NEC

User's Manual

RX850

Real-Time Operating System

Installation

Target device
V850 family™
Target real-time OS
RX850 Ver. 3.13 or later

Document No. U13410EJ2V1UM00 (2nd edition)
Date Published October 2001 J CP(K)

[MEMO]

SUMMARY OF CONTENTS

V800 Series, V850 family, V851, V852, V853, V854, V850/SA1, V850/SB1, V850/SB2, V850/SV1, V850E/MS1, V850E/MA1, and V850E/IA1 are trademarks of NEC Corporation.
Green Hills Software and MULTI are trademarks of Green Hills Software, Inc.
HP9000 series 700 is a trademark of Hewlett-Packard Company.
UNIX is a trademark of X/Open Company, Ltd. licensed in the USA and other countries.
PC/AT is a trademark of IBM Corporation.
Solaris and SPARCstation are trademarks of SPARC International, Inc.
TRON is an abbreviation for The Realtime Operating system Nucleus.
ITRON is an abbreviation for Industrial TRON.
MS-DOS, Windows, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

The export of this product from Japan is prohibited without governmental license. To export or re-export this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives and distributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and other legal issues may also vary from country to country.

NEC Electronics Inc. (U.S.)
Santa Clara, California
Tel: 408-588-6000
800-366-9782
Fax: 408-588-6130
800-729-9288

NEC Electronics (Germany) GmbH
Duesseldorf, Germany
Tel: 0211-65 03 02
Fax: 0211-65 03 490

NEC Electronics (UK) Ltd.
Milton Keynes, UK
Tel: 01908-691-133
Fax: 01908-670-290

NEC Electronics Italiana s.r.l.
Milano, Italy
Tel: 02-66 75 41
Fax: 02-66 75 42 99

NEC Electronics (Germany) GmbH
Benelux Office
Eindhoven, The Netherlands
Tel: 040-2445845
Fax: 040-2444580

NEC Electronics (France) S.A.
Velizy-Villacoublay, France
Tel: 01-3067-5800
Fax: 01-3067-5899

NEC Electronics (France) S.A.
Madrid Office
Madrid, Spain
Tel: 091-504-2787
Fax: 091-504-2860

NEC Electronics (Germany) GmbH
Scandinavia Office
Taeby, Sweden
Tel: 08-63 80 820
Fax: 08-63 80 388

NEC Electronics Hong Kong Ltd.
Hong Kong
Tel: 2886-9318
Fax: 2886-9022/9044

NEC Electronics Hong Kong Ltd.
Seoul Branch
Seoul, Korea
Tel: 02-528-0303
Fax: 02-528-4411

NEC Electronics Singapore Pte. Ltd.
Novena Square, Singapore
Tel: 253-8311
Fax: 250-3583

NEC Electronics Taiwan Ltd.
Taipei, Taiwan
Tel: 02-2719-2377
Fax: 02-2719-5951

NEC do Brasil S.A.
Electron Devices Division
Guarulhos-SP, Brasil
Tel: 11-6462-6810
Fax: 11-6462-6829

J01.2

MAJOR REVISIONS IN THIS EDITION

Pages	Description
p.19	Modification of description in **Section 1.3**
p.20	Modification of description in **Section 1.4**
p.23	Modification of description in **Chapter 2**
p.37	Modification of description in **Chapter 3**
p.51	Addition of **Chapter 4**
p.55	Addition of **Chapter 5**
p.83	Addition of **Chapter 6**
p.99	Addition of **Appendix B**
p.45 in the previous edition	Deletion of **Chapter 4**
p.51 in the previous edition	Deletion of **Chapter 5**
p.79 in the previous edition	Deletion of **Chapter 6**
p.91 in the previous edition	Deletion of **Chapter 7**

The mark ★ shows major revised points.

[MEMO]

PREFACE

Users This manual is intended for those users who design and develop application systems of the V850 family.

Purpose This manual explains the functions of the RX850.

Organization This manual includes the following:

- Overview
- Installation
- System construction
- Memory and estimating its capacity
- Configuration file
- Operating configurater (CF850)

How to read this manual It is assumed that the readers of this manual have general knowledge on electric engineering, logic circuits, microcontrollers, the C language, and assembler.
In this manual, the "V851™", "V852™", "V853™", and "V854™" are referred to as the "V850 family".
Unless otherwise specified, the directory names used throughout this manual are Windows™-based. If you use a UNIX™-based OS, read "\" in a directory name as "/".

Notation

Note	:	Explanation of item indicated in the text		
Caution	:	Information to which the user should afford special attention		
Remark	:	Supplementary information		
Numeric value	:	Binary	:	XXXX or XXXXB
		Decimal	:	XXXX
		Hexadecimal	:	0xXXXX

Units for representing powers of 2 (address space or memory space):

K (kilo)	:	2^{10} = 1,024
M (mega)	:	2^{20} = $1,024^2$

Related documents

When using this manual, also refer to the following documents.

Some related documents may be preliminary versions. Note, however, that whether a related document is preliminary is not indicated in this manual.

Documents related to development tools (User's manual)

