify the value of 1 for the relay number of the relay to which the data should be written, and of 0 for that of the relay in which the value should be retained. The bits are located just like data[].

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the output relay number (1, 17, 33, and 49).
num	Specifies the number of blocks to be written to (1 to 4).
data[]	Buffer to store the data to be written
mask[]	Buffer to store the data for write masking

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no output relay was specified.

---

writeM3OutRelayP

Feature	Write to the output relay	
Synopsis	int writeM3OutRelayP(int unit, int slot, int pos, unsigned short data);	
Description	The function writes to the output relay in an I/O module in one point. Only 1-point data is written to the output relay with output relay number pos in the I/O module specified by the arguments unit and slot. To set the relay to ON, store 1 in data, and to set it to OFF, store 0.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the output relay number (1 to 64).
	data	Data to be written
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no output relay was specified.

readM3IoRegister

Feature	Read 16-bit data from the I/O register	
Synopsis	int readM3IoRegister(int unit, int slot, int pos, int num, unsigned short *data);	
Description	The function reads 16-bit data from the I/O register in an I/O module. The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data is read and then stored in data.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the I/O register number (from 1).
	num	Specifies how many points of data are read (from 1).
	data	Buffer to store the read data
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no I/O register was specified.

readM3IoRegisterB

Feature: Read 8-bit data from the I/O register

Synopsis: int readM3IoRegisterB (int unit, int slot, int pos, int num, unsigned char *data);

Description: The function reads 8-bit data from the I/O register in an I/O module.
The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data is read and then stored in data.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the I/O register number (from 1).
num	Specifies how many points of data are read (from 1).
data	Buffer to store the read data

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no I/O register was specified.

Remarks: This function is used on modules that handle byte sequences, such as serial communication modules and device net modules.
The register number that is to be set to argument pos can be obtained from the following formula by using register numbers for 16-bit data written in the manual for each module.
I/O register number for 8-bit data = 2 x (I/O register number for 16-bit data - 1) + 1

---

readM3IoRegisterL

Feature: Read 32-bit data from the I/O register

Synopsis: int readM3IoRegisterL (int unit, int slot, int pos, int num, unsigned long *data);

Description: The function reads 32-bit data from the I/O register in an I/O module.
The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data is read and then stored in data.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the I/O register number (from 1).
num	Specifies how many points of data are read (from 1).
data	Buffer to store the read data

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no I/O register was specified.

Remarks: This function is used on modules that handle long word data such as high-speed counter modules.
The register number that is to be set to argument pos can be obtained from the following formula by using register numbers for 16-bit data written in the manual for each module.
I/O register number for 32-bit data = 2 x (I/O register number for 16-bit data + 1) / 2

---

writeM3IoRegister

Feature: Write 16-bit data to the I/O register

Synopsis: int writeM3IoRegister(int unit, int slot, int pos, int num, unsigned short *data);

Description: The function writes 16-bit data to the I/O register in an I/O module.
The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data to be written is stored in data.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the I/O register number (from 1).
num	Specifies how many points of data are written (from 1).
data	Buffer to store the data to be written

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no I/O register was specified.

---

writeM3IoRegisterB

Feature: Write 8-bit data to the I/O register

Synopsis: int writeM3IoRegisterB(int unit, int slot, int pos, int num, unsigned char *data);

Description: The function writes 8-bit data to the I/O register in an I/O module.
The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data to be written is stored in data.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the I/O register number (from 1).
num	Specifies how many points of data are written (from 1).
data	Buffer to store the data to be written

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_NUMBER	An invalid parameter was specified.
S_m3io_INVALID_MODULE	A module with no I/O register was specified.

Remarks: This function is used on modules that handle byte sequences, such as serial communication modules and device net modules.
The register number that is to be set to argument pos can be obtained from the following formula by using register numbers for 16-bit data written in the manual for each module.
I/O register number for 8-bit data = 2 x (I/O register number for 16-bit data - 1) + 1

● writeM3IoRegisterL

Feature	Write 32-bit data to the I/O register	
Synopsis	int writeM3IoRegisterL(int unit, int slot, int pos, int num, unsigned long *data);	
Description	The function writes 32-bit data to the I/O register in an I/O module. The I/O module is specified in the arguments unit and slot, and the I/O register and the data range are specified in pos and num. The data to be written is stored in data.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the I/O register number (from 1).
	num	Specifies how many points of data are written (from 1).
	data	Buffer to store the data to be written
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no I/O register was specified.
Remarks	This function is used on modules that handle long word data such as high-speed counter modules. The register number that is to be set to argument pos can be obtained from the following formula by using register numbers for 16-bit data written in the manual for each module. I/O register number for 32-bit data = 2 x (I/O register number for 16-bit data + 1) / 2	

■ Mode configuration

● readM3IoModeRegister

Feature	Read from the mode register	
Synopsis	int readM3IoModeRegister(int unit, int slot, int pos, int num, unsigned short mode[8]);	
Description	The function reads 16-bit data from the mode register in an I/O module. The I/O module is specified in the arguments unit and slot, and the mode register and the data range are specified in pos and num. The data is read and stored in mode[]. The maximum number of data points is eight.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the mode register number (from 1).
	num	Specifies how many points of data are read (1 to 8).
	mode []	Buffer to store the read data
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no mode register was specified.
Remarks	- The function to read from the mode register is an API to configure DIO modules. - The size of the mode register varies depending on the module. - To configure an advanced module, work with the I/O register.	

Note

For details on the register map and setting values of the mode register, refer to "4.4 Mode register access" of BSP Common Function Manual (IM 34M06M52-02E).

● writeM3IoModeRegister

Feature	Write to the mode register	
Synopsis	int writeM3IoModeRegister(int unit, int slot, int pos, int num, unsigned short mode[8]);	
Description	The function writes 16-bit data to the mode register in an I/O module. The I/O module is specified in the arguments unit and slot, and the mode register and the data range are specified in pos and num. The data to be written is stored in mode[]. The maximum number of data points is eight.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
	pos	Specifies the mode register number (from 1).
	num	Specifies how many points of data are written (1 to 8).
	mode[]	Buffer to store the data to be written
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3io_INVALID_NUMBER	An invalid parameter was specified.
	S_m3io_INVALID_MODULE	A module with no mode register was specified.
Remarks	- The function to read from the mode register is an API to configure DIO modules. - The size of the mode register varies depending on the module. - To configure an advanced module, work with the I/O register.	

Note

For details on the register map and setting values of the mode register, refer to "4.4 Mode register access" of BSP Common Function Manual (IM 34M06M52-02E).

■ Input relay interrupt

● enableM3IoIrq

Feature
Enable interrupts (in all points)

Synopsis
int enableM3IoIrq (int unit, int slot, unsigned short mask[4], int msgQId);

Description
The function enables or disables interrupts for all the points from the input relay in an I/O module.
The I/O module is specified in the arguments unit and slot. The data for interrupt masking is set in the argument mask[] in 16 points, storing the possibilities of interrupts in the array starting from the first bit, ordered by the lowest input relay number.

mask[0]	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
mask[1]	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17
mask[2]	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33
mask[3]	64	63	62	61	60	59	58	57	56	55	54	53	52	51	50	49

If the corresponding bit is set to 1, interrupts from the input relay are enabled, and if set to 0, they are disabled.
To disable I/O interrupts, specify 0 for all the data for interrupt masking, or specify -1 for the message queue ID.

The msgrcv system call receives the interrupts from I/O modules by using the message queue ID registered by the argument msgQId.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
mask[]	Pointer of unsigned short type to store the data for interrupt masking
msgQId	Message queue ID obtained by the msgget system call

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_MODULE	A module with no interrupt support was specified.

Remarks
- The message queue ID is overwritten every time the function is called.
- A message queue ID of 0 or more is valid.
- Messages in a message queue cannot be received by multiple processes. A single process must be responsible for receiving the messages.
- If any message remains unread in the message queue when the interrupts are disabled, the messages are sent when an interrupt request is received next time interrupts are enabled.

● enableM3IoIrqP

Feature: Enable interrupts (in one point)

Synopsis: int enableM3IoIrqP (int unit, int slot, int pos, int msgQId);

Description: The function enables interrupts from the input relay in the I/O module specified by the argument in one point.
The msgrcv system call receives the interrupts from I/O modules by using the message queue ID registered by the argument msgQId.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the input relay number (1 to 32).
msgQId	Message queue ID obtained by the msgget system call

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_MODULE	A module with no interrupt support was specified.

Remarks
- The message queue ID is overwritten every time the function is called.
- A message queue ID of 0 or more is valid.
- Messages in a message queue cannot be received by multiple processes. A single process must be responsible for receiving the messages.
- If any message remains unread in the message queue when the interrupts are disabled, the messages are sent when an interrupt request is received next time interrupts are enabled.

● disableM3IoIrqP

Feature: Disable interrupts (in one point)

Synopsis: int disableM3IoIrqP (int unit, int slot, int pos);

Description: The function disables interrupts from the input relay in the I/O module specified by the argument in one point.
Disabling interrupts from all input relays also clears the registered message queue ID.

Argument

unit	Specifies the unit number (0 to 7).
slot	Specifies the slot number (1 to 16).
pos	Specifies the input relay number (1 to 32).

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3io_INVALID_UNIT	An invalid unit number was specified.
S_m3io_INVALID_SLOT	An invalid slot number was specified.
S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.
S_m3io_INVALID_MODULE	A module with no interrupt support was specified.

Remarks: The ability or inability to support interrupts in an I/O module is contained in the manual for each module.
If any message remains unread in the message queue when interrupts are disabled, the messages are sent when an interrupt request is received next time interrupts are enabled.

■ Module information

● getM3IoName

Feature	Get the module ID	
Synopsis	char* getM3IoName (int unit, int slot);	
Description	The function obtains the module ID of an I/O module. It can get the module ID (module model name) of the I/O module specified in the arguments unit and slot. A pointer to the string of four ASCII characters, followed by '\0', is returned as a return value, and NULL is returned if getting the module name fails.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Not NULL	Module model name with four ASCII characters, including '\0' as the fifth character
	NULL	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_INVALID_SLOT	An invalid slot number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.

● getM3IoMapAdr

Feature	Get the mapping address of the I/O space	
Synopsis	int getM3IoMapAdr (int unit, int slot);	
Description	The function obtains the mapping address for the I/O space in an I/O module. As a return value, it can get the address of the I/O space mapping register in the I/O module specified in the arguments unit and slot. The address ranges from 0x0 to 0x40000.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	I/O space mapping address
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

● getM3IoMapSize

Feature	Get the mapping size of the I/O space	
Synopsis	int getM3IoMapSize (int unit, int slot);	
Description	The function obtains the mapping size of the I/O space in an I/O module. It can get the I/O space mapping size of the I/O module specified in the arguments unit and slot. The size ranges from 0x0 to 0x8000.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	I/O space mapping size
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.
	S_m3io_MODULE_NOT_FOUND	No module exists in the specified slot.

---

getM3IoDRegAdr

Feature	Get the offset address of the I/O space in the I/O register	
Synopsis	int getM3IoDRegAdr (int unit, int slot);	
Description	The function obtains the address of the I/O register in an I/O module as an offset from the beginning of the mapping area. As a return value, it can get mapping information of the I/O register in the I/O module specified in the arguments unit and slot.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Offset address of the I/O space in the I/O register
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

---

getM3IoDRegSize

Feature	Get the size of the I/O space in the I/O register	
Synopsis	int getM3IoDRegSize (int unit, int slot);	
Description	The function obtains the size of the area where the I/O register in an I/O module is located. You can determine the existence or non-existence of the I/O register by checking this size. The size can be obtained in bytes. To get the number of points in the I/O register, divide the size by 2 to convert it into the value in words.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Size of the I/O space in the I/O register
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

---

getM3IoXAdr

Feature	Get the offset address of the I/O space in the input relay	
Synopsis	int getM3IoXAdr (int unit, int slot);	
Description	The function obtains the address of the input relay in an I/O module as an offset from the beginning of the mapping area. As a return value, it can get mapping information of the input relay in the I/O module specified in the arguments unit and slot.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Offset address of the I/O space in the input relay
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

---

getM3IoXSize

Feature	Get the size of the I/O space in the input relay	
Synopsis	int getM3IoXSize (int unit, int slot);	
Description	The function obtains the size of the area where the input relay in an I/O module is located. You can determine the existence or non-existence of the input relay by checking this size. The size can be obtained in bytes. To get the number of points in the input relay, multiply the size by 8 to convert it into the value in bits.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Size of the I/O space in the input relay
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

---

getM3IoYAdr

Feature	Get the offset address of the I/O space in the output relay	
Synopsis	int getM3IoYAdr (int unit, int slot);	
Description	The function obtains the address of the output relay in an I/O module as an offset from the beginning of the mapping area. As a return value, it can get mapping information of the output relay in the I/O module specified in the arguments unit and slot.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Offset address of the I/O space in the output relay
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

---

getM3IoYSize

Feature	Get the size of the I/O space in the output relay	
Synopsis	int getM3IoYSize (int unit, int slot);	
Description	The function obtains the size of the area where the output relay in an I/O module is located. You can determine the existence or non-existence of the output relay by checking this size. The size can be obtained in bytes. To get the number of points in the output relay, first divide the size by 2 considering the data for masking, and then multiply it by 8 to convert it into the value in bits.	
Argument	unit	Specifies the unit number (0 to 7).
	slot	Specifies the slot number (1 to 16).
Return value	Positive number	Size of the I/O space in the output relay
	-1	Error
errno	EFAULT	The function failed to get data.
	S_m3io_INVALID_UNIT	An invalid unit number was specified.

A1.5.2 CPU module

■ Device access

Note

For details on device types specified for CPU device access and error codes stored in response commands, refer to "4.5 CPU device access" of BSP Common Function Manual (IM 34M06M52-02E).

● readM3CpuDevice

Feature	Read from the CPU device	
Synopsis	int readM3CpuDevice(int cpuno, int type, int pos, int num, unsigned short *data);	
Description	The function reads 16-bit data from the CPU device in a CPU module. The argument cpuno specifies a CPU module, type specifies a device type, and pos and num specify the range in the device. The data is read and stored in the argument data. The accessible range in each device varies depending on the CPU module. If the error code is set to EIO when the function fails (the return value is -1), the code indicates that an error is returned from the CPU module to be operated. In this case, a detailed error code is stored in data[0].	
Argument	cpuno	Specifies the CPU number (1 to 4).
	type	Device type
	pos	Specifies the number of the start device to be read (from 1).
	num	Number of points in the device to be read from (1 to 256)
	data	Pointer to the unsigned short type buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	ENOMEM	The system memory is running low.
	EFAULT	The function failed to get data.
	EIO	The function received an error response command from the CPU module.
	S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
	S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
	S_m3cpu_TIMEOUT_ERROR	The CPU module did not respond within the prescribed period.
Remarks	- The upper limit of points in the device cannot be exceeded. - The function cannot be used for the special relay M. - Make sure to access the special register Z on a 1-point basis.	

Note

For details on device types and error codes stored in response commands, refer to "4.5 CPU device access" of BSP Common Function Manual (IM 34M06M52-02E).

● readM3CpuDeviceP

Feature	Read from the CPU relay device	
Synopsis	int readM3CpuDeviceP(int cpuno, int type, int pos, int num, unsigned short *data);	
Description	The function reads the specified number of points of data from the CPU device in other CPU module. The argument cpuno specifies a CPU module, type specifies a device type, and pos and num specify the range in the device. The data is read and stored in the argument data. The accessible range in each device varies depending on the CPU module. The values in the relay are stored in the argument data in device number order, starting from the LSB to the MSB. The data array needs ((number of points - 1)/16 + 1) elements. If the error code is set to EIO when the function fails (the return value is -1), the code indicates that an error is returned from the CPU module to be operated. In this case, a detailed error code is stored in data[0].	
Argument	cpuno	Specifies the CPU number (1 to 4).
	type	Device type
	pos	Specifies the number of the start device to be read (from 1).
	num	Number of points in the device to be read from (1 to 256)
	data	Pointer to the unsigned short type buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	ENOMEM	The system memory is running low.
	EFAULT	The function failed to get data.
	EIO	The function received an error response command from the CPU module.
	S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
	S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
	S_m3cpu_TIMEOUT_ERROR	The CPU module did not respond within the prescribed period.
Remarks	- The upper limit of points in the device cannot be exceeded. - Make sure to access the special relay M on a 1-point basis. - The function cannot be used for any register device.	

Note

For details on device types and error codes stored in response commands, refer to "4.5 CPU device access" of BSP Common Function Manual (IM 34M06M52-02E).

writeM3CpuDevice

Feature: Write to the CPU device

Synopsis: int writeM3CpuDevice(int cpuno, int type, int pos, int num, unsigned short *data, unsigned short *error);

Description: The function writes 16-bit data to the CPU device in a CPU module.
The argument cpuno specifies a CPU module, type specifies a device type, and pos and num specify the range in the device. The data to be written is stored in the argument data.
The accessible range in each device varies depending on the CPU module.

If the error code is set to EIO when the function fails (the return value is -1), the code indicates that an error is returned from the CPU module to be operated. In this case, an error code in the response command is stored in error.

Argument

cpuno	Specifies the CPU number (1 to 4).
type	Device type
pos	Specifies the number of the start device to be written to (from 1).
num	Number of points in the device to be written to (1 to 256)
data	Pointer to the unsigned short type buffer in which the data to be written is stored
error	Error code in the response command

Return value

0	Successful
-1	Error

errno

ENOMEM	The system memory is running low.
EFAULT	The function failed to get data.
EIO	The function received an error response command from the CPU module.
S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
S_m3cpu_TIMEOUT_ERROR	The CPU module did not respond within the prescribed period.

Remarks
- The upper limit of points in the device cannot be exceeded.
- The function cannot be used for the special relay M.
- Make sure to access the special register Z on a 1-point basis.

Note

For details on device types and error codes stored in response commands, refer to "4.5 CPU device access" of BSP Common Function Manual (IM 34M06M52-02E).

writeM3CpuDeviceP

Feature: Write to the CPU relay device

Synopsis: int writeM3CpuDeviceP(int cpuno, int type, int pos, int num, unsigned short *data, unsigned short *error);

Description: The function writes the specified number of points of data to the PLC relay device in a CPU module.
The argument cpuno specifies a CPU module, type specifies a device type, and pos and num specify the range in the device. The data to be written is stored in the argument data.
The accessible range in each device varies depending on the CPU module.
The values in the relay are stored in the argument data in device number order, starting from the LSB to the MSB. The data array needs ((number of points - 1)/16 + 1) elements.

If the error code is set to EIO when the function fails (the return value is -1), the code indicates that an error is returned from the CPU module to be operated. In this case, an error code in the response command is stored in error.

Argument

cpuno	Specifies the CPU number (1 to 4).
type	Device type
pos	Specifies the number of the start device to be written to (from 1).
num	Number of points in the device to be written to (1 to 256)
data	Pointer to the unsigned short type buffer in which the data to be written is stored
error	Error code in the response command

Return value

0	Successful
-1	Error

errno

ENOMEM	The system memory is running low.
EFAULT	The function failed to get data.
EIO	The function received an error response command from the CPU module.
S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
S_m3cpu_TIMEOUT_ERROR	The CPU module did not respond within the prescribed period.