Document name		Document No.
IE-703002-MC (In-circuit emulator for V851, V852, V853, V854, V850/SA1™, V850/SB1™, V850/SB2™, V850/SV1™)		U11595E
IE-703003-MC-EM1 (Peripheral I/O board for V853)		U11596E
IE-703008-MC-EM1 (Peripheral I/O board for V854)		U12420E
IE-703017-MC-EM1 (Peripheral I/O board for V850/SA1)		U12898E
IE-703037-MC-EM1 (Peripheral I/O board for V850/SB1, V850/SB2)		U14151E
IE-703040-MC-EM1 (Peripheral I/O board for V850/SV1)		U14337E
IE-703102-MC (In-circuit emulator for V850E/MS1™)		U13875E
IE-703102-MC-EM1, IE-703102-MC-EM1-A (Peripheral I/O board for V850E/MS1)		U13876E
IE-V850E-MC (In-circuit emulator for V850E/IA1™), IE-V850E-MC-A (In-circuit emulator for V850E1 (NB85E core), V850E/MA1™)		U14487E
IE-V850E-MC-EM1-A (Peripheral I/O board for V850E1 (NB85E core))		To be prepared
IE-V850E-MC-EM1-B, IE-V850E-MC-MM2 (Peripheral I/O board for V850E1 (NB85E core))		U14482E
IE-703107-MC-EM1 (Peripheral I/O board for V850E1/MA1)		U14481E
IE-703116-MC-EM1 (Peripheral I/O board for V850E1/IA1)		To be prepared
V800 Series™ Development Tool (for 32-bit) Application Note Tutorial Guide Windows-based		U14218E
CA850 (C compiler package)	Operation	U14568E
	C	U14566E
	Project manager	U14569E
	Assembly language	U14567E
ID850 (Ver.2.20) (Integrated debugger)	Operation Windows-based	U14580E
SM850 (Ver.2.20) (System simulator)	Operation Windows-based	U14782E
RX850 (Real-time OS)	Basics	U13430E
	Installation	This manual
	Technical	U13431E
RX850 Pro (Real-time OS)	Basics	U13773E
	Installation	U13774E
	Technical	U13772E
RD850 (Task debugger)		U13737E
RD850 Pro (Task debugger)		U13916E
AZ850 (System performance analyzer)		U14410E
PG-FP3 (Flash memory programmer)		U13502E

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

[MEMO]

CHAPTER 1 OVERVIEW

1.1 OUTLINE

Rapid advances in semiconductor technologies have led to the explosive spread of microprocessors such that they are now to be found in more fields than many would have imagined only a few years ago. In line with this spread, the number of processing programs that must be created for microprocessors is also increasing. This rule of growth makes it difficult to create processing programs specific to given hardware.

For this reason, there is a need for operating systems (OSs) that can fully exploit the capabilities of the latest generation of ever-newer high-performance, multi-function microprocessors.

Conversely, control OSs are incorporated into control units. That is, these OSs are found in those environments where standard OSs cannot easily be applied because the hardware configuration varies from system to system and because efficient operation matching the application is required.

Against this market background, NEC has developed and released the RX850 to exploit the performance and functions of its high-end microprocessors, the V850 family, and to support the systematic organization of software in the future.

The RX850 is a built-in real-time, multitasking control OS that provides a highly efficient real-time, multitasking environment to increases the application range of processor control units.

The RX850 is a high-speed, compact OS capable of being stored in and run from the ROM of a target system.

1.2 FEATURES

The RX850 has the following features:

(1) Conformity with μITRON 3.0 specification

The RX850 is designed to conform with the μITRON 3.0 specification, that defines a typical built-in control OS architecture. The RX850 implements μITRON 3.0 functions of up to level S.
The μITRON 3.0 specification applies to a built-in, real-time control OS.

(2) High generality

The RX850 supports all the system calls specified by the μITRON 3.0 specification to offer superior application system generality.
The RX850 can be used to create a real-time, multitasking OS that is compact and optimum for the user's needs because the functions (system calls) to be used by the application system can be selected.

(3) Realization of real-time processing and multitasking

The RX850 supports the following functions to realize complete real-time processing and multitasking:

- Task management function
- Task-associated synchronization function
- Synchronous communication function
- Interrupt management function
- Memory pool management function
- Time management function
- System management function
- Scheduling function

(4) Scheduling lock function

The RX850 supports functions for disabling and resuming dispatching (task scheduling) by a user processing program.

(5) Compact design

The RX850 is a real-time, multitasking OS that has been designed on the assumption that it will be incorporated into the target system; it has been made as compact as possible to enable it to be loaded into a system's ROM.

(6) Utilization of original instructions

The high-speed execution speed of the V850 Family, combined with the original instructions, enables high-speed processing.

(7) Utility support

The RX850 supports the following utility to aid in system construction:

- CF850 (configurater)

★ ## 1.3 EXECUTION ENVIRONMENT

The RX850 has been developed as an OS for embedded control and runs on a target system equipped with the following hardware.

(1) Target CPU

- V851
- V852
- V853
- V854
- V850/SA1
- V850/SBx
- V850/SV1
- V850E/MS1
- V850E/MA1
- NB85E core
- V850E/IA1

(2) Peripheral controller

The RX850 eliminates the hardware-dependent portions from the nucleus and supplies them as sample source files, in order to support a range of execution environments. If these sample source files are rewritten for the respective target systems, a specific peripheral controller is not required.

★ 1.4 DEVELOPMENT ENVIRONMENT

This section explains the hardware and software environments required to develop application systems.

1.4.1 Hardware Environment

(1) Host machine

- PC-9800 series
- PC/AT™-compatible machine
- SPARCstation™
- HP9000 series 700™

(2) In-circuit emulators

- IE-703002-MC (V851, V852, V853, V854, V850/SA1, V850/SBx, V850/SV1)
- IE-703102-MC (V850E/MS1)
- IE-V850E-MC-A (V850E/MA1, NB85E core)
- IE-V850E-MC (V850E/IA1)

(3) I/O board for in-circuit emulator

- IE-703003-MC-EM1 (V853)
- IE-703008-MC-EM1 (V854)
- IE-703017-MC-EM1 (V850/SA1)
- IE-703037-MC-EM1 (V850/SBx)
- IE-703040-MC-EM1 (V850/SV1)
- IE-703102-MC-EM1 (V850E/MS1 5 V)
- IE-703102-MC-EM1-A (V850E/MS1 3.3 V)
- IE-703107-MC-EM1 (V850E/MA1)
- IE-703116-MC-EM1 (V850E/IA1)
- IE-V850E-MC-EM1-A (NB85E core 5 V)
- IE-V850E-MC-EM1-B (NB85E core 3.3 V)

Caution These I/O boards must be used in combination with the in-circuit emulator.

(4) PC interface boards

- IE-70000-98-IF-C (for PC-9800 series C bus)
- IE-70000-PC-IF-C (for PC/AT-compatible machines ISA bus)
- IE-70000-CD-IF-A (for PCMCIA socket)
- IE-70000-PCI-IF (for PCI bus)

1.4.2 Software Environment

(1) OS (( ): host machine)

- Windows 95/Windows 98/Windows NT™ 4.0 (PC-9800 series, PC/AT-compatible machines)
- Solaris™ Rel. 2.5.x (SPARCstation)
- UNIX HP-UX Rel. 10.20 (HP9000)

(2) Cross tools

- CA850 (NEC Corporation)
- CCV850 (Green Hills Software Inc.)

(3) Debuggers

- ID850 (NEC Corporation)
- SM850 (NEC Corporation)
- MULTI™ (Green Hills Software Inc.)
- PARTNER™ (Kyoto Microcomputer)

(4) Task debugger

- RD850 (NEC Corporation)

(5) System performance analyzer

- AZ850 (NEC Corporation)

[MEMO]

★

CHAPTER 2 INSTALLATION

This chapter explains how to install or uninstall the RX850.

2.1 INSTALLING

2.1.1 Installing Windows Version

This section explains how to install the Windows version of RX850. To re-install the original RX850, installation must first be uninstalled.

The Windows version of RX850 is supplied on a single CD-R, regardless of whether it is an object release version or source release version. The package of the RX850 includes the RD850 (task debugger). This program can be also installed at the same time.

In the example installation described below, the following setting is assumed:

- Install directory: c: \nectools32
- CD drive: Q drive
- Directory to which Windows is installed: b: \Windows

Install RX850 by following the procedure below:

<1> Start Windows.

<2> Insert the CD-R into the CD drive (Q drive). The set-up program will start automatically. If the set-up program does not start, start Explorer, and then double-click “Setup.exe” in the rx850\DISK1 folder on the Q drive.

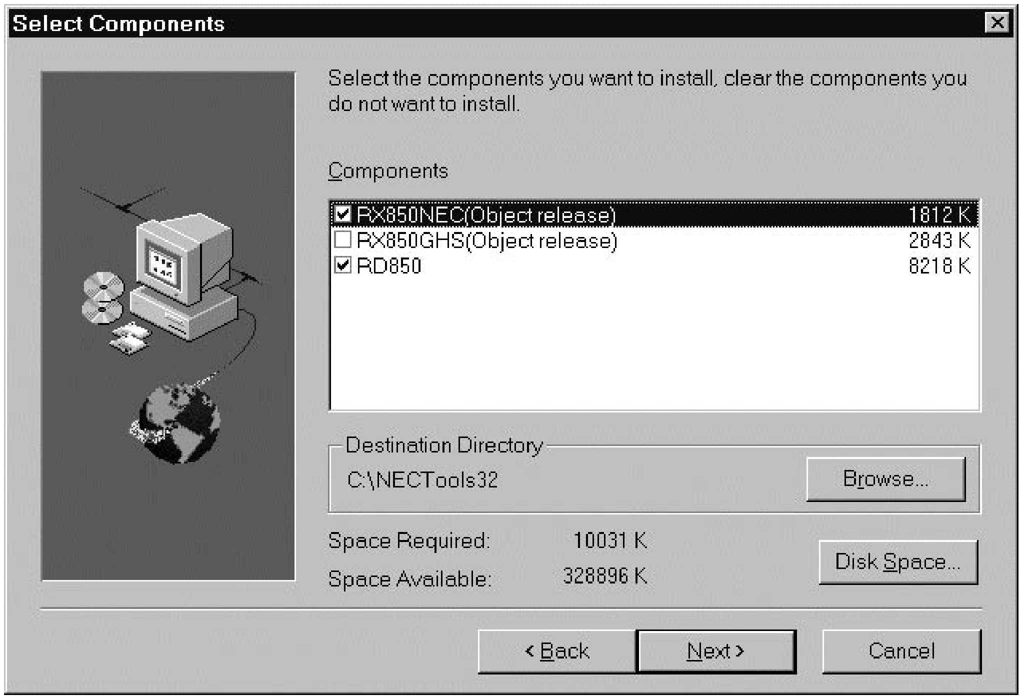
<3> The set-up program will be started after the initialization of set-up. An example of setting up the object release version is described below. Note that, with the object release version, RX850_OBJ is RX850_SRC. Click the Next > button.

<4> To install RX850, you must agree with the software license contract. To do so, click the Yes button. To abort the installation, click the No button.

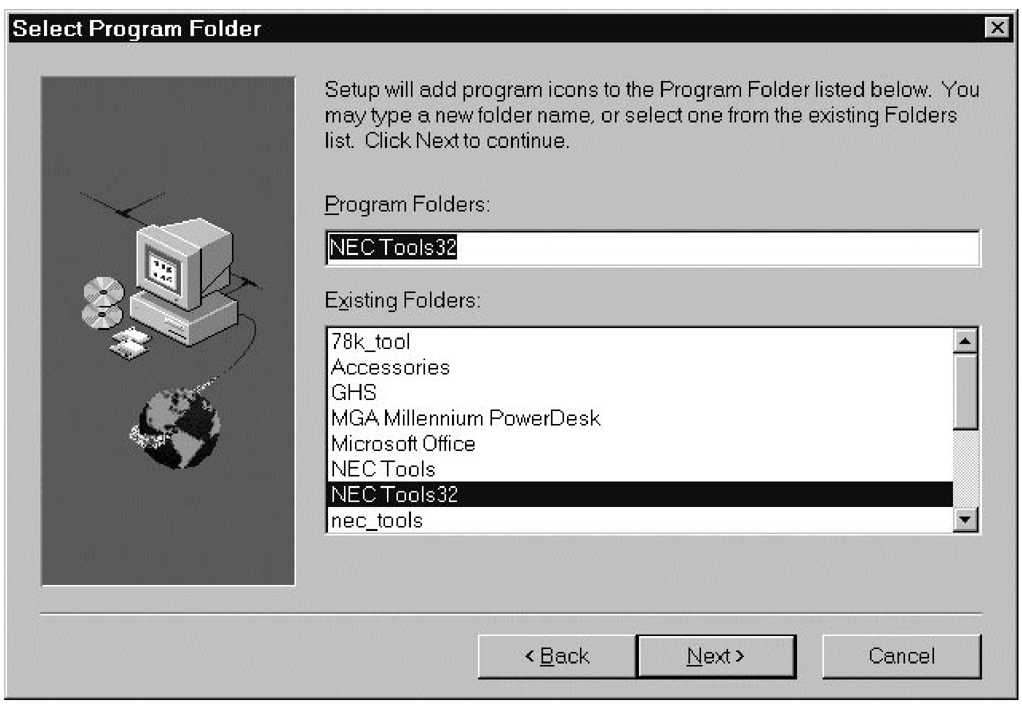
<5> Select the items to install and the drive and directory to which RX850 is to be installed. The items to install in this example are RX850 (of NEC or GHS) and RD850. Remove the check mark from the components not to be installed[Note 1]. RX850 contains two packages: one for NEC compiler “CA850” and the other for GHS compiler “CCV850”. Install either of the packages depending on the compiler being used[Note 2].
If there is any problem in the drive or directory to which the RX850 is to be installed, click the Brows button and correct the drive or directory.
After setting all the items, click the Next > button. To cancel the installation, click the Cancel button.
If the previous RX850 installation has not been uninstalled, a dialog box appears asking you if it should be uninstalled. To re-install RX850, you must first uninstall any existing installation.