Remarks
- The upper limit of points in the device cannot be exceeded.
- Make sure to access the special relay M on a 1-point basis.
- The function cannot be used for any register device.

Note

For details on device types and error codes stored in response commands, refer to "4.5 CPU device access" of BSP Common Function Manual (IM 34M06M52-02E).

■ Signal notification

● enableM3CpuSignal

Feature: Enable signal reception

Synopsis: int enableM3CpuSignal (int cpuno, int msgQId);

Description: The function enables receiving signal notifications used to synchronize operations among CPU modules.
Signal notifications sent from CPU modules are received through a Linux message queue. Thus, your applications must pre-register the message queue for reception.
This message-queue registration for reception of signal notifications can be done by this function. The argument cpuno specifies the number of the slot in the CPU module that receives the signal notifications.
The msgrcv system call receives the signals by using the message queue ID registered by the argument msgQId.

Argument

cpuno	Specifies the CPU number (1 to 4).
msgQId	Message queue ID obtained by the msgget system call

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3cpu_INVALID_SLOT	An invalid slot number was specified.
S_m3cpu_INVALID_MODULE	A module with no interrupt support was specified.

Remarks
- The message queue ID is overwritten every time the function is called.
- A message queue ID of 0 or more is valid.
- Messages in a message queue cannot be received by multiple processes. A single process must be responsible for receiving the messages.
- If any message remains unread in the message queue when the interrupts are disabled, the messages are sent when an interrupt request is received next time interrupts are enabled.

● disableM3CpuSignal

Feature: Disable signal reception

Synopsis: int disableM3CpuSignal (int cpuno);

Description: The function disables receiving signal notifications used to synchronize operations among CPU modules.
Signal notifications sent from CPU modules are received through a Linux message queue. The message queue for reception created in advance must be registered with your application via the enableM3CpuSignal function.
This function releases the registered message queue to stop receiving signals. For argument cpuno, specify the slot number in the CPU module that sends (was sending) signal notifications.

Argument

cpuno	Specifies the CPU number (1 to 4).

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3cpu_INVALID_SLOT	An invalid slot number was specified.
S_m3cpu_INVALID_MODULE	A module with no interrupt support was specified.

Remarks: If any message remains unread in the message queue when interrupts are disabled, the messages are sent when an interrupt request is received next time interrupts are enabled.

● sendM3CpuSignal

Feature	Send signals	
Synopsis	int sendM3CpuSignal(int cpuno, char signal[8], unsigned short data);	
Description	The function sends signal notifications. They are sent to e-RT3 2.0 CPU modules specified in the argument cpuno. A signal notification consists of 8-byte signal signal[] and 1-word data data.	
Argument	cpuno	Specifies the CPU (slot) number (1 to 4) for the signal destination.
	signal[]	Registers a name of eight ASCII characters or less.
	data	Specifies 1-word data.
Return value	0	Successful
	-1	Error
errno	ENOMEM	The system memory is running low.
	EFAULT	The function failed to get data.
	S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
	S_m3cpu_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3cpu_INVALID_MODULE	An invalid module was specified.
	S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
Remarks	Do not use this function for sequence CPU modules with no capability of receiving signal notifications.	

■ CPU information

● getM3CpuNumber

Feature	Get the CPU number	
Synopsis	int getM3CpuNumber (void);	
Description	The function obtains the slot number of the slot (1 to 4) in which this CPU module is mounted. It returns the slot number as a return value.	
Argument	None	
Return value	1 to 4	CPU number (slot number)
	-1	Error
errno	EFAULT	The function failed to get data.

● getM3CpuType

Feature: Get the CPU type

Synopsis: int getM3CpuType (int type[4]);

Description: The function obtains the CPU types of all CPU modules mounted in the system.
The CPU types of the CPU modules with CPU numbers 1 through 4 are stored in the argument type[]. The following table shows the relationship between the types and values of the obtained CPU types:

CPU type	Macro name	Value
Not a CPU module	M3CPU_TYPE_NON	0
Sequence CPU	M3CPU_TYPE_SEQ	1
BASIC CPU	M3CPU_TYPE_BASIC	2
AT-compatible CPU	M3CPU_TYPE_AT	3
e-RT3CPU	M3CPU_TYPE_RTOS	4

Argument	type[]	Pointer to the int type array in which CPU-type values are stored.
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.

● readM3CpuMemory

Feature: Read from the CPU-shared memory

Synopsis: int readM3CpuMemory(int cpuno, int pos, int num, unsigned short *buf);

Description: The function directly reads values from the CPU-shared memory.
The data is read directly from the shared memory area in the CPU module specified in the argument cpuno to buf.
The maximum size of the data is 8.5 KB. Specify the offset not exceeding the limit in words for pos and num.

Argument	cpuno	Specifies the CPU number (1 to 4).
	pos	Specifies the offset in the shared memory in words.
	num	Specifies the number of data sets in the shared memory.
	buf	Pointer to the unsigned short type array in which the read data is stored
Return value	0	Successful
	-1	Error
errno	ENOMEM	The system memory is running low.
	EFAULT	The function failed to get data.
	S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
	S_m3cpu_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3cpu_INVALID_MODULE	An invalid module was specified.
	S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
	S_m3cpu_INTERNAL_ERROR	An internal error occurred.

Note

For details on the CPU-shared memory, refer to "5.2.1 Shared memory access" of BSP Common Function Manual (IM 34M06M52-02E).

● writeM3CpuMemory

Feature	Write to the CPU-shared memory	
Synopsis	int writeM3CpuMemory(int cpuno, int pos, int num, unsigned short *buf);	
Description	The function directly writes values to the CPU-shared memory. The data in buf is written directly to the shared memory area in the CPU module specified in the argument cpuno. The maximum size of the data is 8.5 KB. Specify the offset not exceeding the limit in words for pos and num.	
Argument	cpuno	Specifies the CPU number (1 to 4).
	pos	Specifies the offset in the shared memory in words.
	num	Specifies the number of data sets written to the shared memory.
	buf	Pointer to the unsigned short type array in which the data to be written is stored
Return value	0	Successful
	-1	Error
errno	ENOMEM	The system memory is running low.
	EFAULT	The function failed to get data.
	S_m3cpu_INVALID_SLOT	An invalid CPU number was specified.
	S_m3cpu_MODULE_NOT_FOUND	No module exists in the specified slot.
	S_m3cpu_INVALID_MODULE	An invalid module was specified.
	S_m3cpu_INVALID_NUMBER	An invalid parameter was specified.
	S_m3cpu_INTERNAL_ERROR	An internal error occurred.

Note

For details on the CPU-shared memory, refer to "5.2.1 Shared memory access" of BSP Common Function Manual (IM 34M06M52-02E).

A1.5.3 PLC device

■ Local device

● setM3InternalDataTable

Feature	Set local device assignment information	
Synopsis	int setM3InternalDataTable (unsigned int location, unsigned int relaySize, unsigned int registerSize);	
Description	The function sets the number of data points in the local devices. It sets relaySize for the number of data points in the internal relay and registerSize for the number of data points in the data register. Specify a multiple of 32 for the number of relay data points and a multiple of 2 for the number of register data points.	
Argument	location	Location for the local device 0: SDRAM (kernel space) 1: User SRAM (effective only for F3RP71-2L)
	relaySize	Number of points in the internal relay (0, or 32 or more)
	registerSize	Number of points in the data register (0, or 2 or more)
Return value	0	Successful
	Other than 0	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
	S_m3dev_DEVICE_NOT_FOUND	The device specified in location is not found.

referM3InternalDataTable

Feature	Get local device assignment information	
Synopsis	int referM3InternalDataTable(unsigned int *relaySize, unsigned int *registerSize);	
Description	The function obtains the number of data points in the local devices. It stores the number of data points in the internal relay in relaySize and the number of data points in the data register in registerSize.	
Argument	relaySize	Pointer to the buffer in which the number of points in the internal relay is stored
	registerSize	Pointer to the buffer in which the number of points in the data register is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.

readM3InternalRelay

Feature	Read from the internal relay in blocks	
Synopsis	int readM3InternalRelay (int no, int num, unsigned short *pBuff);	
Description	The function reads from the internal relay in 16 points. It reads num points of data sets from the device with device number no in the internal relay "I" into pBuff. For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the internal relay to be read (from 1)
	num	Number of blocks in the internal relay to be read (from 1)
	pBuff	Pointer to the buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	The upper limit specified during configuration cannot be exceeded.	

readM3InternalRelayB

Feature	Read from the internal relay	
Synopsis	int readM3InternalRelayB (int no, int num, unsigned char *data);	
Description	The function reads from the internal relay in one point. It reads num points of data sets from the device with device number no in the internal relay "I" into data. For the data pointer, reserve an array of num elements.	
Argument	no	Start number of the internal relay to be read (from 1)
	num	Number of points in the internal relay to be read (from 1).
	data	Pointer to the buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	The upper limit specified during configuration cannot be exceeded.	

readM3InternalRegister

Feature	Read 16-bit data from the data register	
Synopsis	int readM3InternalRegister (int no, int num, unsigned short *pBuff);	
Description	The function reads 16-bit data from the data register. It reads num points of data sets from the device with device number no in the data register "D" into pBuff. For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the data register to be read (from 1)
	num	Number of points in the data register to be read (from 1)
	pBuff	Pointer to the buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	The upper limit specified during configuration cannot be exceeded.	

writeM3InternalRelay

Feature	Write to the internal relay in blocks	
Synopsis	int writeM3InternalRelay (int no, int num, unsigned short *pBuff);	
Description	The function writes to the internal relay in 16 points. It writes num points of data sets in pBuff to the device starting from device number no in the internal relay "I". For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the internal relay to be written to (from 1)
	num	Number of blocks in the internal relay to be written to (from 1)
	pBuff	Pointer to the buffer in which the data to be written is stored
Return value	0	Successful
	Other than 0	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	The upper limit specified during configuration cannot be exceeded.	

writeM3InternalRelayB

Feature	Write to the internal relay	
Synopsis	int writeM3InternalRelayB (int no, int num, unsigned char *data);	
Description	The function writes to the internal relay in one point. It writes num points of data sets in data to the device starting from device number no in the internal relay "I". For the data pointer, reserve an array of num elements.	
Argument	no	Start number of the internal relay to be written to (from 1)
	num	Number of points in the internal relay to be written to (from 1)
	data	Pointer to the buffer in which the data to be written is stored
Return value	0	Successful
	Other than 0	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	The upper limit specified during configuration cannot be exceeded.	

● writeM3InternalRegister

Feature
Write 16-bit data to the data register

Synopsis
int writeM3InternalRegister (int no, int num, unsigned short *pBuff);

Description
The function writes 16-bit data to the data register.
It writes num points of data sets in pBuff to the device starting from device number no in the data register "D". For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the data register to be written to (from 1)
num	Number of points in the data register to be written to (from 1)
pBuff	Pointer to the buffer in which the data to be written is stored

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.

Remarks
The upper limit specified during configuration cannot be exceeded.

■ Shared device

● setM3SharedDataConfig

Feature
Set shared device assignment information

Synopsis

```
int setM3SharedDataConfig
        (
        LPM3SHDDATACONFIG shdCnf,
        LPM3SHDDATACONFIG extShdCnf
        );
```

Description
The function assigns shared devices to CPU modules.
Each index number of the member variables wNumberOfRelay[] and wNumberOfRegister[] in the structure for shared device assignment information corresponds to "CPU number - 1".
Specify a multiple of 32 for the number of relay data points and a multiple of 2 for the number of register data points.

The following table shows the ranges that the parameter can accept:

Device name	Total points of all CPUs	Specify in
Shared relay	0 to 2048 points	32 points
Extended shared relay	0 to 2048 points	32 points
Shared register	0 to 1024 points	2 points
Extended shared register	0 to 3072 points	2 points

Structure

```
typedef struct tagM3SHDDATACONFIG {
        unsigned short wNumberOfRelay[4];
        unsigned short wNumberOfRegister[4];
) M3SHDDATACONFIG, *LPM3SHDDATACONFIG;
```

Argument

shdCnf.wNumberOfRelay[]	Points in the shared relay that are assigned to CPU1 to 4
shdCnf.wNumberOfRegister[]	Points in the shared register that are assigned to CPU1 to 4
extShdCnf.wNumberOfRelay[]	Points in the extended shared relay that are assigned to CPU1 to 4
extShdCnf.wNumberOfRegister[]	Points in the extended shared register that are assigned to CPU1 to 4

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.

Remarks
Configure all CPU modules so that they use the same settings.

● referM3SharedDataConfig

Feature: Get shared device assignment information

Synopsis:
```
int referM3SharedDataConfig
        (
        LPM3SHDDATACONFIG shdCnf,
        LPM3SHDDATACONFIG extShdCnf
        );
```

Description: The function obtains the configuration of the shared device assigned to each CPU module. Each index number of the member variables wNumberOfRelay[] and wNumberOfRegister[] in the structure for shared device assignment information corresponds to "CPU number - 1".

Structure:
```
typedef struct tagM3SHDDATACONFIG {
        unsigned short wNumberOfRelay[4];
        unsigned short wNumberOfRegister[4];
} M3SHDDATACONFIG, *LPM3SHDDATACONFIG;
```

Argument

shdCnf.wNumberOfRelay[]	Points in the shared relay that are assigned to CPU1 to 4
shdCnf.wNumberOfRegister[]	Points in the shared register that are assigned to CPU1 to 4
extShdCnf.wNumberOfRelay[]	Points in the extended shared relay that are assigned to CPU1 to 4
extShdCnf.wNumberOfRegister[]	Points in the extended shared register that are assigned to CPU1 to 4

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.

● readM3SharedRelay

Feature: Read from the (extended) shared relay in blocks

Synopsis: int readM3SharedRelay (int no, int num, unsigned short *pBuff);

Description: The function reads from the (extended) shared relay in 16 points.
It reads num points of data sets from the device with device number no in the (extended) shared relay "E" into pBuff. Calling the function once can read from all the shared relays. For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the (extended) shared relay to be read from (shared relay: 1 to 2048, extended shared relay: 2049 to 4096)
num	Number of blocks in the (extended) shared relay to be read (from 1)
pBuff	Pointer to the buffer in which the read data is stored

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_BOUNDARY_ERROR	The start number of the device is invalid; the boundary is incorrect.
S_m3dev_INTERNAL_ERROR	An I/O bus failure occurred.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check.
- Access across CPU areas requires word boundaries not to be crossed.
- Access across the shared relay and extended shared relay areas is not possible.

---

readM3SharedRelayB

Feature: Read from the (extended) shared relay

Synopsis: int readM3SharedRelayB (int no, int num, unsigned char *data);

Description: The function reads from the (extended) shared relay in one point.
It reads num points of data sets from the device with device number no in the (extended) shared relay "E" into data. Calling the function once can read from all the shared relays. For the data pointer, reserve an array of num elements.

Argument

no	Start number of the (extended) shared relay to be read from (shared relay: 1 to 2048, extended shared relay: 2049 to 4096)
num	Number of points in the (extended) shared relay to be read (from 1)
data	Pointer to the buffer in which the read data is stored

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_BOUNDARY_ERROR	The start number of the device is invalid; the boundary is incorrect.
S_m3dev_INTERNAL_ERROR	An I/O bus failure occurred.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check.
- Access across CPU areas requires word boundaries not to be crossed.
- Access across the shared relay and extended shared relay areas is not possible.

---

readM3SharedRegister

Feature: Read 16-bit data from the (extended) shared register

Synopsis: int readM3SharedRegister (int no, int num, unsigned short *pBuff);

Description: The function reads 16-bit data from the (extended) shared register.
It reads num points of data sets from the device with device number no in the (extended) shared register "R" into pBuff. Calling the function once can read from all the shared registers. For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the (extended) shared register to be read from (shared register: 1 to 1024, extended shared register: 1025 to 4096)
num	Number of points in the (extended) shared register to be read (from 1)
pBuff	Pointer to the buffer in which the read data is stored

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_INTERNAL_ERROR	An I/O bus failure occurred.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check.
- Access across the shared register and extended shared register areas is not possible.

---

writeM3SharedRelay

Feature: Write to the (extended) shared relay in blocks

Synopsis: int writeM3SharedRelay (int no, int num, unsigned short *pBuff);

Description: The function writes to the (extended) shared relay in 16 points.
It writes num points of data sets in pBuff to the device starting from device number no in the (extended) shared relay "E". Calling the function once can write to all the shared relays. For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the (extended) shared relay to be written to (shared relay: 1 to 2048, extended shared relay: 2049 to 4096)
num	Number of blocks in the (extended) shared relay to be written to (from 1)
pBuff	Pointer to the buffer in which the data to be written is stored

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_BOUNDARY_ERROR	The start number of the device is invalid; the boundary is incorrect.
S_m3dev_INTERNAL_ERROR	An I/O bus failure occurred.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check.
- Access across CPU areas requires word boundaries not to be crossed.
- Access across the shared relay and extended shared relay areas is not possible.

---

writeM3SharedRelayB

Feature: Write to the (extended) shared relay

Synopsis: int writeM3SharedRelayB (int no, int num, unsigned char *data);

Description: The function writes to the (extended) shared relay in one point.
It writes num points of data sets in data to the device starting from device number no in the (extended) shared relay "E". Calling the function once can write to all the shared relays. For the data pointer, reserve an array of num elements.

Argument

no	Start number of the (extended) shared relay to be written to (shared relay: 1 to 2048, extended shared relay: 2049 to 4096)
num	Number of points in the (extended) shared relay to be written to (from 1)
data	Pointer to the buffer in which the data to be written is stored

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_BOUNDARY_ERROR	The start number of the device is invalid; the boundary is incorrect.
S_m3dev_INTERNAL_ERROR	An I/O bus failure occurred.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check.
- Access across CPU areas requires word boundaries not to be crossed.
- Access across the shared relay and extended shared relay areas is not possible.

● writeM3SharedRegister

Feature	Write 16-bit data to the (extended) shared register	
Synopsis	int writeM3SharedRegister (int no, int num, unsigned short *pBuff);	
Description	The function writes 16-bit data to the (extended) shared register. It writes num points of data sets in pBuff to the device starting from device number no in the (extended) shared register "R". Calling the function once can write to all the shared registers. For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the (extended) shared register to be written to (shared register: 1 to 1024, extended shared register: 1025 to 4096)
	num	Number of points in the (extended) shared register to be written to (from 1)
	pBuff	Pointer to the buffer in which the data to be written is stored
Return value	0	Successful
	Other than 0	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
Remarks	- The upper limit specified during configuration cannot be exceeded. - The parameter check follows the configuration. Any difference between the setting and the CPU module configuration does not cause an error in the parameter check. - Access across the shared register and extended shared register areas is not possible.	