Notes 1. The source release version does not include the RD850.

2. If both the NEC RX850 and GHS RX850 are installed, configurater CF850 cannot be installed correctly. Only ever install one version of RX850.



<6> Specify the name of the folder in which the icon of the RX850 is to be registered.
After specifying a group name, click the Next > button.
To cancel the installation, click the Cancel button.
The default group name is “NEC Tools32”.



<7> Final confirmation of starting installation

Confirm the items set in steps **<5>** and **<6>** above.

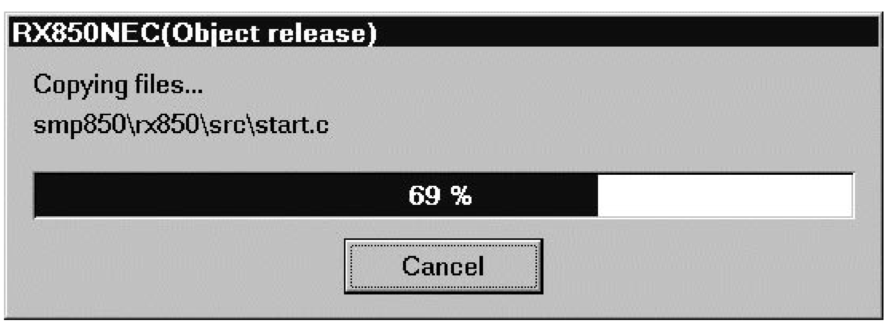
If it is unnecessary to make modification, click the Next > button.

To make a modification, return to the item to be modified by using the < Back button. If there is any problem, cancel the installation by clicking the Cancel button.

<8> Copying files

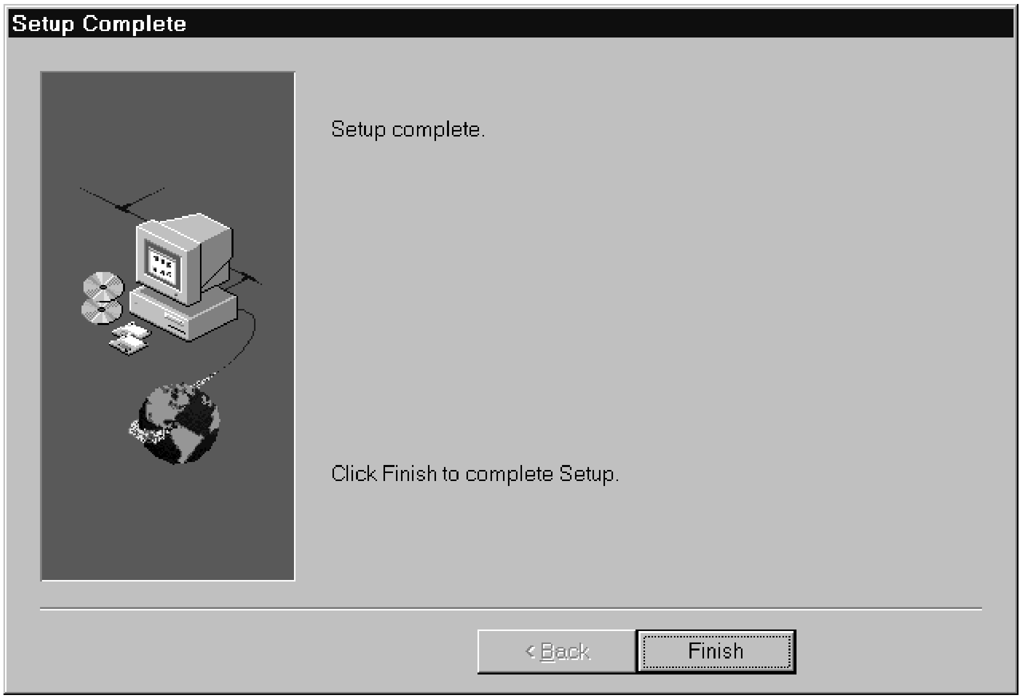
Copy the files to the directory specified in **<5>** above.

<Display of installation status>

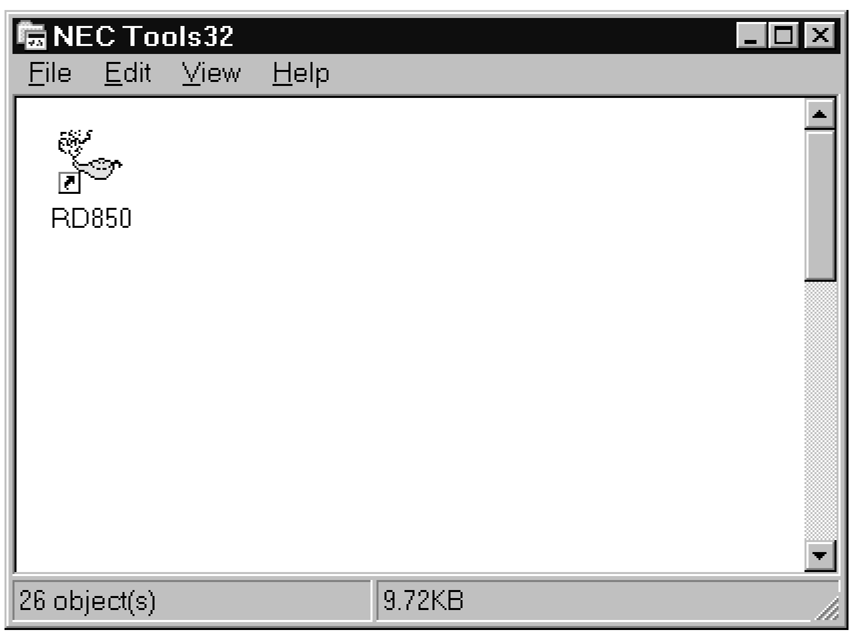


<9> Completing installation of files

When installing of the files has been completed, a dialog box indicating completion of the set-up appears. Click the Finish button. This completes the installation of RX850.



<10> When copying of the system disk has been completed, the RX850 icon appears in the “NEC Tools32” group. However, no icon is registered if only RX850 has been installed. The icon is displayed when RD850 has been installed.



<11> This completes the installation.

2.1.2 Installing the UNIX Version

This section explains how to install the UNIX version of RX850. The UNIX version is supplied on a single CD-R[Note] regardless of whether it is the object release version or the source release version.

Note CD-R was created with RockRidgeExtension of ISO9660.

<1> Log on to the host machine.

<2> Move to the install directory.
In this example, the install directory is /usr/nectools32.

```
%cd/usr/nectools32  ↵
```

Confirm that the attribute of the install directory is write.

<3> Mount the CD-R in the CD drive and close the drive.

<4> Execute the cp command to copy the files from the CD-R.

<5> Set a command search path to the bin directory.
In this example, the environmental variable path in the .cshrc file is set.

```
Set path = (.../usr/nectools32/bin)
```

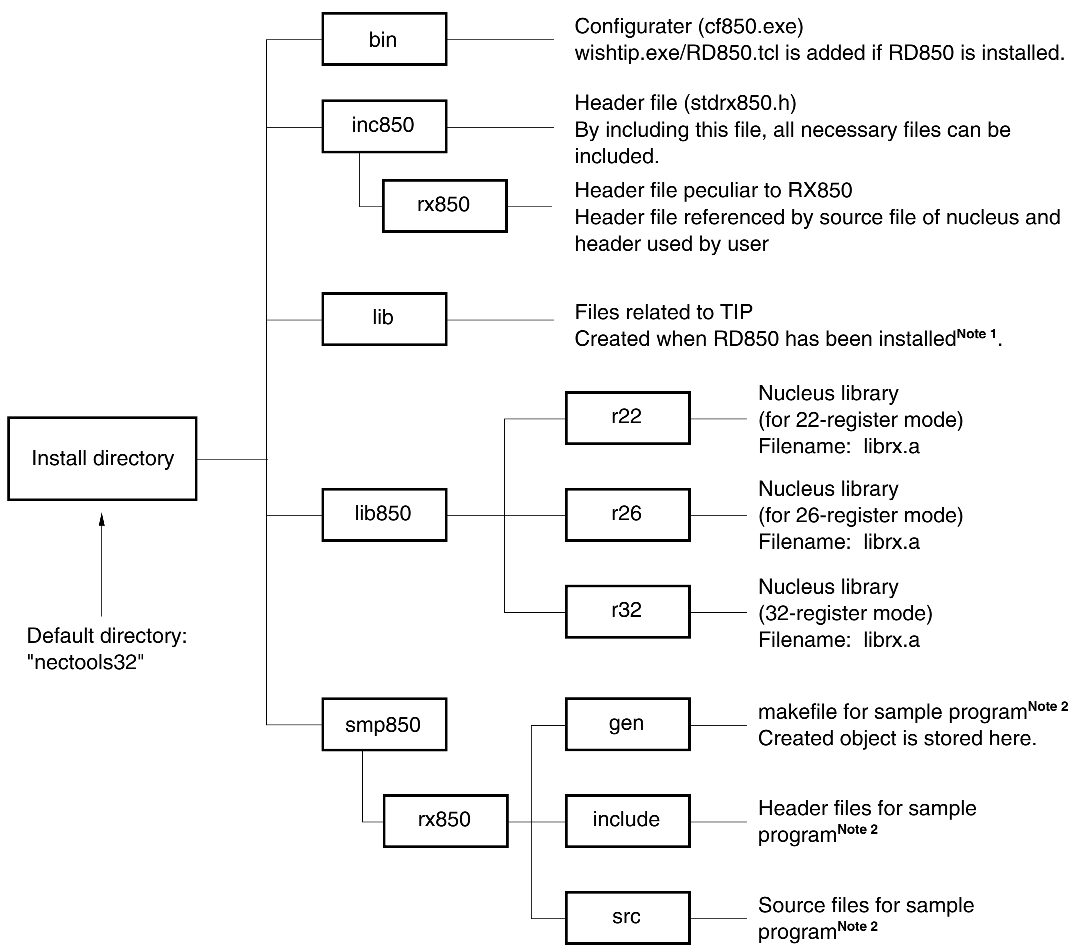
2.2 DIRECTORY CONFIGURATION

This section explains the directory configuration of the files read from the supply medium when RX850 has been installed. RX850 is supplied in the form of an object release version or a source release version. Each version is available as an NEC compiler (CA850) version and a GHS compiler (CCV850) version.

2.2.1 Object Release Version/NEC Compiler Version

Figure 2-1 shows the directory configuration when the NEC compiler version of the object release version has been installed.