■ Link device

● referM3LinkDeviceConfig

Feature	Get link device assignment information	
Synopsis	int referM3LinkDeviceConfig(LPM3LINKDATACONFIG linkCnf);	
Description	The function obtains the number of data points in the link device. Each index number of the member variables wNumberOfRelay[] and wNumberOfRegister[] in the structure for link device assignment information corresponds to "system number - 1".	
Structure	typedef struct tagM3LINKDATACONFIG { unsigned short wNumberOfRelay[8]; unsigned short wNumberOfRegister[8]; } M3LINKDATACONFIG, *LPM3LINKDATACONFIG;	
Argument	linkCnf.wNumberOfRelay[]	Number of points that are assigned to system numbers 1 to 8 in the link relay
	linkCnf.wNumberOfRegister[]	Number of points that are assigned to system numbers 1 to 8 in the link register
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
Remarks	The system numbers 3 through 8 are system-reserved numbers.	

● setM3LinkDeviceConfig

Feature
Set link device assignment information

Synopsis
int setM3LinkDeviceConfig(LPM3LINKDATACONFIG linkCnf);

Description
The function sets the number of data points in the link devices.
Each index number of the member variables wNumberOfRelay[] and wNumberOfRegister[] in the structure for link device assignment information corresponds to "system number - 1". Specify a multiple of 16 for both the number of relay data points and the number of register data points.

The following table shows the ranges that the parameter can accept:

Device name	Total points of all CPUs	Specify in
Link relay	0 to 8192 points	16 points
Link register	0 to 8192 points	16 points

A single CPU module can have up to two systems assigned to it. Thus, the CPU module can have a maximum of 16384 points of link devices.

Structure

```
typedef struct tagM3LINKDATACONFIG {
        unsigned short wNumberOfRelay[8];
        unsigned short wNumberOfRegister[8];
} M3LINKDATACONFIG, *LPM3LINKDATACONFIG;
```

Argument

linkCnf.wNumberOfRelay[]	Number of points that are assigned to system numbers 1 to 8 in the link relay
linkCnf.wNumberOfRegister[]	Number of points that are assigned to system numbers 1 to 8 in the link register

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.

Remarks
The system numbers 3 through 8 are system-reserved numbers.

● readM3LinkRelay

Feature
Read from the link relay in blocks

Synopsis
int readM3LinkRelay (int no, int num, unsigned short *pBuff);

Description
The function reads from the link relay in 16 points.
It reads num points of data sets from the device with device number no in the link relay "L" into pBuff. For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the link relay to be read from (n0001 to n8192: n represents system number - 1)
num	Number of blocks in the link relay to be read (from 1)
pBuff	Pointer to the buffer in which the read data is stored

Return value

0	Successful
-1	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The system numbers 3 through 8 are system-reserved numbers.

---

readM3LinkRelayB

Feature	Read from the link relay	
Synopsis	int readM3LinkRelayB (int no, int num, unsigned char *data);	
Description	The function reads from the link relay in one point. It reads num points of data sets from the device with device number no in the link relay "L" into data. For the data pointer, reserve an array of num elements.	
Argument	no	Start number of the link relay to be read from (n0001 to n8192: n represents system number - 1)
	num	Number of points in the link relay to be read (from 1)
	data	Pointer to the buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
	S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.
Remarks	- The upper limit specified during configuration cannot be exceeded. - The system numbers 3 through 8 are system-reserved numbers.	

---

readM3LinkRegister

Feature	Read 16-bit data from the link register	
Synopsis	int readM3LinkRegister (int no, int num, unsigned short *pBuff);	
Description	The function reads 16-bit data from the link register. It reads num points of data sets from the device with device number no in the link register "W" into pBuff. Access across the systems is not possible. For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the link register to be read from (n0001 to n8192: n represents system number - 1)
	num	Number of points in the link register to be read (from 1)
	pBuff	Pointer to the buffer in which the read data is stored
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
	S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.
Remarks	- The upper limit specified during configuration cannot be exceeded. - The system numbers 3 through 8 are system-reserved numbers.	

---

writeM3LinkRelay

Feature
Write to the link relay in blocks

Synopsis
int writeM3LinkRelay (int no, int num, unsigned short *pBuff);

Description
The function writes to the link relay in 16 points.
It writes num points of data sets in pBuff to the device starting from device number no in the link relay "L". Access across the systems is not possible. For the pBuff pointer, reserve an array of num elements.

Argument

no	Start number of the link relay to be written to (n0001 to n8192: n represents system number - 1)
num	Number of blocks in the link relay to be written to (from 1)
pBuff	Pointer to the buffer in which the data to be written is stored

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The system numbers 3 through 8 are system-reserved numbers.

---

writeM3LinkRelayB

Feature
Write to the link relay

Synopsis
int writeM3LinkRelayB (int no, int num, unsigned char *data);

Description
The function writes to the link relay in one point.
It writes num points of data sets in data to the device starting from device number no in the link relay "L". Access across the systems is not possible. For the data pointer, reserve an array of num elements.

Argument

no	Start number of the link relay to be written to (n0001 to n8192: n represents system number - 1)
num	Number of points in the link relay to be written to (from 1)
data	Pointer to the buffer in which the data to be written is stored

Return value

0	Successful
Other than 0	Error

errno

EFAULT	The function failed to get data.
ENOMEM	The function failed to reserve the working area.
S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.

Remarks
- The upper limit specified during configuration cannot be exceeded.
- The system numbers 3 through 8 are system-reserved numbers.

---

writeM3LinkRegister

Feature	Write 16-bit data to the link register	
Synopsis	int writeM3LinkRegister (int no, int num, unsigned short *pBuff);	
Description	The function writes 16-bit data to the link register. It writes num points of data sets in pBuff to the device starting from device number no in the link register "W". Access across the systems is not possible. For the pBuff pointer, reserve an array of num elements.	
Argument	no	Start number of the link register to be written to (n0001 to n8192: n represents system number - 1)
	num	Number of points in the link register to be written to (from 1)
	pBuff	Pointer to the buffer in which the data to be written is stored
Return value	0	Successful
	Other than 0	Error
errno	EFAULT	The function failed to get data.
	ENOMEM	The function failed to reserve the working area.
	S_m3dev_INVALID_NUMBER	An invalid parameter was specified.
	S_m3dev_DEVICE_NOT_FOUND	The specified device is not found.
Remarks	- The upper limit specified during configuration cannot be exceeded. - The system numbers 3 through 8 are system-reserved numbers.	

A1.5.4 System administration

■ Library management

● getM3LibVersion

Feature	Get the library version	
Synopsis	char *getM3LibVersion(void);	
Description	The function returns libm3 version information as a string.	
Argument	None	
Return value	String	It returns a string that consists of the library version number and the build time.
errno	None	

■ LED indicator

● setM3RunLed

Feature	Set the state of the RUN LED	
Synopsis	int setM3RunLed(int led);	
Description	The function sets the on/off state of the RUN LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the RUN LED 0: Off Not 0: On
Return value	0 -1	Successful Error
errno	EFAULT	The function failed to write data.

● getM3RunLed

Feature	Get the state of the RUN LED	
Synopsis	int getM3RunLed(void);	
Description	The function gets the on/off state of the RUN LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1 0 -1	On Off Error
errno	EFAULT	The function failed to get data.

---

setM3AlmLed

Feature	Set the state of the ALM LED	
Synopsis	int setM3AlmLed(int led);	
Description	The function sets the on/off state of the ALM LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the ALM LED 0: Off Not 0: On
Return value	0 -1	Successful Error
errno	EFAULT	The function failed to write data.

getM3AlmLed

Feature	Get the state of the ALM LED	
Synopsis	int getM3AlmLed(void);	
Description	The function gets the on/off state of the ALM LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1 0 -1	On Off Error
errno	EFAULT	The function failed to get data.

setM3ErrLed

Feature	Set the state of the ERR LED	
Synopsis	int setM3ErrLed(int led);	
Description	The function sets the on/off state of the ERR LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the ERR LED 0: Off Not 0: On
Return value	0 -1	Successful Error
errno	EFAULT	The function failed to write data.

● getM3ErrLed

Feature	Get the state of the ERR LED	
Synopsis	int getM3ErrLed (void);	
Description	The function gets the on/off state of the ERR LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1	On
	0	Off
	-1	Error
errno	EFAULT	The function failed to get data.

● setM3U1Led

Feature	Set the state of the U1 LED	
Synopsis	int setM3U1Led(int led);	
Description	The function sets the on/off state of the U1 LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the U1 LED 0: Off Not 0: On
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to write data.

● getM3U1Led

Feature	Get the state of the U1 LED	
Synopsis	int getM3U1Led(void);	
Description	The function gets the on/off state of the U1 LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
	1	On
Return value	0	Off
	-1	Error
errno	EFAULT	The function failed to get data.

setM3U2Led

Feature	Set the state of the U2 LED	
Synopsis	int setM3U2Led(int led);	
Description	The function sets the on/off state of the U2 LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the U2 LED 0: Off Not 0: On
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to write data.

getM3U2Led

Feature	Get the state of the U2 LED	
Synopsis	int getM3U2Led(void);	
Description	The function gets the on/off state of the U2 LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1	On
	0	Off
	-1	Error
errno	EFAULT	The function failed to get data.

setM3U3Led

Feature	Set the state of the U3 LED	
Synopsis	int setM3U3Led(int led);	
Description	The function sets the on/off state of the U3 LED at the top front of a CPU module to the value specified in the argument led.	
Argument	led	State of the U3 LED 0: Off Not 0: On
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to write data.

● getM3U3Led

Feature	Get the state of the U3 LED	
Synopsis	int getM3U3Led(void);	
Description	The function gets the on/off state of the U3 LED at the top front of a CPU module. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1 0 -1	On Off Error
errno	EFAULT	The function failed to get data.

● getM3ModeSwitch

Feature	Get the state of the MODE switch	
Synopsis	int getM3ModeSwitch (void);	
Description	The function gets the state (number) of the MODE switch at the front of a CPU module. The number is obtainable as a return value of the function.	
Argument	None	
Return value	0 to F -1	MODE switch number Error
errno	EFAULT	The function failed to get data.

● getM3BatteryPower

Feature	Get the battery level	
Synopsis	int getM3BatteryPower (void);	
Description	The function checks the remaining capacity of the RTC and SRAM backup battery. This action is equivalent to checking the state of the BAT LED. The state can be obtained as a return value of the function.	
Argument	None	
Return value	1 0 -1	Low battery voltage (BAT LED is on) Normal battery voltage (BAT LED is off) Error
errno	EFAULT EIO	The function failed to get data. Access failed due to a device operation conflict.
Remarks	Checking the battery level slightly consumes the battery power. To avoid unnecessary consumption due to highly frequent checks, the hardware checks the battery level at about 1-hour intervals, and this function returns the results of these checks by the hardware. It takes time for the function to respond. If an EIO error is returned, try again later.	

■ Logging

● writeM3log

Feature	Write a system log message	
Synopsis	int writeM3log(char msg[128]);	
Description	The function writes a maximum of 128 ASCII characters at a time to the system log area. The available ASCII code range is from 0x20 to 0x7E, excluding the codes for characters, such as a carriage return and line feed.	
Argument	msg	Pointer to the string Avoid writing a string of more than 128 characters.
Return value	0 -1	Successful Error
errno	EFAULT EINVAL	The function failed to get data. An invalid message was written.
Remarks	For F3RP71-2L, messages are written to the non-volatile memory area.	

● cleanM3log

Feature	Clear all system logs	
Synopsis	int cleanM3log (void);	
Description	The function clears all the messages in the system log.	
Argument	None	
Return value	0	The function always returns 0.
errno	None	

A1.5.5 RAS

■ System operation

● setM3Reset

Feature	System reset	
Synopsis	int setM3Reset(void);	
Description	The function resets the entire e-RT3 system. It resets all I/O and CPU modules mounted in the main unit and sub-units, and then restarts the CPU module on which the function is run.	
Argument	None	
Return value	-1	Error
errno	EACCES	The function was run on an add-on CPU module.
Remarks	Avoid using the function on an add-on CPU module.	

● setM3FailOutput

Feature	Failure output	
Synopsis	int setM3FailOutput(int cpuno);	
Description	The function switches the FAIL terminal on the power supply module, notifying the outside of the system of an error. You can use this function when you concluded that the operation can no longer be continued, such as the occurrence of a fatal error in the system program. When a failure is output, outputs from the output relay in the I/O module change according to the external output setting on failure. In the multi-CPU configuration, outputs only from the I/O modules assigned to the CPU with the CPU number specified in the argument cpuno change. In e-RT3 2.0 CPU modules, the assignment of an I/O module to a CPU module is determined by use or non-use of any I/O module. In the single CPU configuration, cpuno is ignored, and outputs from output relays in all I/O modules change according to the setting.	
Argument	cpuno	CPU number of the CPU module on which failure outputs are enabled
Return value	0 -1	Successful Error
errno	EINVAL	An invalid CPU number was specified.

Note

For details on how to assign an I/O module to a CPU module and how to configure whether an I/O module is used, refer to the manual for each CPU module.

■ System monitoring

● getM3FailSubunit

Feature	Sub-unit transmission route diagnosis	
Synopsis	int getM3FailSubunit (unsigned short *change, unsigned short *disconnect, unsigned short position[8]);	
Description	The function obtains the state of a sub-unit transmission route (optical FA bus or optical FA bus 2 connection) upon a sub-unit configuration. Use the function if an error, such as an I/O bus error, occurs during access to an I/O module in the sub-unit. If an error occurs on a sub-unit transmission route, an error type is stored in the arguments change and disconnect, and an error location is stored in the argument position[]. position[] represents an error-detected bit position on F3LR0□, which is set to 1. In each bit, the LSB represents slot number 1, and the MSB represents slot number 16. An index of the array represents a unit number. Example: for position[2] = 0x0020, the unit number is 2 and the slot number is 6.	
Argument	change	Switching of the sub-unit transmission route 0: A sub-unit transmission route has not been switched. 1: A sub-unit transmission route has been switched.
	disconnect	Sub-unit transmission route error 0: A sub-unit transmission route error has not occurred. 1: A sub-unit transmission route error has occurred.
	position[]	Location in which the error-generated module is mounted Index of the array: unit number (0 to 7) Bit position in each array element: slot number (1 to 16)
Return value	0 -1	Successful Error
errno	EFAULT S_m3ras_BUS_ERROR	The function failed to get data. The function failed to access the FA bus module.
Remarks	The sub-unit transmission route error indicates an instantaneous value when this function is carried out. If a failure on the sub-unit transmission route is fixed, the error is also cleared.	

● getM3FailCpu

Feature	CPU module diagnosis	
Synopsis	int getM3FailCpu (int mode);	
Description	The function allows you to check whether other CPU modules, not the CPU module on which the function is run, fail in the multi-CPU configuration. Use the function if an error, such as an I/O bus access error, occurs during access to a shared device. If any of the other CPU modules are found to have an error, the function returns the results with the lower four bits of the return value. The result value of 0 indicates that no error CPU module is found. If an error occurs, an error flag is set bit-by-bit from the LSB in CPU number order. Bit0: CPU1 status (0: OK, 1: Error) Bit1: CPU2 status (0: OK, 1: Error) Bit2: CPU3 status (0: OK, 1: Error) Bit3: CPU4 status (0: OK, 1: Error) If the argument mode is set to a value other than 0, the output setting for the I/O module assigned to an error CPU module is changed according to the external output setting on failure. In e-RT3 2.0 CPU modules, the assignment of an I/O module to a CPU module is determined by use or non-use of any I/O module.	
Argument	mode	DO output condition in case of a CPU failure detection 0: The output setting is changed according to the external output setting on failure. Not 0: The output setting is changed according to the external output setting on failure.
Return value	Positive number -1	CPU status Error
errno	EFAULT S_m3ras_BUS_ERROR S_m3ras_CPUCHK_ERROR	The function failed to get data. The function failed to access the CPU module. The function verified that the CPU module had failed (normal behavior).
Remarks	The function does not switch the FAIL terminal on the power supply module.	

■ System notification

● enableM3HeatIrq

Feature	Enable high CPU temperature detection	
Synopsis	int enableM3HeatIrq (int msgQId);	
Description	The function enables temperature monitoring and notification of a processor equipped with a CPU module. High processor temperature notification always monitors the processors for high temperature due to the surrounding environment or system failure, and notifies your application of high temperatures, if detected. The msgrcv system call receives the notified high temperature of the CPU by using the message queue ID registered by the argument msgQId.	
Argument	msgQId	Message queue ID obtained by the msgget system call
Return value	0 -1	Successful Error
errno	EINVAL	An invalid message queue ID was specified.
Remarks	- The message queue ID is overwritten every time the function is called. - A message queue ID of 0 or more is valid. - Messages in a message queue cannot be received by multiple processes. A single process must be responsible for receiving the messages. - When the function notifies a high CPU temperature, the notification function gets disabled.	

● enableM3PowerIrq

Feature	Enable momentary power failure detection	
Synopsis	int enableM3PowerIrq (int mode, int msgQId);	
Description	The function enables monitoring and notification of a momentary supplied voltage failure or low voltage. The momentary power failure detection detects a failure of the voltage supplied from the power supply module, and notifies the user application of the failure. In the argument mode, specify either the standard mode or the immediate detection mode as the detection mode of the supplied voltage. The msgrcv system call receives a detected momentary power failure by using the message queue ID registered by the argument msgQId.	
Argument	mode	Detection mode of momentary power failure 0: standard mode Not 0: immediate detection mode
	msgQId	Message queue ID obtained by the msgget system call
Return value	0	Successful
	-1	Error
errno	EINVAL	An invalid message queue ID was specified.
Remarks	- The message queue ID is overwritten every time the function is called. - A message queue ID of 0 or more is valid. - Messages in a message queue cannot be received by multiple processes. A single process must be responsible for receiving the messages.	

A1.5.6 WDT

■ Timer operation

● bindM3Wdt

Feature	Get the WDT	
Synopsis	int bindM3Wdt(void);	
Description	The function obtains the right to use the WDT. Only one process can use the WDT at one time. To use the WDT, any process must obtain the right to use the WDT by using this function.	
Argument	None	
Return value	0	Successful
	-1	Error
errno	EBUSY	The WDT is already in use.

● releaseM3Wdt

Feature	Release the WDT	
Synopsis	int releaseM3Wdt(void);	
Description	The function releases the right to use the WDT obtained by the bindM3Wdt function. Only one process can use the WDT timer.	
Argument	None	
Return value	0	Successful
	-1	Error
errno	EBADF	The WDT has not been obtained.

cleanM3Wdt

Feature	Clear the WDT	
Synopsis	int cleanM3Wdt(void);	
Description	The function clears the WDT counter. Call this function periodically at shorter intervals than that specified for the WDT to avoid a WDT timeout.	
Argument	None	
Return value	0 -1	Successful Error
errno	EIO	The WDT is stopped.

startM3Wdt

Feature	Start the WDT	
Synopsis	int startM3Wdt(void);	
Description	The function starts the WDT. The WDT runs based on the specified mode and timeout period.	
Argument	None	
Return value	0 -1	Successful Error
errno	EIO	The WDT is already in operation.

stopM3Wdt

Feature	Stop the WDT	
Synopsis	int stopM3Wdt(void);	
Description	The function stops the WDT.	
Argument	None	
Return value	0 -1	Successful Error
errno	EIO	The WDT is stopped.