Figure 2-1. Directory Configuration (Object Release Version/NEC Compiler Version)



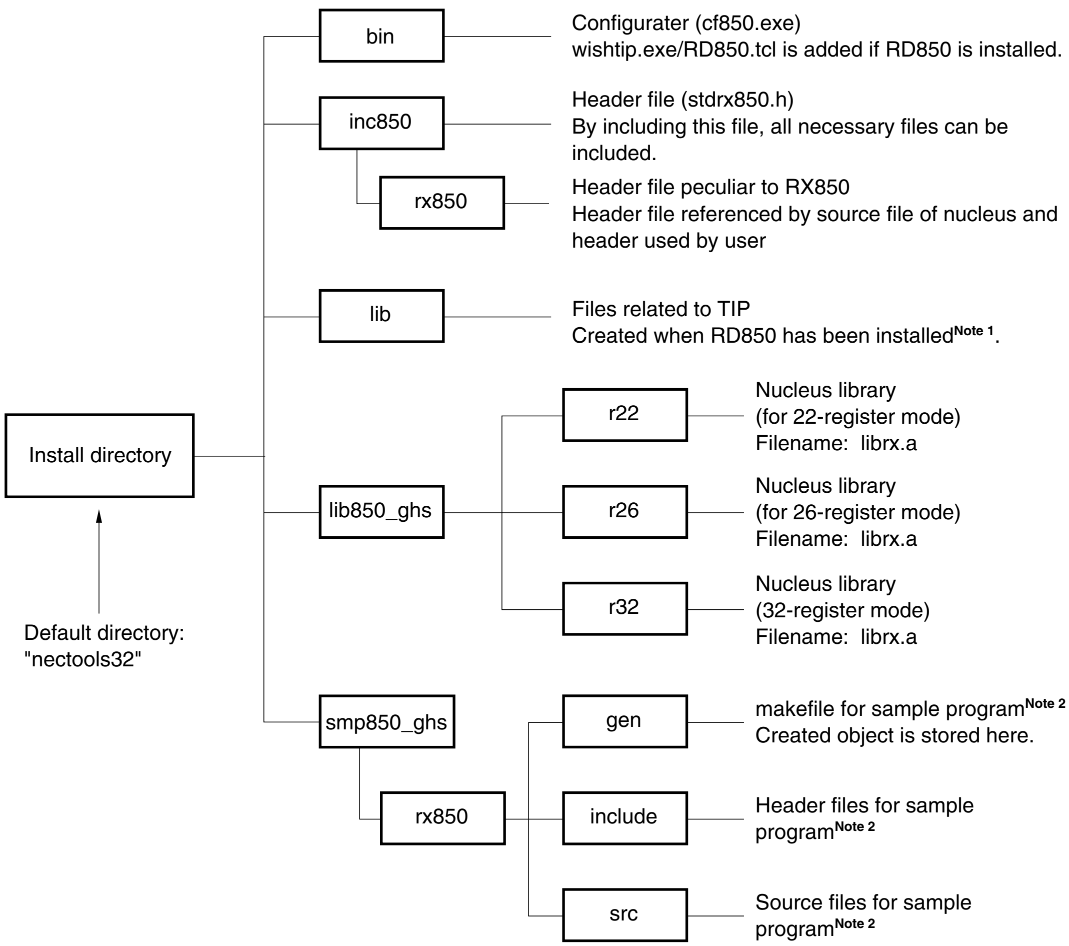
Notes 1. The RD850 is included only in the Windows-based RX850.

2. A sample program is created for the V851 and V852. If any other CPU is being used, the program can still be used by partially modifying the interrupt names and port names.

2.2.2 Object Release Version/GHS Compiler Version

Figure 2-2 shows the directory configuration when the GHS compiler version of the object release version has been installed.

Figure 2-2. Directory Configuration (Object Release Version/GHS Compiler Version)



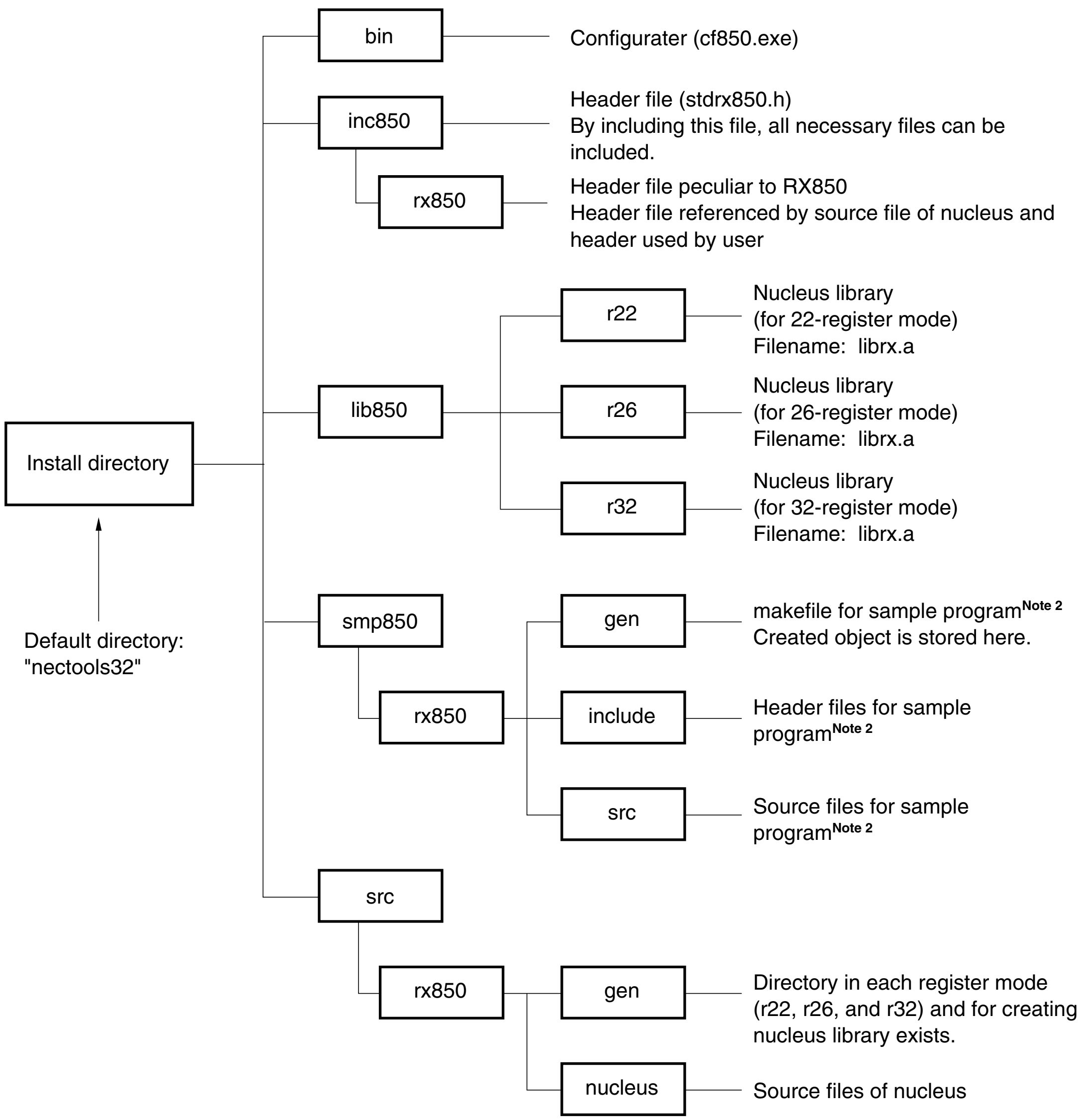
Notes 1. The RD850 is included only in the Windows-based RX850.

2. The sample program is created for the V851 and V852. If any other CPU is being used, the program can still be used by partially modifying the interrupt names and port names.

2.2.3 Source Release Version/NEC Compiler Version

Figure 2-3 shows the directory configuration when the NEC compiler version of the source release version[Note 1] has been installed.

Figure 2-3. Directory Configuration (Source Release Version/NEC Compiler Version)



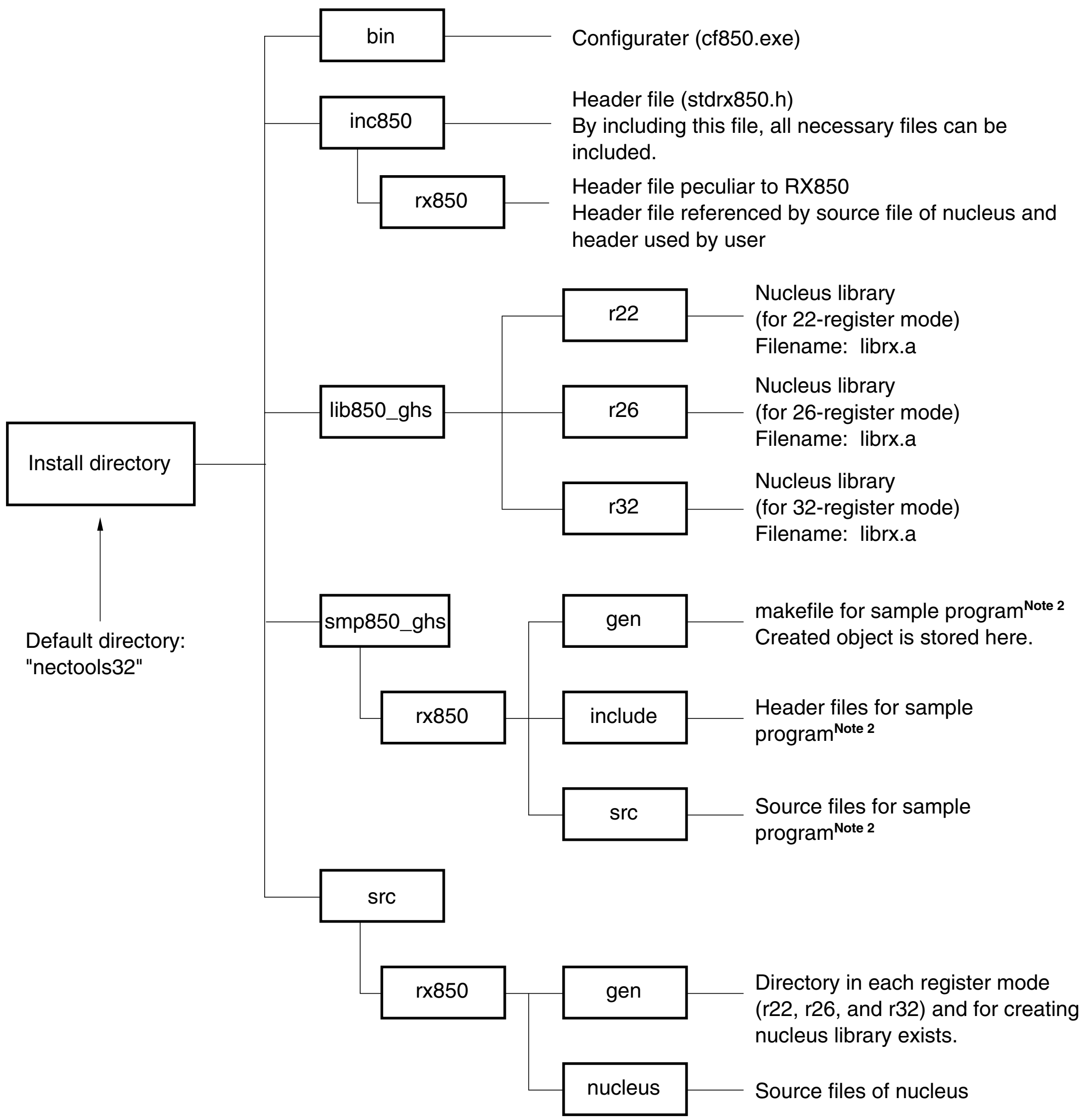
Notes 1. The RD850 is not included in the source release version.

2. The sample program is created for the V851 and V852. If any other CPU is being used, the program can still be used by partially modifying the interrupt names and port names.

2.2.4 Source Release Version/GHS Compiler Version

Figure 2-4 shows the directory configuration when the GHS compiler version of the source release version[Note 1] has been installed.

Figure 2-4. Directory Configuration (Source Release Version/GHS Compiler Version)



Notes 1. The RD850 is not included in the source release version.

2. The sample program is created for the V851 and V852. If any other CPU is being used, the program can still be used by partially modifying the interrupt names and port names.