■ Mode configuration

● setM3WdtTimeout

Feature: Set the WDT timeout period

Synopsis: setM3WdtTimeout(int timeout);

Description: The function specifies the timeout period for the WDT.
The initial timeout period is set to 5,000 milliseconds when the WDT is started. Use timeout to change this timeout period of the WDT. timeout can be from 1,000 to 120,000 milliseconds. You can change the timeout period at any time.

Argument	timeout	WDT timeout period
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	EINVAL	An invalid period was specified.

Remarks: The period for the WDT is rounded up at the value obtained from dividing the HZ macro value by 10. The default value of the HZ macro is 1000. Although you can change the HZ macro value with the kernel configuration, set the HZ macro to a value of at least 100 based on the previous reason.

● setM3WdtMode

Feature: Set the WDT operating mode

Synopsis: int setM3WdtMode(int mode);

Description: The function sets the operating mode of the WDT with the argument mode.
For mode, set a logical addition of M3WDT_MODE_RESET, M3WDT_MODE_HALT, M3WDT_MODE_SIG, and M3WDT_MODE_CLOSESTOP macros. The default value after the system restart is M3WDT_MODE_RESET.

Macro name	Value	Description
M3WDT_MODE_RESET	0x0000	Rests the WDT when it times out.
M3WDT_MODE_HALT	0x0001	Halts the WDT when it times out. The RDY LED lights off, and the I/O module becomes inaccessible. In the multi-CPU configuration, other CPU modules recognize this CPU module as a failed CPU.
M3WDT_MODE_SIG	0x0002	Works as a software WDT, causing no actions like RESET and HALT shown in the above. When the WDT times out, a SIGTERM signal is sent to the process that changed the mode to this mode. If the WDT is closed, no signal notification is sent.
M3WDT_MODE_CLOSESTOP	0x8000	An option to prevent the RESET operation from being performed when the WDT driver is closed by the OS for any reason. This option causes the WDT to be stopped upon closing. The WDT continues to work by default, so that it can run based on the mode setting after closing.

Argument	mode	WDT operating mode
Return value	0	Successful
	-1	Error
errno	EFAULT	The function failed to get data.
	EIO	The WDT is already in operation.

getM3WdtMode

Feature	Get the WDT operating mode	
Synopsis	int getM3WdtMode(void);	
Description	The function obtains the operating mode of the WDT. It returns a logical addition of macros, which is described in the setM3WdtMode function, as a return value of the function.	
Argument	None	
Return value	Positive number -1	WDT operating mode Error
errno	EFAULT	The function failed to write data.

Appendix2 Web Maintenance Tool

A2.1 Before Use

A2.1.1 Overview

The Web Maintenance Tool provides monitoring and setting functions for the internal parameters of the F3RP70-2L, CPU modules. This tool can be used from the client PC via a Web browser such as Microsoft Internet Explorer or Google Chrome. Therefore, engineers who perform maintenance and start-up and end users who don't have a development environment can easily perform settings and maintenance from a Web browser, regardless of their PC environment.
Apache2 is installed in the CPU module's system, enabling a Web server, and the Web Maintenance Tool is built into the CPU module. Therefore, there is no need to install dedicated software into your PC.
Also, you can view the PDF data such as manuals via your browser.

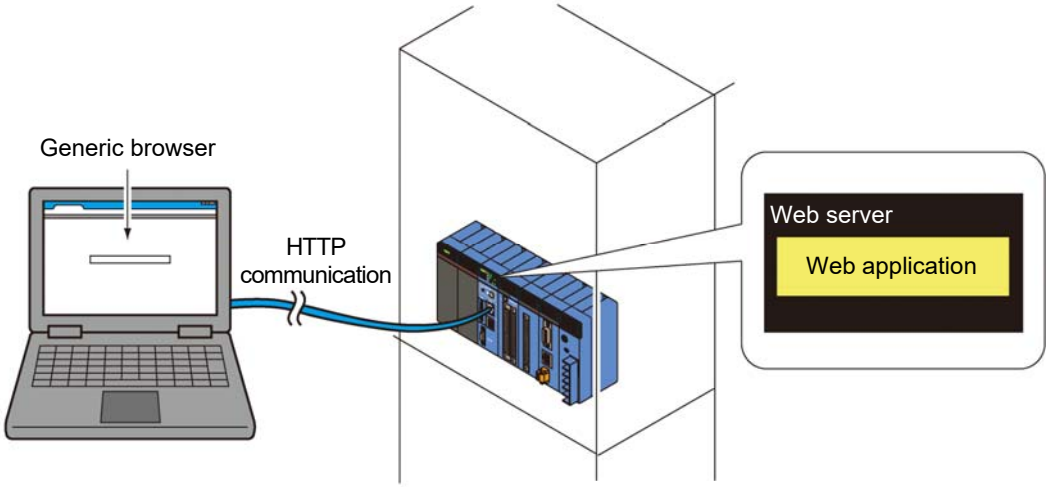


Figure A2.1.1 Overview

A2.1.2 Operating environment

The following describes the operating environment of the Web Maintenance Tool.

Table A2.1.1 Operating environment

Item	Specification
Supported browsers	Generic browsers such as Google Chrome
PC	PC where a browser listed above operates properly
OS	Platform OS where a browser listed above operates properly
Supported CPU modules	F3RP70-2L
Communication conditions	PC that can start up a browser listed above, and network environment where an Ethernet cable can be connected to the port on the front of the CPU module

Note

This application is optimized for operation on Internet Explorer on a Windows PC. If using it in a different execution environment, the display images may vary

depending on the screen resolution and the installation format.

A2.1.3 Setup and start-up

(1) Make sure that the CPU module's power is on.

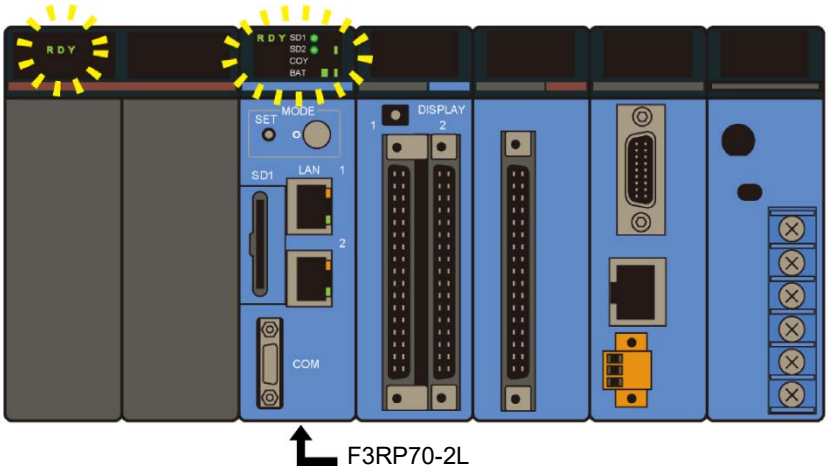


Figure A2.1.2 CPU module operating status

(2) Use a LAN cable to connect the PC to the Port 1 on the CPU module, then start up the browser.

Note

Use Port 1 during setup.

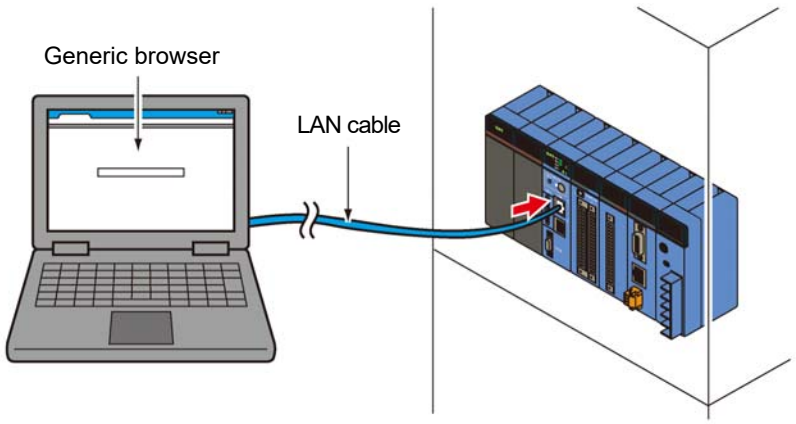


Figure A2.1.3 PC and CPU module connection

(3) Input the IP address "192.168.3.72" in the browser's address bar.

Note

The IP address (192.168.3.72) is set by default.

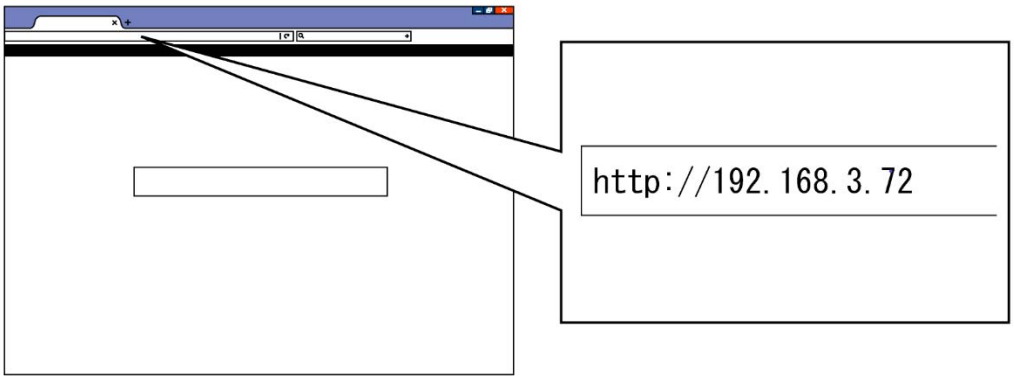


Figure A2.1.4 Inputting the IP address

The e-RT3 Plus Studio screen is displayed.

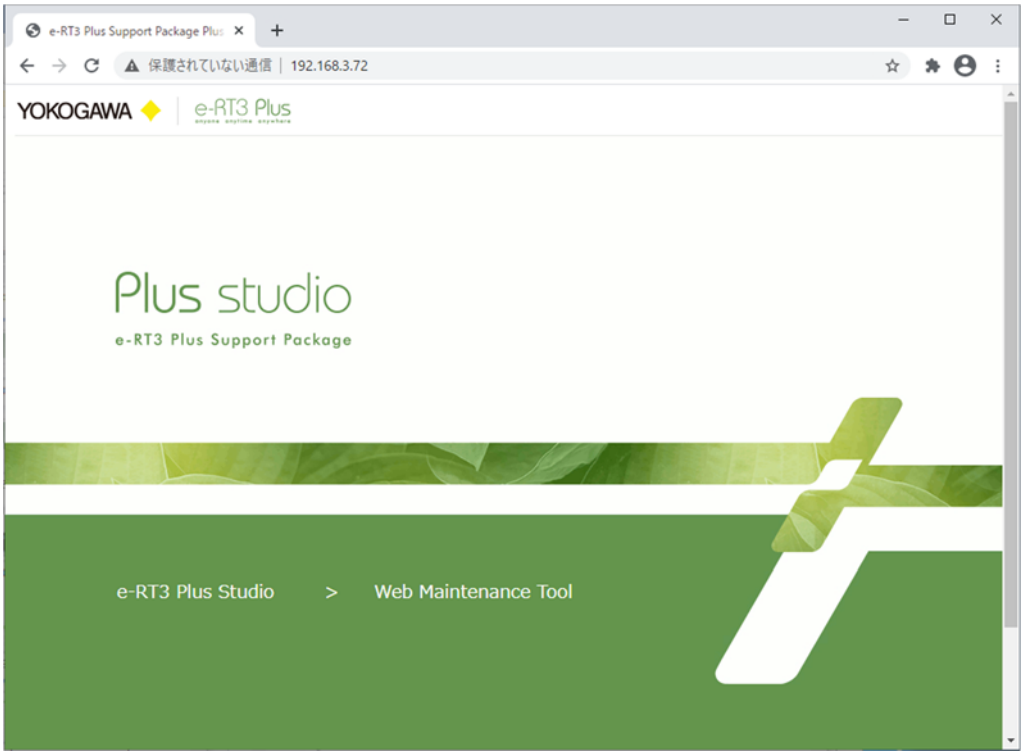


Figure A2.1.5 e-RT3 Plus Studio screen

(4) Click [Web Maintenance Tool].
The Web Maintenance Tool starts up and displays the main screen.

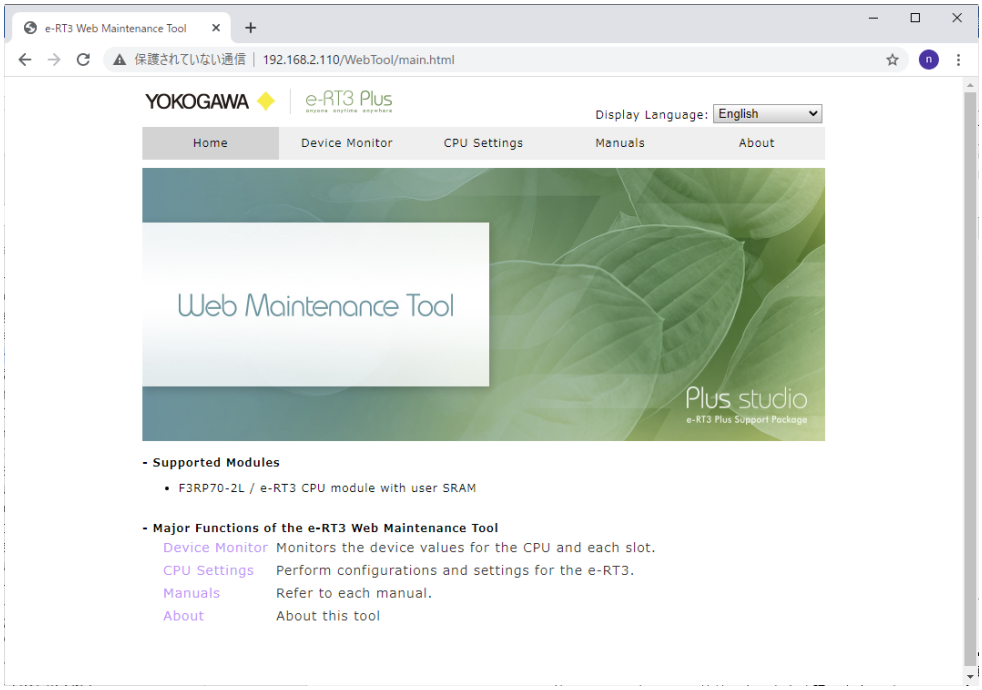


Figure A2.1.6 Main screen

Note

If the Web Maintenance Tool does not start up, check the PC's Ethernet port settings, and use the ping command to make sure you can connect to the CPU module. (The IP address for Ethernet Port 2 is not set.)
If the CPU module is starting up, wait a bit and then input the IP address "192.168.3.72" again.

A2.2 Screen configuration and basic functions

A2.2.1 List of functions

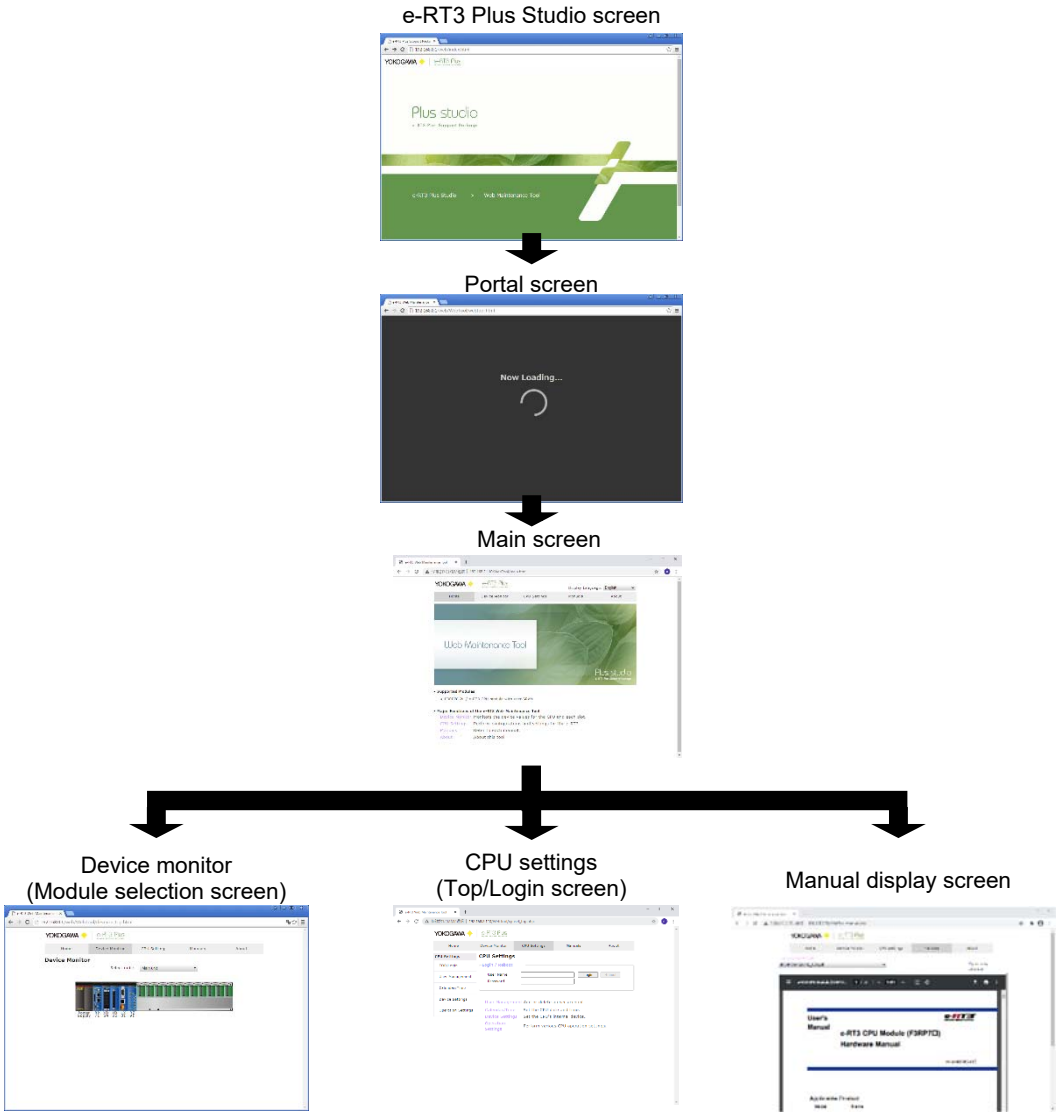


Figure A2.2.1 Screen configuration

The following describes the screen configuration for the Web Maintenance Tool.

Table A2.2.1 Screen configuration

Screen		Description
e-RT3 Plus Studio screen		Top page for the e-RT3 Plus Support Package.
Portal screen		Displayed while the Web Maintenance Tool is initialized.
Main screen		Main screen of the Web Maintenance Tool.
Device monitor (Module selection screen)		Top screen for selecting the modules to be monitored in the device monitor. LED information is also displayed.
	CPU module monitor screen	Monitors the CPU module devices.
	I/O device monitor screen	Monitors the I/O module devices.
CPU settings (Top/Login screen)		Top screen for CPU settings. This is also the screen for users to log into the Web Maintenance Tool.
	User management screen	Adds, deletes, or changes the password for Web Maintenance Tool users.
	Calendar / Time settings screen	Sets the date and time for the CPU module.
	Device settings screen	Sets the internal devices, links, and shared devices used by the CPU module.
	Operation settings screen	Sets the functional operation of peripheral services for the CPU module.
Manual		Displays the manuals.
About		Displays information such as the version.

A2.2.2 Portal screen (Start-up screen)

This screen is displayed when the PC is connected to the CPU module and the Web Maintenance Tool is starting up.

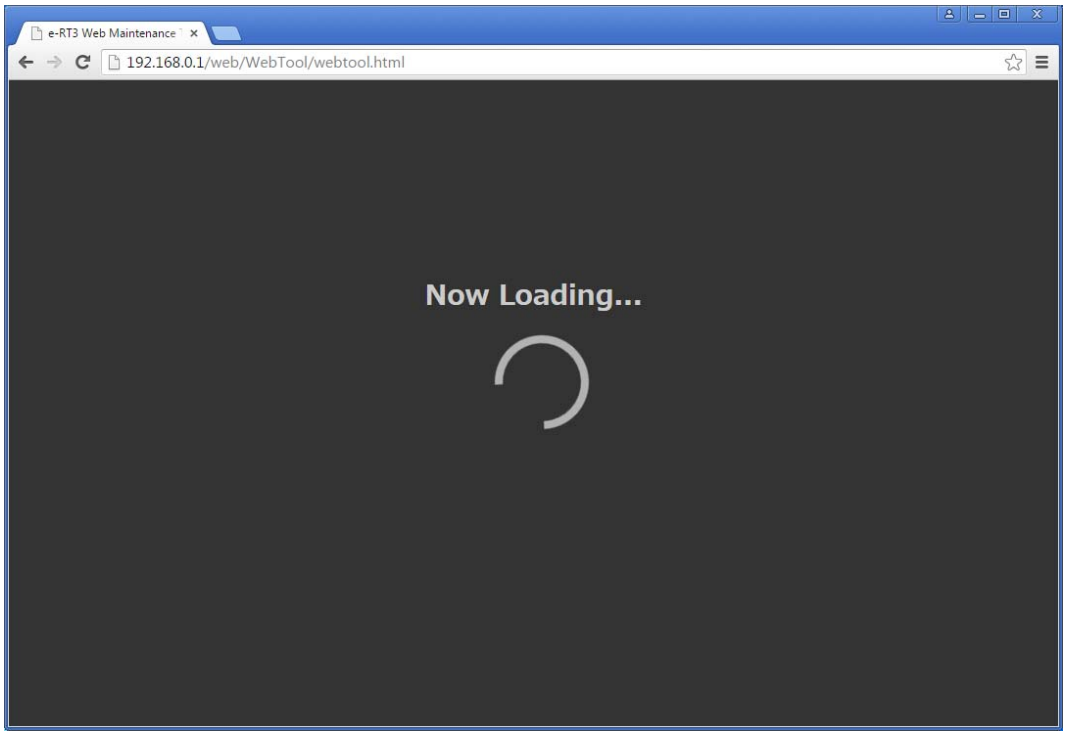


Figure A2.2.2 Portal screen

A2.2.3 Main screen

This is the top page for the Web Maintenance Tool.

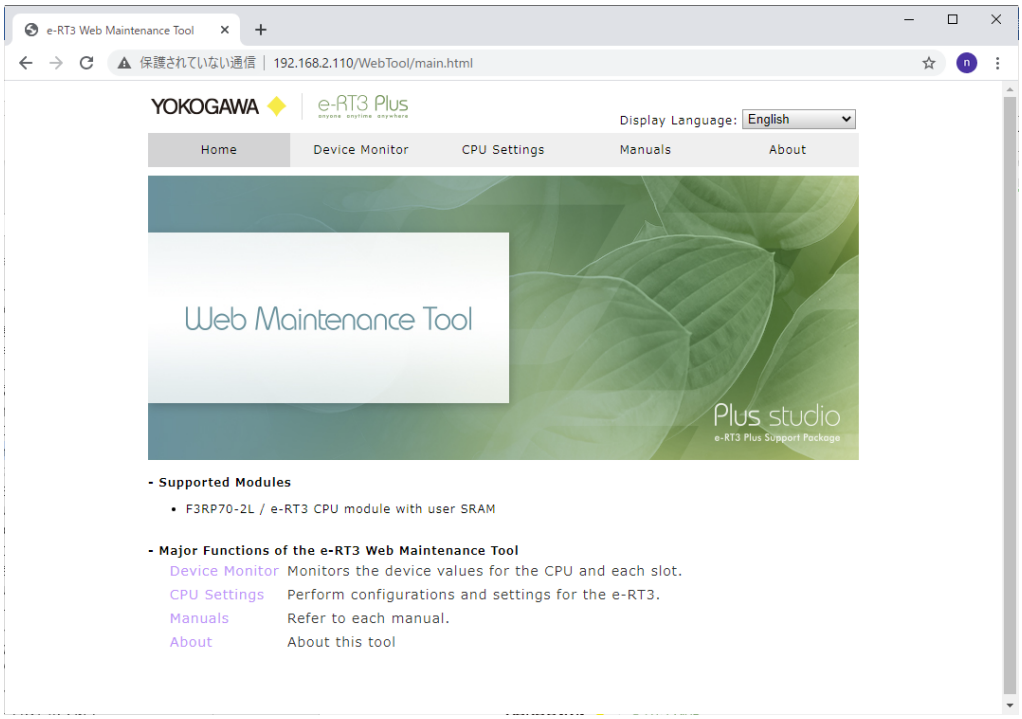


Figure A2.2.3 Main screen

■ About the navigation menu

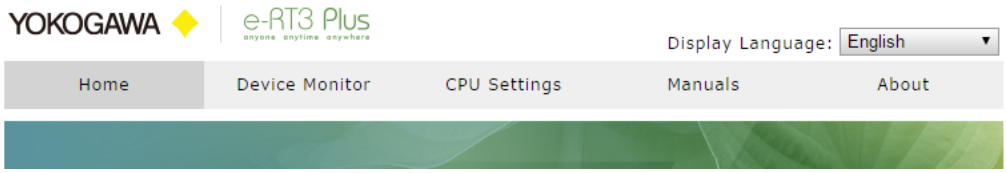


Figure A2.2.4 Navigation menu

Table A2.2.2 Navigation menu

Item	Description
Home	Displays the main screen.
Device monitor	Displays the device monitor screen. (Refer to "A2.3 Device monitor (Module selection screen)".)
CPU settings	Displays the CPU settings screen. (Refer to "A2.4 CPU settings".)
Manual	Browses the PDF data such as manuals. (Refer to "A2.5 Manual display".)
About	Displays information such as the version.

A2.2.4 Changing languages

On the main window, you can change the language the Web Maintenance Tool is displayed in (Japanese or English). The initial setting is English.

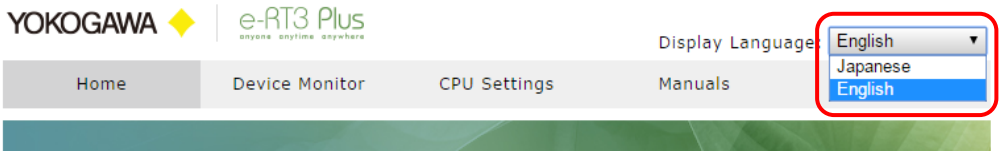


Figure A2.2.5 Display language

A2.3 Device monitor (Module selection screen)

Click [Device Monitor] on the Navigation menu to display the device monitor (module selection screen).

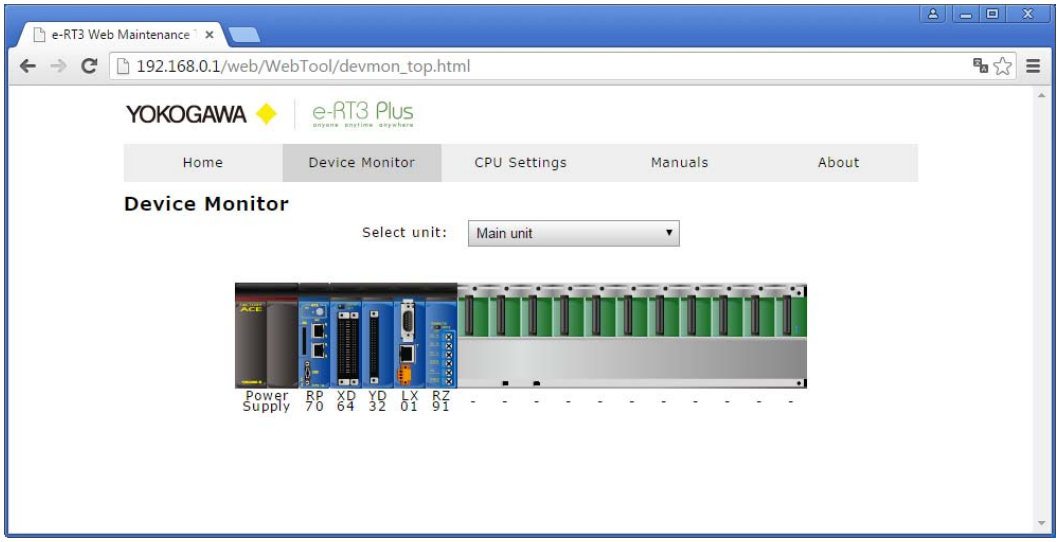


Figure A2.3.1 Device monitor (Module selection screen)

If browsing a sub-unit, select one from the [Select unit] pull-down menu.

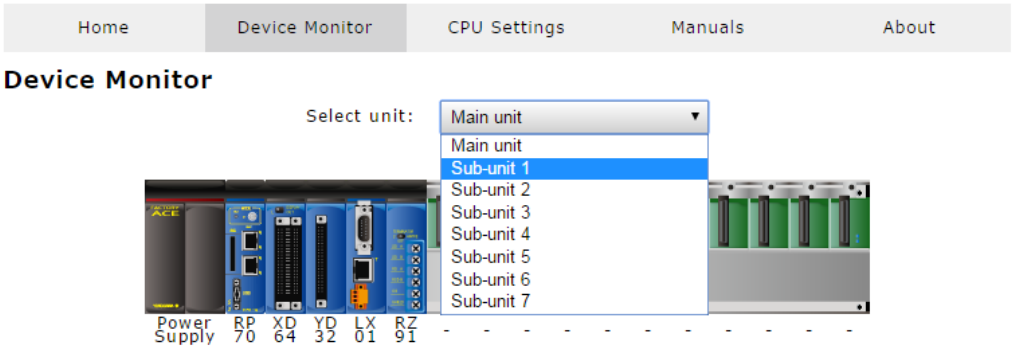


Figure A2.3.2 Device monitor (Sub-unit)

The module selection screen displays the configuration of modules equipped to the connected unit. Select a module to display the device monitor screen for that module.

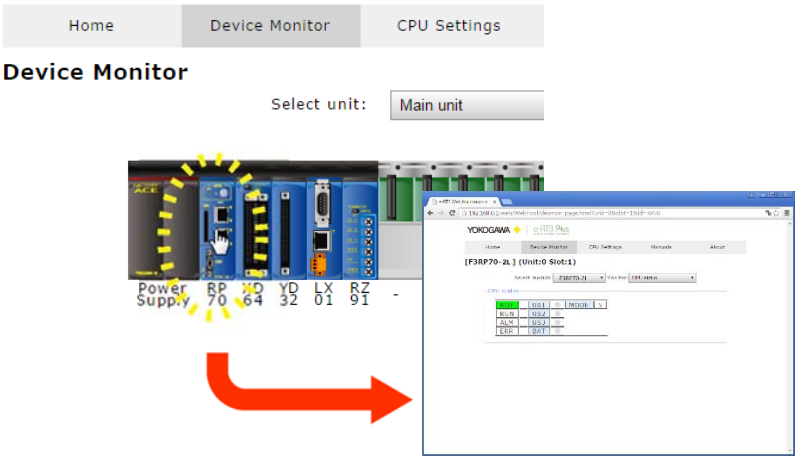


Figure A2.3.3 Selecting a module

A2.3.1 CPU module monitor screen

Select a CPU module from the module configuration to display that CPU module's status and internal device monitor screen. The CPU module monitor screen can show the following screens.

- CPU status screen
- CPU device monitor screen

The top page is the CPU status screen. To change the display, select from the [Monitor] pull-down menu.

Note

To display the module selection screen, click [Device Monitor] in the navigation menu, or click the back button in your browser's menu bar.

■ CPU status screen

This screen displays the CPU module's LED light status and mode switch status.

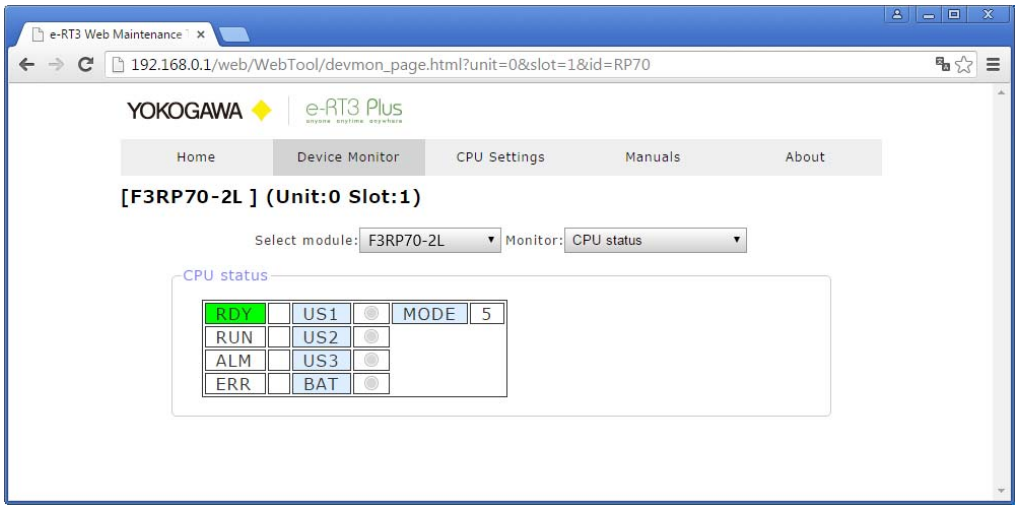


Figure A2.3.4 CPU status screen

■ CPU device (Relay device) monitor screen

On the relay device monitor screen, you can monitor the following.

- Internal relays (I)
- Shared relays (E)
- Extended shared relays (EE)
- Link relays (L)

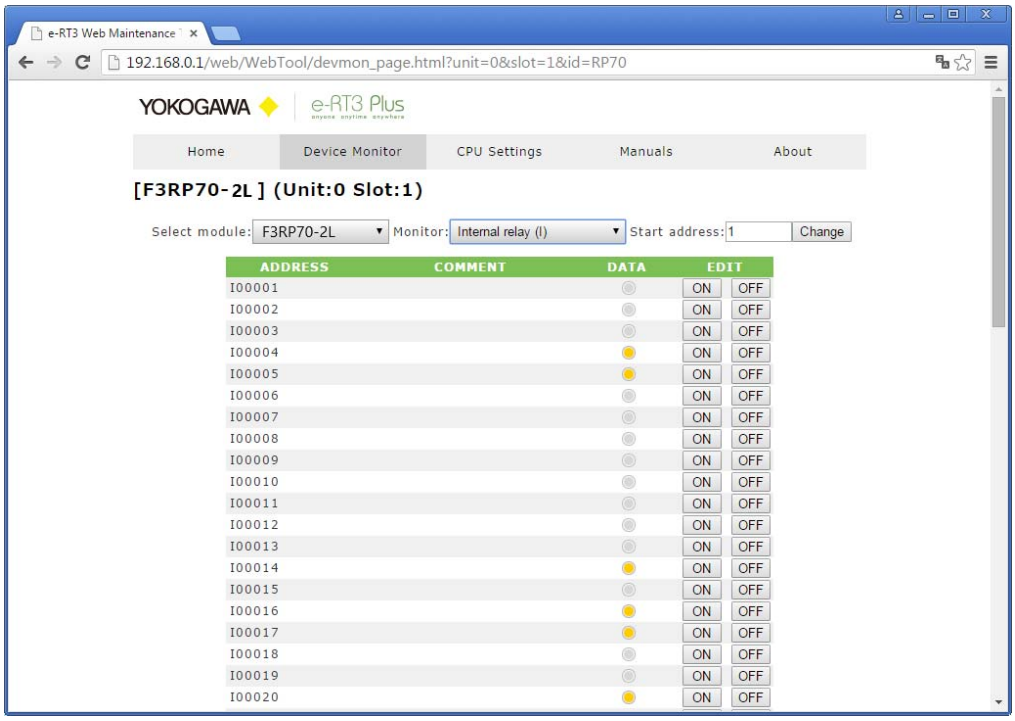


Figure A2.3.5 Relay device monitor screen

The relay device monitor screen can display 64 devices at once.
To change the displayed devices, input the address number to be displayed at the first line into the [Start address] input field, then click the [Change] button.

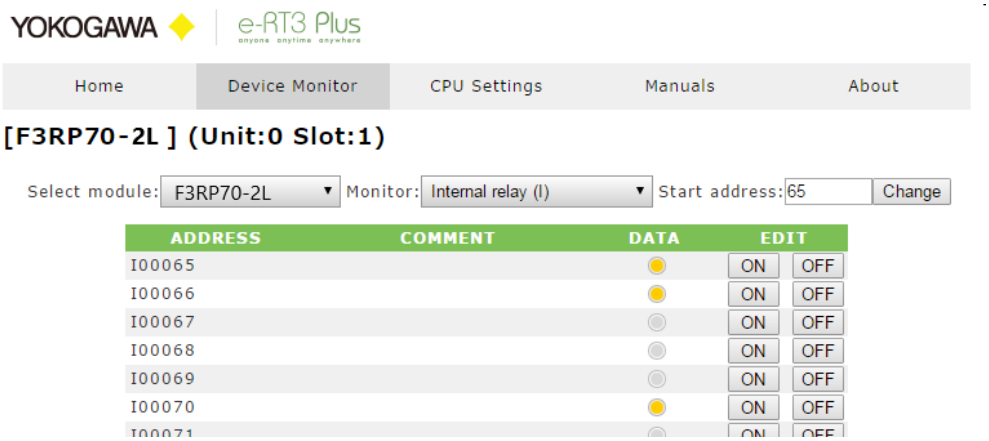


Figure A2.3.6 Changing addresses

The [COMMENT] column in the table displays the device comment defined by the user. Regardless of the selection for [Display Language], the comment will be displayed in the language the user used to define it.
The [DATA] column indicates if a device is on or off. You can also use the on/off buttons in the [EDIT] column to turn a device on or off.

Note

In order to turn a device on or off, you must login to the Web Maintenance Tool on the CPU settings screen.
For details about device comments, refer to "A2.3.3 Using and installing comment file".

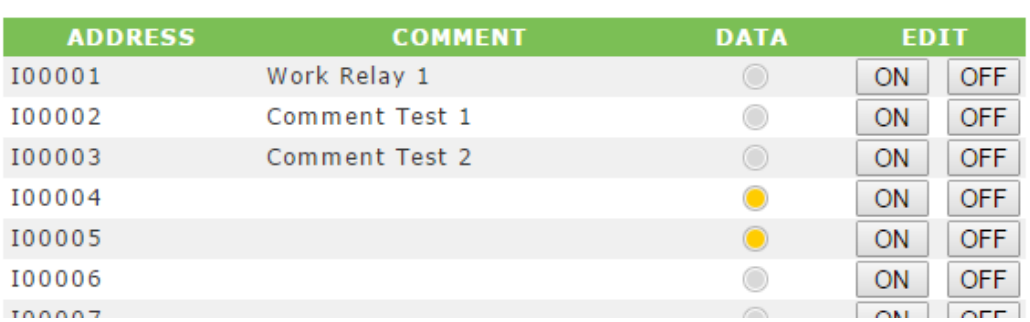


Figure A2.3.7 Displaying device comments and ON/OFF statuses

■ CPU device (Register device) monitor screen

On the register device monitor screen, you can monitor the following.

- Data registers (D)
- Shared registers (R)
- Extended shared registers (ER)
- Link registers (W)

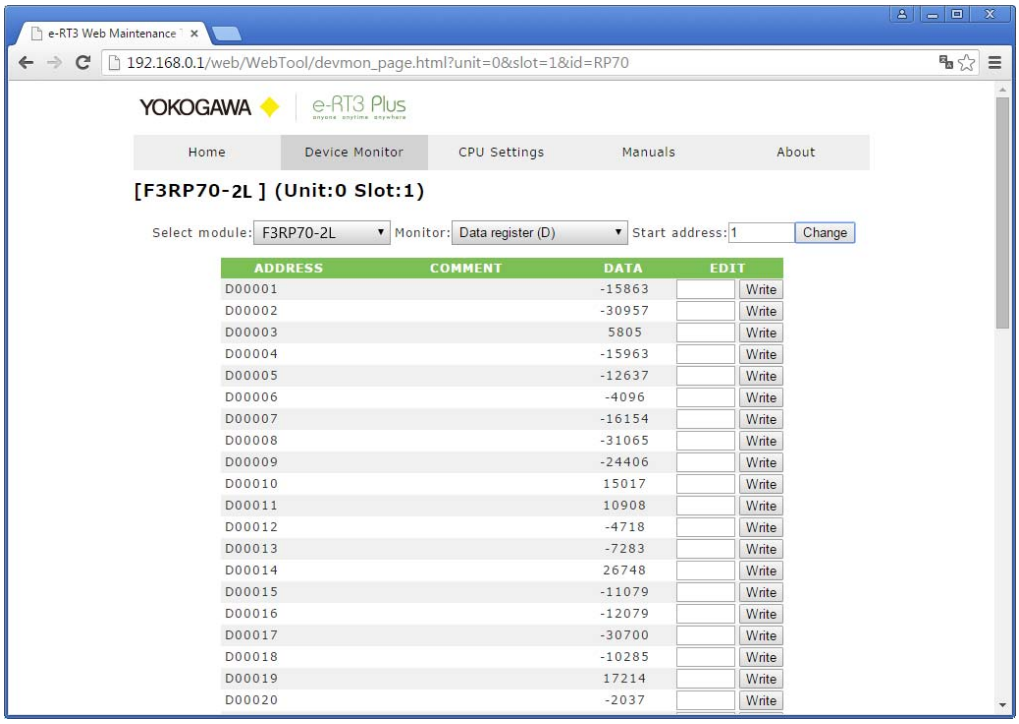


Figure A2.3.8 Register device monitor screen

The device monitor screen can display 64 devices at once.
To change the displayed devices, input the address number to be displayed at the first line into the [Start address] input field, then click the [Change] button.

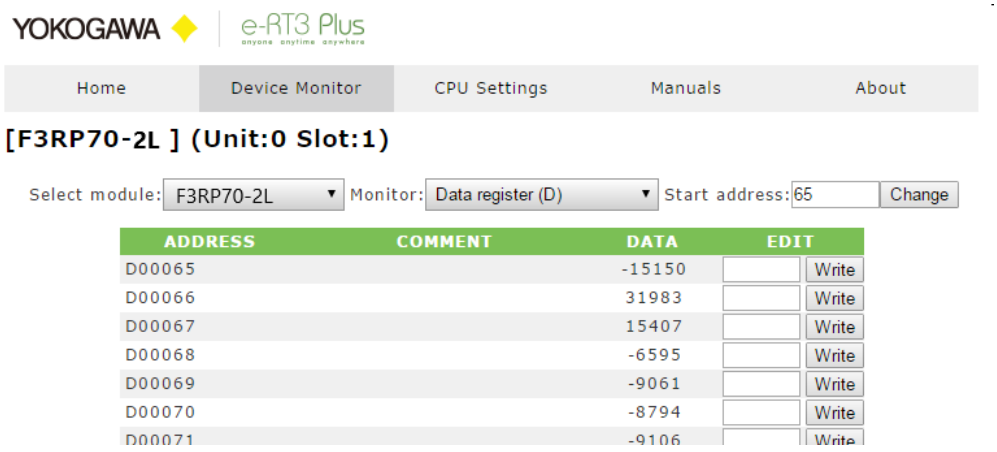


Figure A2.3.9 Changing addresses

The [COMMENT] column in the table displays the device comment defined by the user. Regardless of the selection for [Display Language], the comment will be displayed in the language the user used to define it.

The [DATA] column displays the current value. The word length value is displayed as a signed decimal.
Also, you can input any value into the [EDIT] column's input field and click [Write] to change the setting value.

Note

In order to change a setting value, you must login to the Web Maintenance Tool on the CPU settings screen.
For details about device comments, refer to "A2.3.3 Using and installing comment file".

ADDRESS	COMMENT	DATA	EDIT	
D00001		-10000	-10000	Write
D00002		-30957		Write
D00003		5805		Write
D00004		-15963		Write
D00005		-12637		Write
D00006		-4096		Write
D00007		-16154		Write

Figure A2.3.10 Changing the setting value

If you input a value outside the specified range, an error message appears.

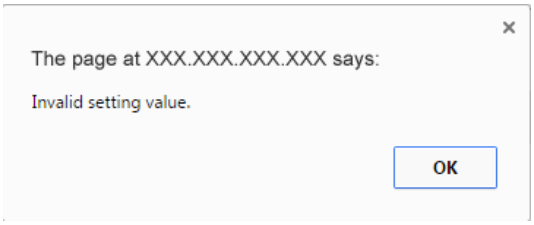


Figure A2.3.11 Error message

A2.3.2 I/O device monitor screen

Select an I/O module from the module configuration to display that I/O module's monitor screen. On the I/O module monitor screen, you can monitor the following depending on the module type.

- Input relays
- Output relays
- Internal registers

If there is a user comment file, the comments are quoted and displayed. For an advanced I/O module, if there is no user comment file, the system comments are quoted from the module definition file and displayed.
The top page is the device monitor screen for I/O relays. To change the display, select from the [Monitor] pull-down menu.

Note

To display the module selection screen, click [Device Monitor] in the navigation menu, or click the back button in your browser's menu bar.

■ I/O relay monitor

If [Input relay] or [Output relay] is selected for [Monitor], the screen displays the monitor for the I/O module's input or output relays.
The [COMMENT] column displays user comments about the module's position and model name.
If there is no user comment and the system has advanced I/O module definition information, the system's comments are displayed.

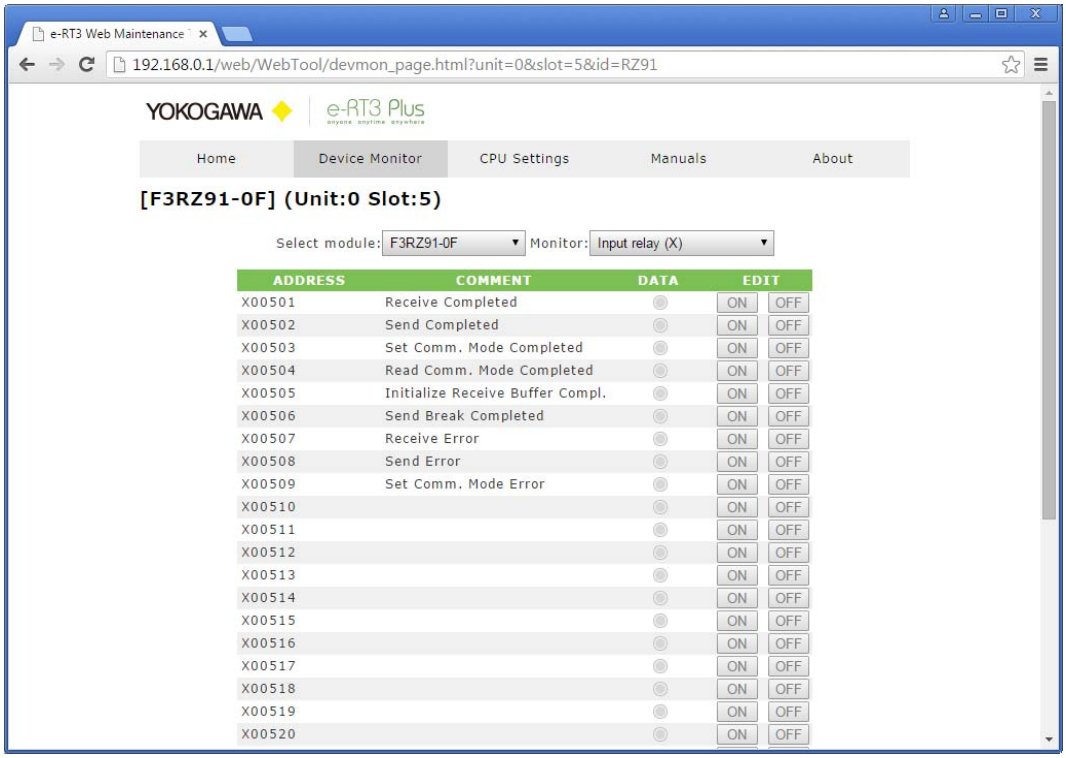


Figure A2.3.12 I/O device monitor screen

The [DATA] column indicates if a device is on or off. For output relays, you can also use the on/off buttons in the [EDIT] column to turn a device on or off.

Note

In order to turn a device on or off, you must login to the Web Maintenance Tool on the CPU settings screen.

ADDRESS	COMMENT	DATA	EDIT
Y00533	Read Received Data Completed		ON OFF
Y00534	Request to Send		ON OFF
Y00535	Req. to Set Comm. Mode		ON OFF
Y00536	Req. to Read Comm. Mode		ON OFF
Y00537	Req to Initialize Receive Buffer		ON OFF
Y00538	Req. to Send Break		ON OFF

Figure A2.3.13 Output relay monitor

■ Advanced register monitor

If [Internal register] is selected for [Monitor], the screen displays the monitor for the advanced I/O module's internal registers.
The [COMMENT] column displays user comments about the module's position and model name.
If there is no user comment and the system has advanced I/O module definition information, the system's comments are displayed.

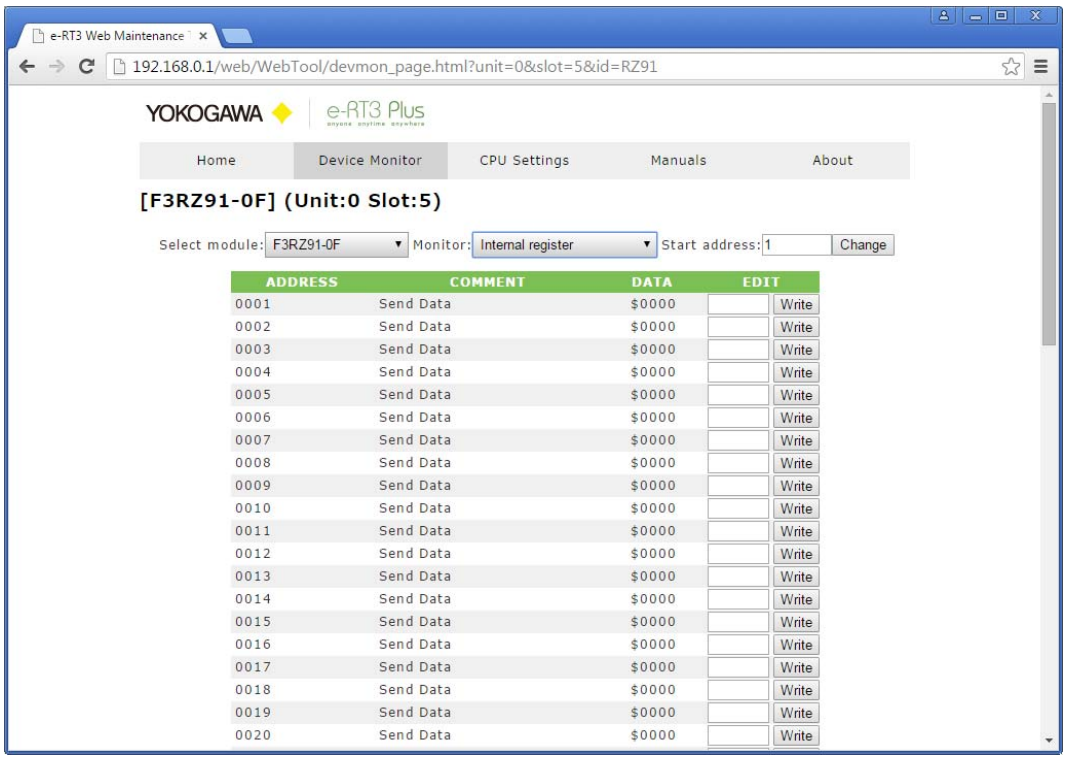


Figure A2.3.14 Advanced register monitor screen

The register device monitor screen can display 64 devices at once.
To change the displayed registers, input the address number to be displayed at the first line into the [Start address] input field, then click the [Change] button.

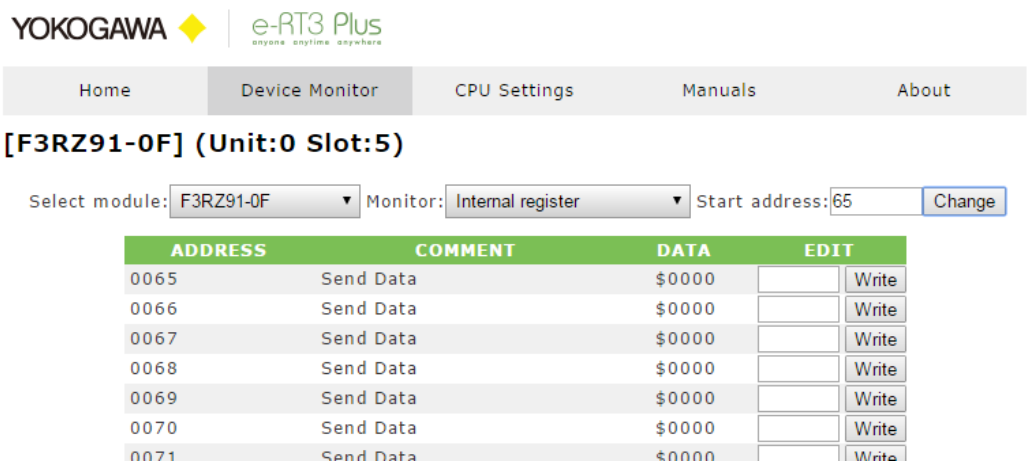


Figure A2.3.15 Changing addresses

The [DATA] column displays the current value. The word length value is displayed as a signed decimal.
Also, you can input any value into the [EDIT] column's input field and click [Write] to change the setting value.

Note

In order to change a setting value, you must login to the Web Maintenance Tool on the CPU settings screen.

Figure A2.3.16 Changing the setting value

If you input a value outside the specified range, an error message appears.

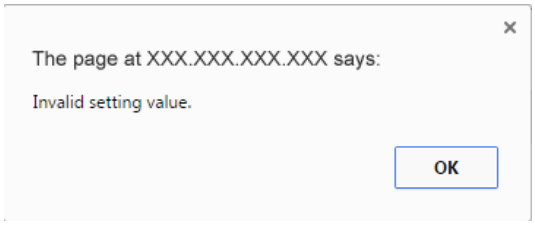


Figure A2.3.17 Error message

Note

If the user comment is not displayed, there is an error in the definition of the user comment file. Check it and make corrections, then re-execute.
If the comment information is displayed with corrupted characters, there is an error in the encoding specification of the saved user comment file. Save it in UTF-8 format, then re-execute.

A2.3.3 Using and installing comment file

With the Web Maintenance Tool, you can create user comments for the device monitors.

■ Creation procedure

Note

Create the comment file in the ini file format.

(1) Specify the module model number and slot number.

Separate the four characters in front of the hyphen ("-") in the module model number and the slot number where the module is installed with a comma (","), and enclose them in brackets ("[ ]").

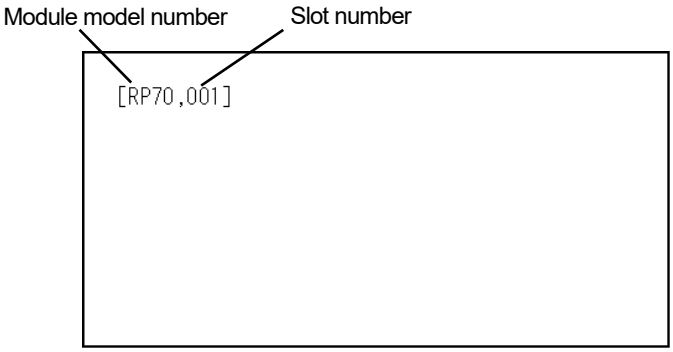


Figure A2.3.18 Example comment file

(2) Specify a register in the module.

Input the device type and address, then input "=".

The address is expressed using zero suppression (written without 0s, and I00001 results in an error).

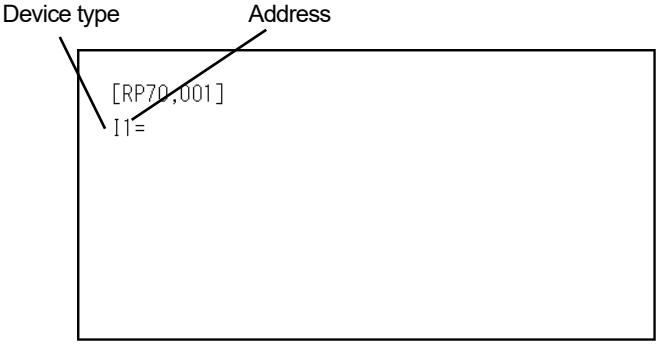


Figure A2.3.19 Example comment file

For a CPU module, the following devices can be defined.

Device type	Device name
I	Internal relay
D	Data register
E	Shared / Extended shared relay
R	Shared / Extended shared register
L	Link relay
W	Link register

For an I/O module, the following devices can be defined.

Device type	Device name
X	Input relay
Y	Output relay

Note

The device settings for the CPU module can be specified within the address ranges in the device settings. For details about the setting ranges, refer to "A2.4.4 Device setting screen".
For I/O modules, the actual address is stored internally depending on the installation slot.
For example, the comment for a module installed in Slot 004 shows Y1 = "Comment" in the ini file definition, but the actual monitor address is "Y00401".

(3) Create a comment.

Input the comment after "=". Input the comment with 32 characters and enclose it in quotation marks (" ").

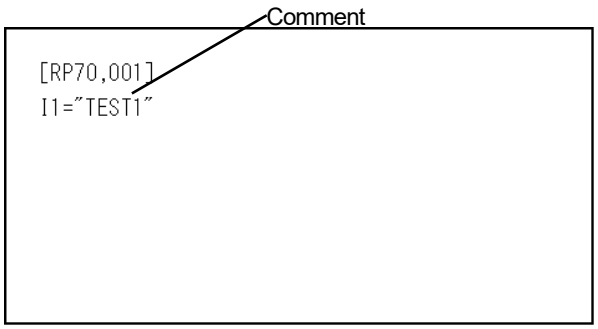


Figure A2.3.20 Example comment file

(4) When you finish creating all the comments, save the file in the UTF-8 format using "UserComment.rpc" as the file name.

```
[RP70,001]
I1="TEST1"
I3="TEST2"
D5="TRIAL1"
D6="TRIAL2"
     .
     .
     .
```

Figure A2.3.21 Example comment file

(5) Store the comment file in "/media/sd/WebTool/".

Make directory for storage

```
# mkdir -p /media/sd/WebTool
# chmod 0777 /media/sd/WebTool
```

You should store it in the above directory, through SFTP server. You can use any SFTP client such as Visual Studio Code described in chapter 5 or WinSCP described in chapter 6.

A2.4 CPU settings

A2.4.1 CPU settings (Top/Login) screen

This is the top page for CPU module settings.
This is also the screen where a user with CPU setting privileges can login to perform configuration settings.

Note

General users cannot change CPU settings.
To register a new user, refer to "A2.4.2 User management screen".

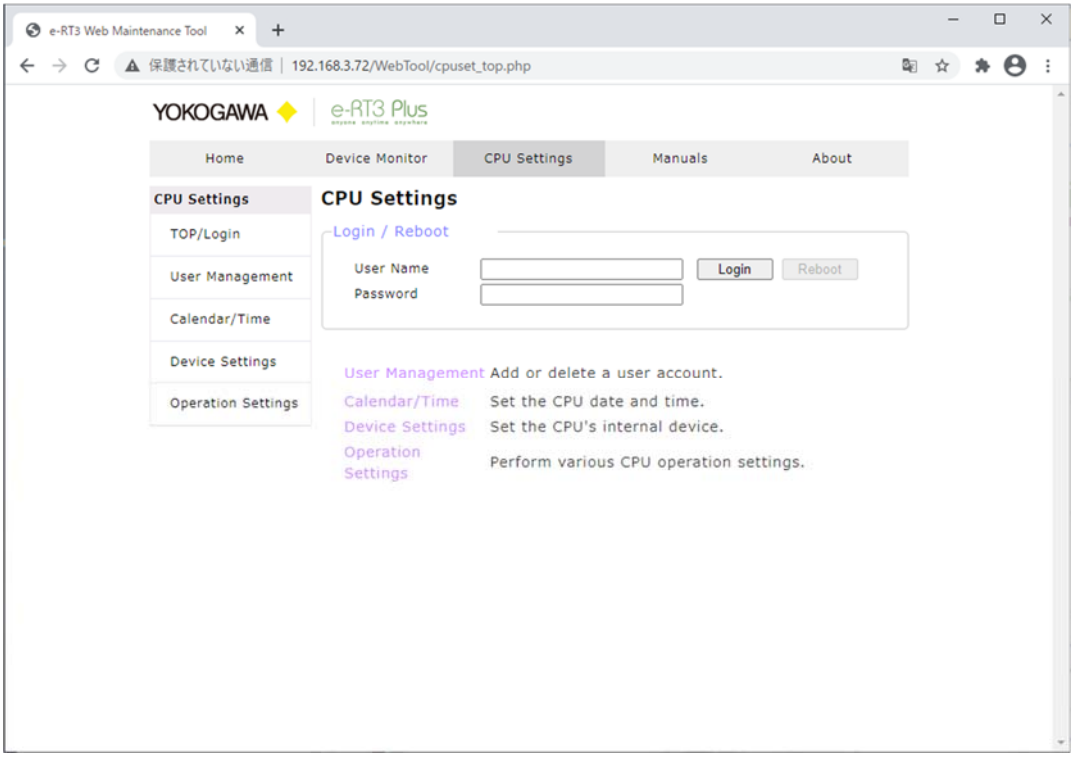


Figure A2.4.1 CPU settings (Top/Login) screen

Input your registered user name and password, then click the [Login] button.
You can perform configuration settings after logging in.

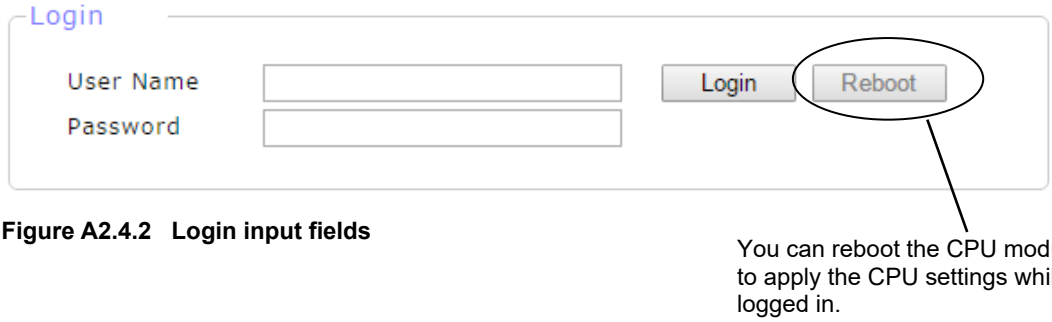


Figure A2.4.2 Login input fields

A2.4.2 User management screen

This screen is for adding and deleting user accounts which set the CPU modules.
"Administrator" is registered by default. (Default password : Administrator)

Note

In order to add or delete a user account, you must login to the Web Maintenance Tool on the CPU settings screen.

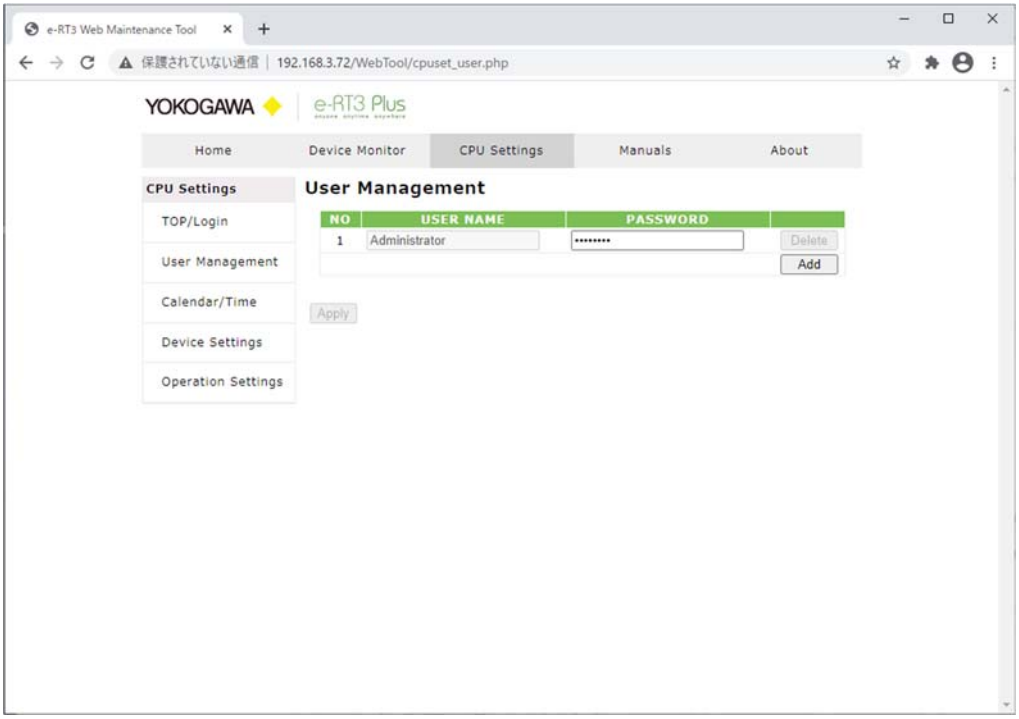


Figure A2.4.3 User management screen

To add a user account, click the [Add] button.
Input fields are displayed for the user name and password. Input the new user name and password.
To delete a user account, click the [Delete] button to the right of the field you want to delete.
The user account is deleted.
If writing the settings to the CPU module, click the [Apply] button.

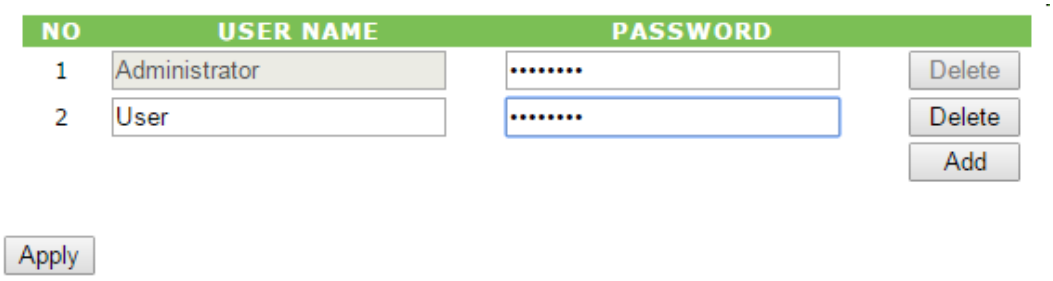


Figure 2.4.4 Adding or deleting user accounts

A2.4.3 Calendar / Time settings screen

This screen is for setting the date and time of the CPU module.

Note

In order to execute the calendar and time settings, you must login to the Web Maintenance Tool on the CPU settings screen.

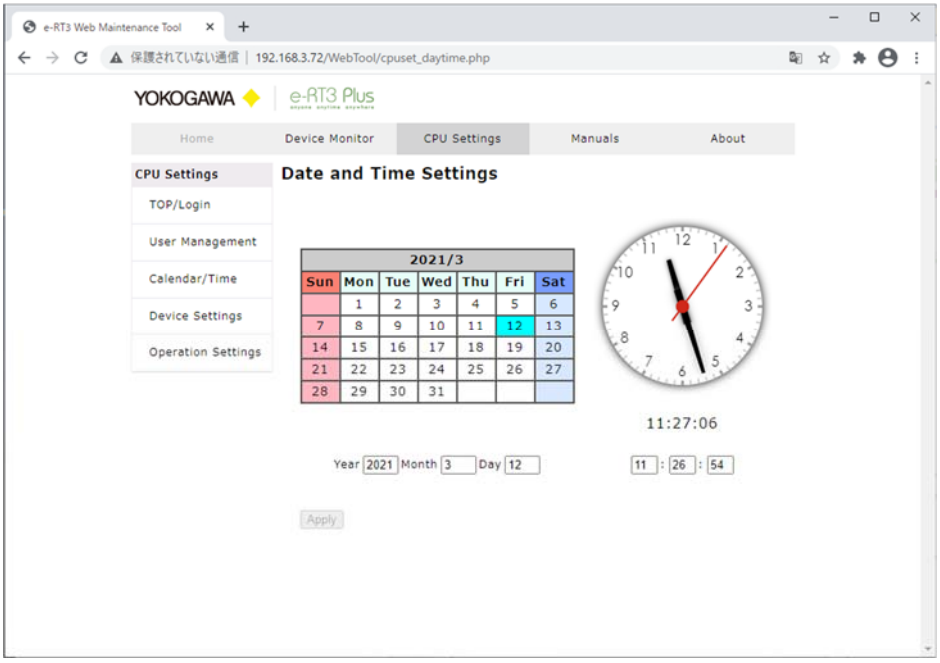


Figure A2.4.5 Calendar / Time settings screen

Input the date and time you want to set into the input fields displayed below the calendar and clock, then click the [Apply] button.

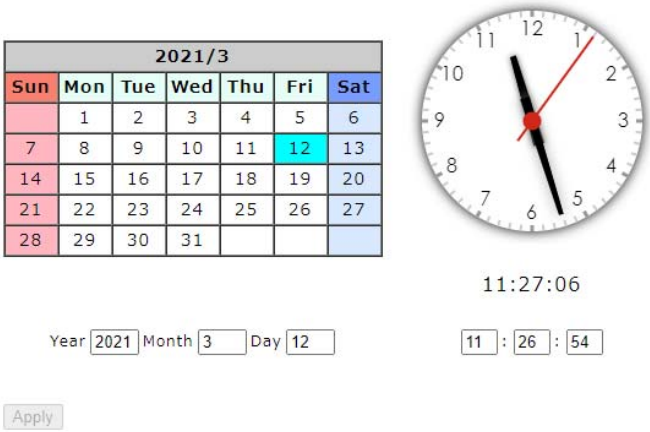


Figure A2.4.6 Setting the date and time

Note

When you change the time zone of Ubuntu system, reboot the system before setting time with this tool.

A2.4.4 Device settings screen

This screen is for setting the configuration of the internal devices, shared devices, and link devices used by the CPU.
If writing the settings to the CPU module, click the [Apply] button.

Note

In order to execute device settings, you must login to the Web Maintenance Tool on the CPU settings screen.
Settings are enabled after the CPU module is restarted.

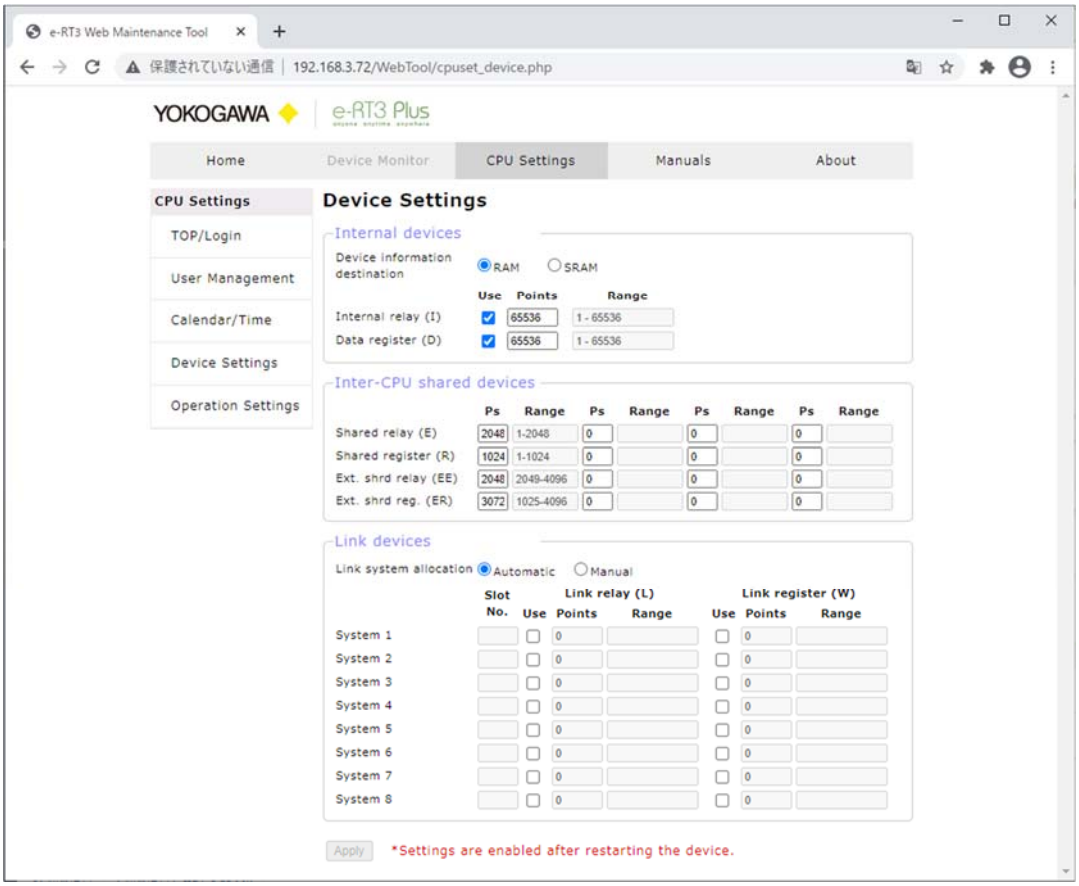


Figure A2.4.7 Device settings screen

■ Internal devices

This sets the internal relays and registers.

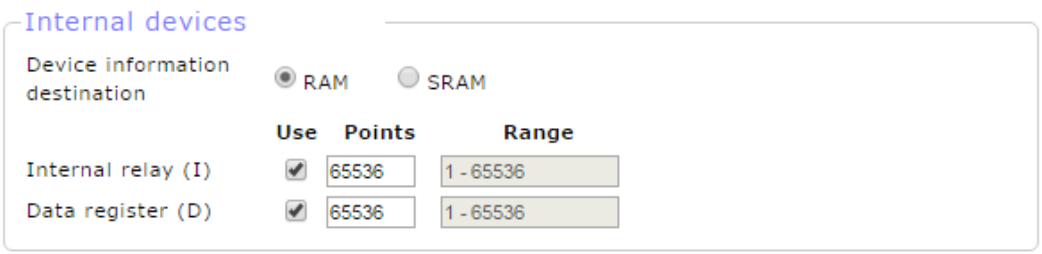


Figure A2.4.8 Internal devices

Table A2.4.1 Settings for internal devices

Item	Description	Range
Device information destination	Select RAM or SRAM.	RAM/SRAM
Internal relay (I) / Use	Sets whether to use the internal relay or not. **Note** If you unselect [Use], you cannot input [Points].	-
Internal relay (I) / Points	Sets the number of internal relay points used. **Note** Set the internal relay in units of 32 points. Example settings: 32, 64, 96, 128...	0 to 65536
Data register (D) / Use	Sets whether to use the data register or not. **Note** If you unselect [Use], you cannot input [Points].	-
Data register (D) / Points	Sets the number of data register points used. **Note** Set the data register in units of 2 points. Example settings: 2, 4, 6, 8...	0 to 65536

■ Inter-CPU shared devices

This sets the shared relays and shared registers.

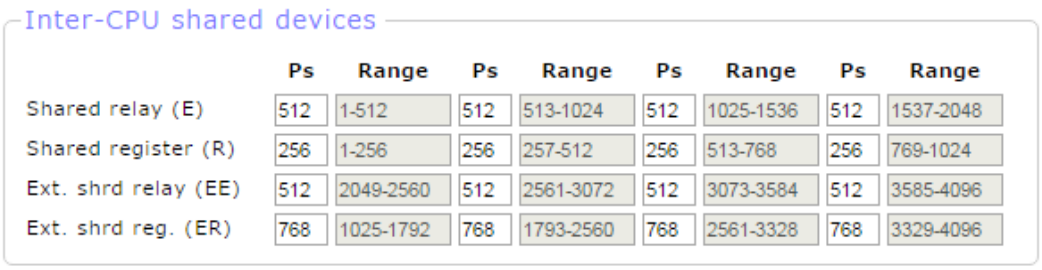


Figure A2.4.9 Inter-CPU shared devices

Table A2.4.2 Settings for inter-CPU shared devices

Item	Description	Range
Shared relay (E)	Sets the range of the CPU shared relay. **Note** Set the shared relay so that the total number of points for CPUs 1 to 4 comes to a range of 0 to 2048. Also, set the shared relay in units of 32 points. Example settings: 32, 64, 96, 128...	0 to 2048
Shared register (R)	Sets the range of the CPU shared register. **Note** Set the shared register so that the total number of points for CPUs 1 to 4 comes to a range of 0 to 1024. Also, set the shared register in units of 2 points. Example settings: 2, 4, 6, 8...	0 to 1024
Ext. shrd relay (EE)	Sets the range of the CPU extended shared relay. **Note** Set the extended shared relay so that the total number of points for CPUs 1 to 4 comes to a range of 0 to 2048. Also, set the extended shared relay in units of 32 points. Example settings: 32, 64, 96, 128...	0 to 2048
Ext. shrd reg. (ER)	Sets the range of the CPU extended shared register. **Note** Set the extended shared register so that the total number of points for CPUs 1 to 4 comes to a range of 0 to 3072. Also, set the extended shared register in units of 2 points. Example settings: 2, 4, 6, 8...	0 to 3072

A2.4.5 Operation settings screen

This screen is for setting the configuration of the CPU module's peripheral services (M command server, FL-net link refresh, higher-level link service).
If writing the settings to the CPU module, click the [Apply] button.

Note

In order to execute operation settings, you must login to the Web Maintenance Tool on the CPU settings screen.
Settings are enabled after the CPU module is restarted.

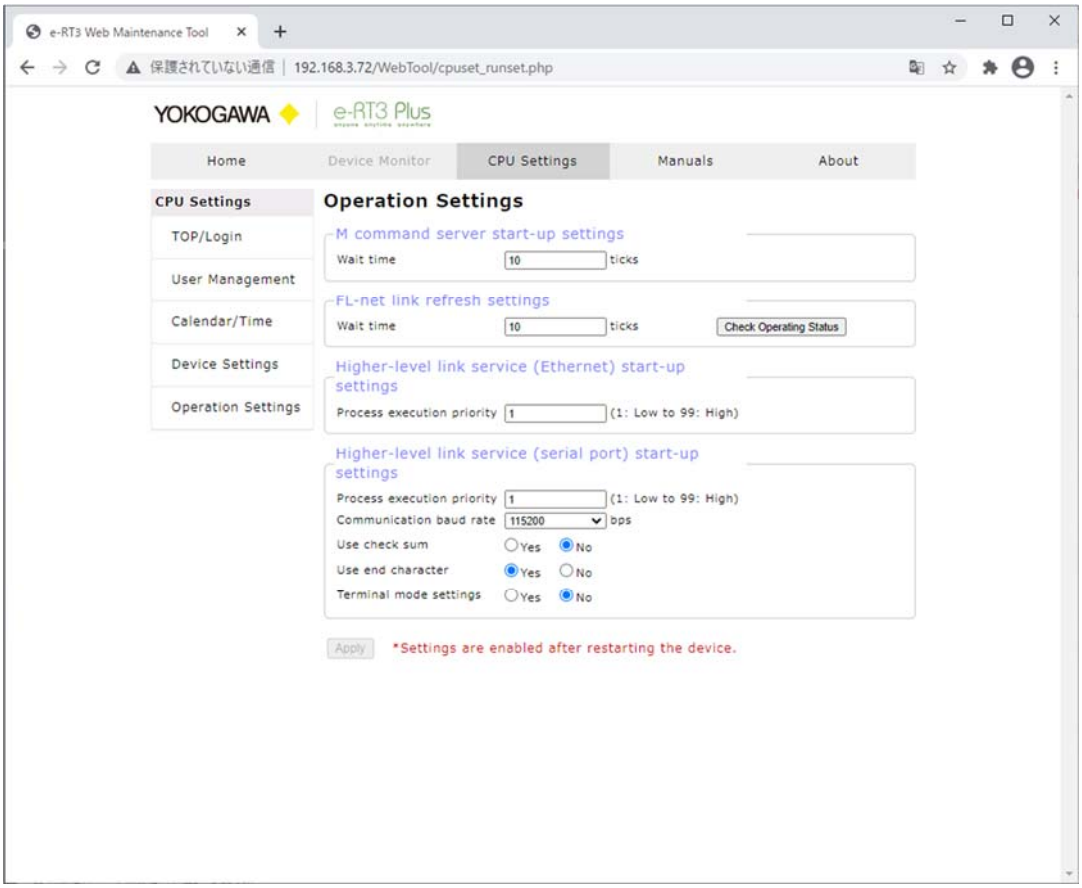


Figure A2.4.10 Operation settings screen

■ M command server start-up settings

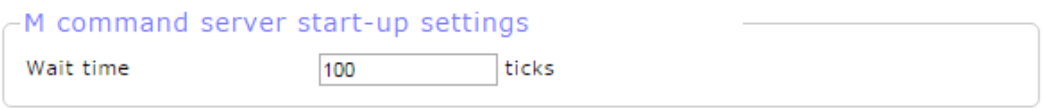


Figure A2.4.11 M command server start-up settings

Table A2.4.3 M command server start-up settings

Item	Description	Range
Wait time	Sets the start-up wait time for the M command server.	1 to 100

■ Higher-level link service (Ethernet) start-up settings

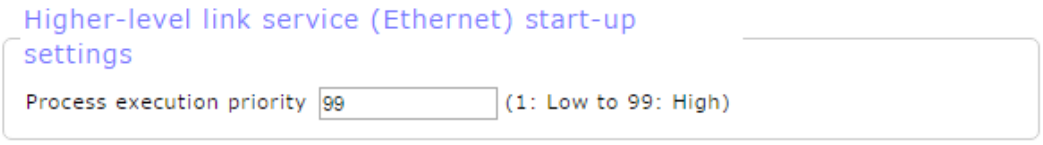


Figure A2.4.12 Higher-level link service (Ethernet) start-up settings

Table A2.4.4 Higher-level link service (Ethernet) start-up settings

Item	Description	Range
Process execution priority	Specifies the process execution priority for the higher-level link service (Ethernet).	1 to 99

■ Higher-level link service (serial port) start-up settings

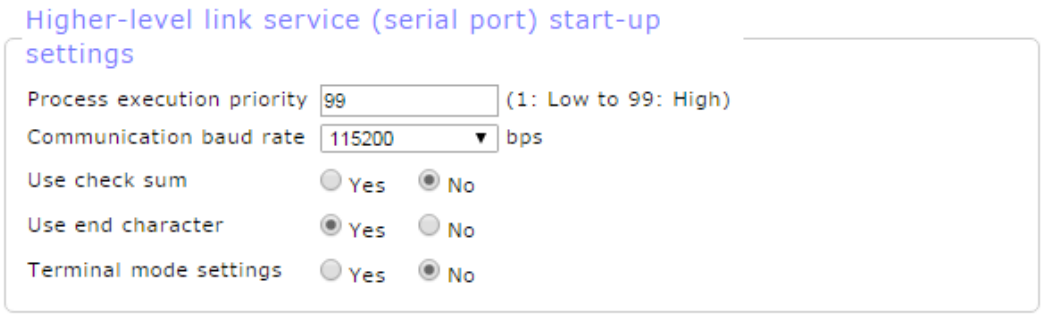


Figure A2.4.13 Higher-level link service (serial port) start-up settings

Table A2.4.5 Higher-level link service (serial port) start-up settings

Item	Description	Range
Process execution priority	Specifies the process execution priority for the higher-level link service (serial port).	1 to 99
Communication baud rate	Selects the communication baud rate.	9600/19200/ 38400/57600/ 115200
Use check sum	Specifies whether to use the check sum or not.	Yes / No
Use end character	Specifies whether to use the end character or not.	Yes / No
Terminal mode settings	Specifies whether to use the terminal mode or not.	Yes / No

A2.5 Manual display

On the manual screen, you can display PDF documents for manuals on the Web Maintenance Tool.

Note

If the execution environment does not have a function for viewing PDF data (browser plug-in or application for viewing PDFs), then you cannot display the PDF documents.

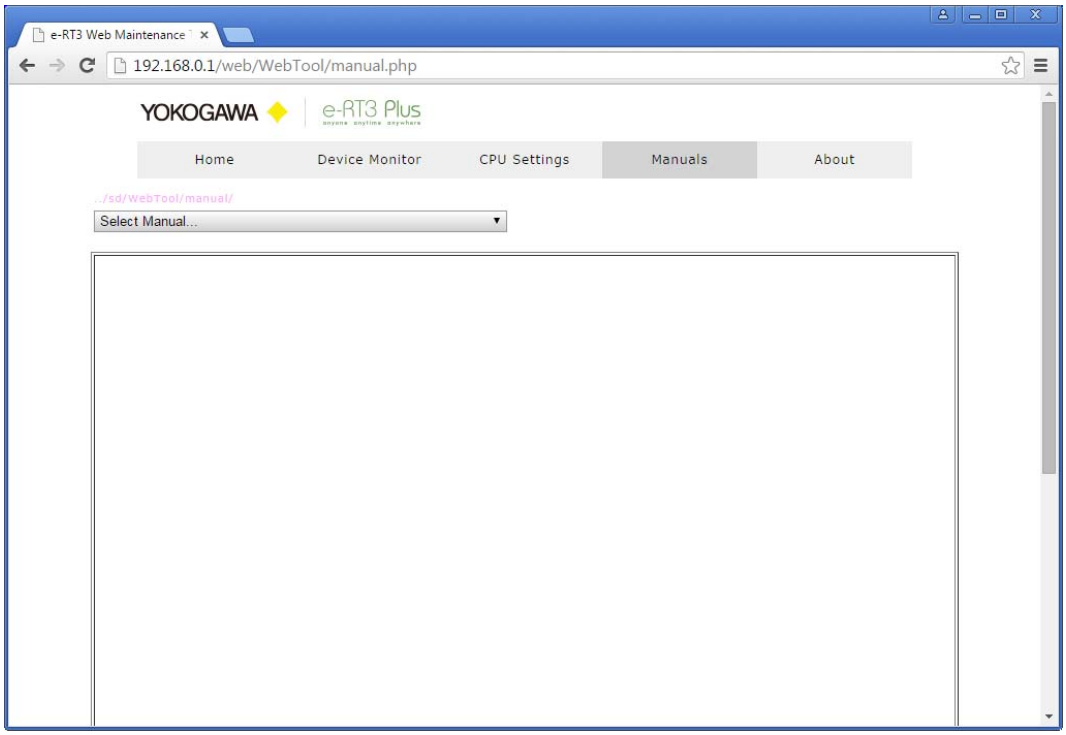


Figure A2.5.1 Manual screen

A2.5.1 Installing manual files

(1) Make directory for storage

```
# mkdir -p /media/sd/WebTool/manual
# chmod 0777 /media/sd/WebTool/manual
```

(2) Store the document's PDF data.

You should store them in the above directory, through SFTP server. You can use any SFTP client such as Visual Studio Code described in chapter 5 or WinSCP described in chapter 6.

Note

Use only single-byte alphanumeric characters for the file names of the PDF data stored in the directory.

A2.5.2 Displaying the manuals

■ Display method

Click [Manuals] in the Navigation menu to display the manuals screen.
From the pull-down menu, select the PDF document you want to display.

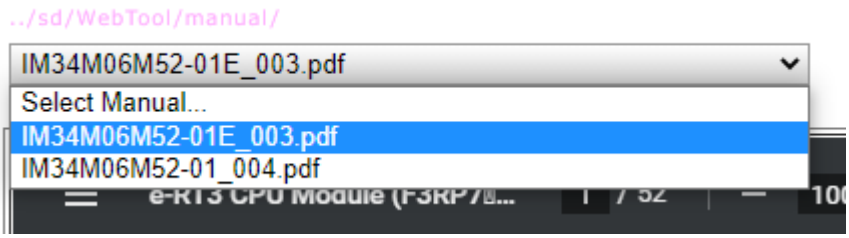


Figure A2.5.2 Pull-down menu for selecting a document

Note

If a PDF document's file name is not displayed in the pull-down menu, there is either an error in the specification of the directory on the SD card (/media/sd/WebTool/manual), or the file itself is invalid.
Check the file and the storage location.
If the PDF document is large, it may take some time for until it is displayed.

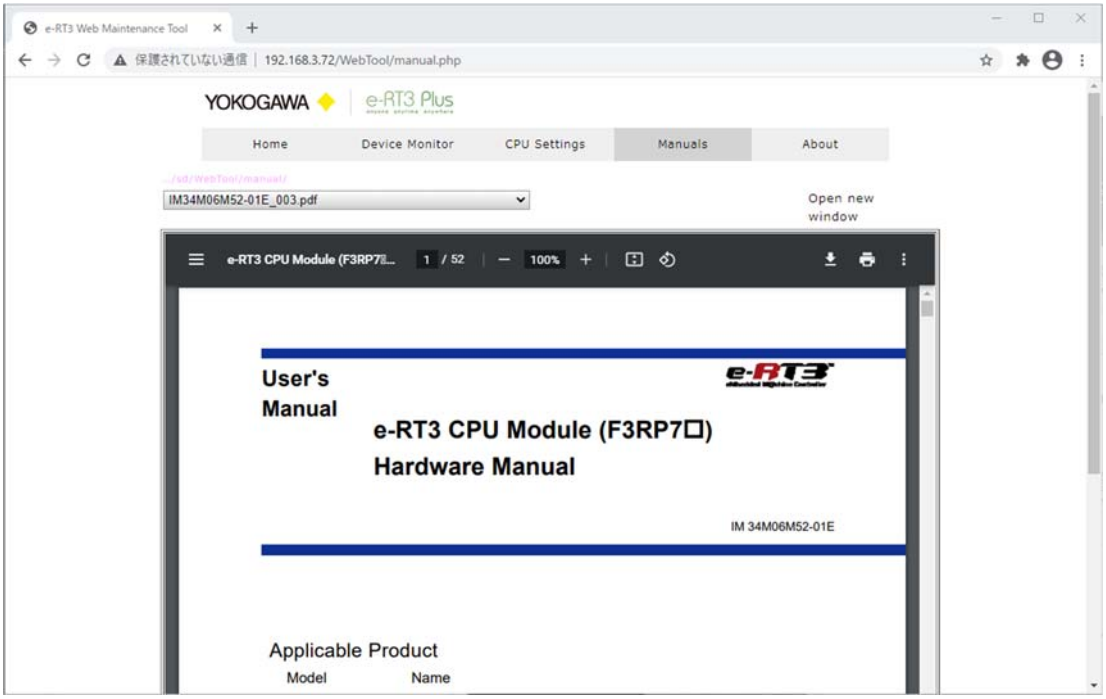


Figure A2.5.3 Displayed PDF document

■ Useful functions

Open new window function

Click the [Open new window] button on the top-right of the PDF document display to display the PDF document in a separate window.
This function enables you to display PDF documents even if the browser in your environment does not support PDF display plug-ins (such as a Macintosh or smartphone).

Note

If you cannot display a PDF document, check if a PDF viewer is installed.

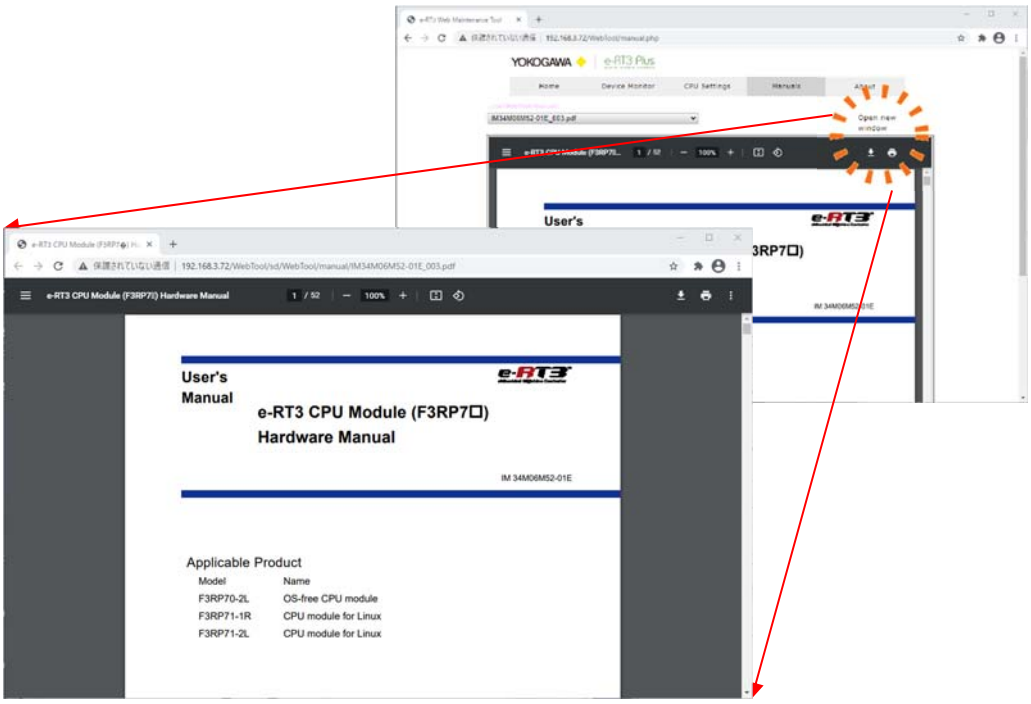


Figure A2.5.4 Open new window function

Revision Information

Title : Ubuntu Image for F3RP70 User's Guide
Document No. : TI 34M06T02-02E

Mar. 2021 / 2nd Edition
Add chapter 7
Add appendix 1 and 2
Correction of mistake

Apr. 2020 / 1st Edition
New publication

■ For Questions and More Information
If you have any questions, you can send an E-mail to the following address.
E-mail: plc_message@cs.jp.yokogawa.com
■ Written by
Yokogawa Electric Corporation
■ Published by
Yokogawa Electric Corporation
2-9-32 Nakacho, Musashino-shi, Tokyo, 180-8750, JAPAN