2.3 UNINSTALLING

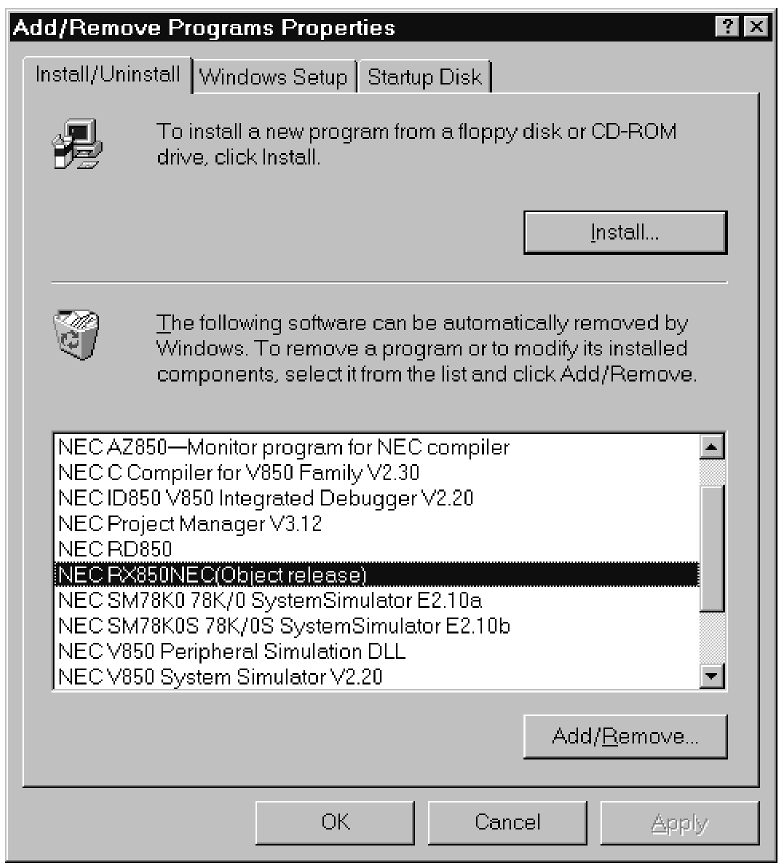
2.3.1 Uninstalling Windows Version

This section explains how to uninstall the Windows version of RX850. In the following example, it is assumed that Windows is installed in directory “b: \Windows”.

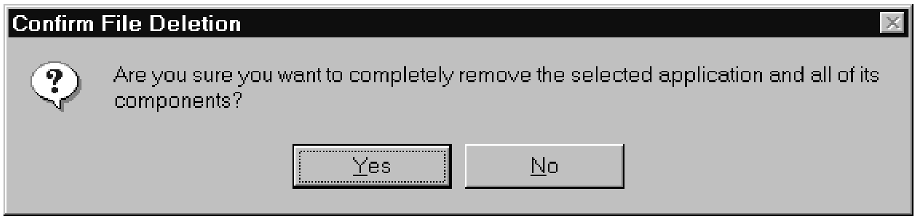
<1> Start Windows.

<2> Start “Add/Remove Programs” on the control panel.

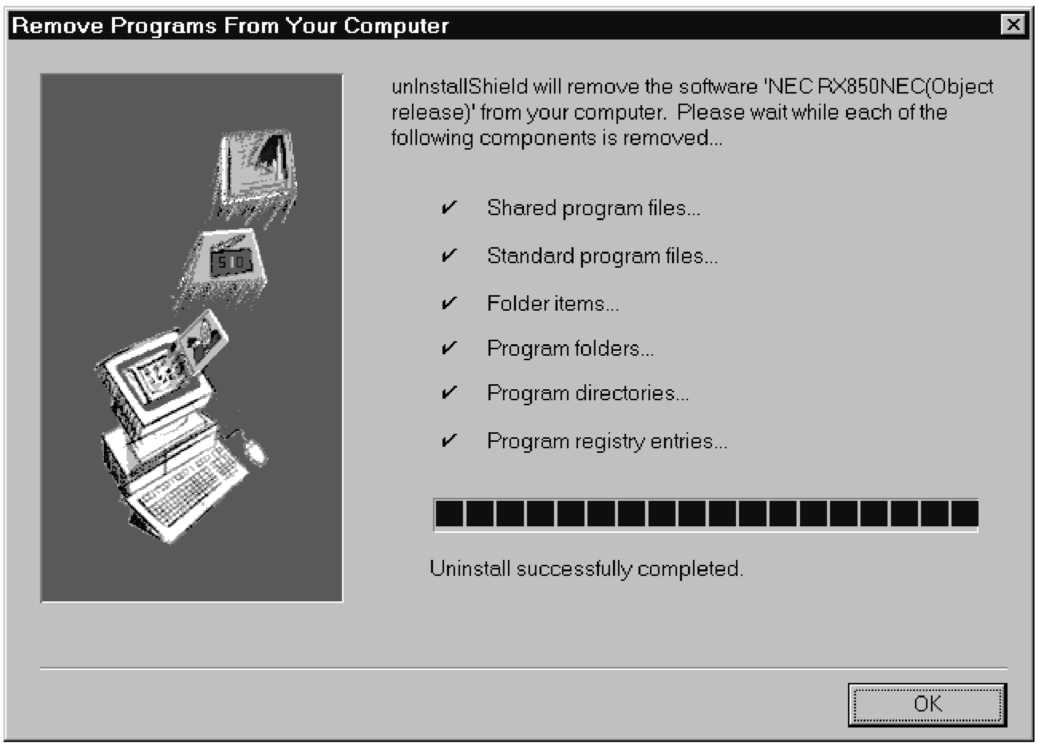
<3> Select the item to be uninstalled. To uninstall the NEC version (object version) of RX850, for example, select “NEC RX850NEC (object version)” from the list displayed when the set-up and deletion tab is selected, and then click the Add/Remove… button.



<4> The following <Confirm File Deletion> dialog box will be displayed. Click the Yes button.



<5> The program will be deleted. When the message “Uninstall successfully completed.” apperas, click the OK button. This completes the uninstallation. An example of uninstalling RX850 is shown below.



2.3.2 Uninstalling the UNIX Version

If the install directory is /usr/nectools32, the file will be deleted when the rm command is executed from the command line.

```
% rm r /usr/nectools32 ↵
```

[MEMO]

★

CHAPTER 3 SYSTEM CONSTRUCTION

This chapter explains how to construct a system.

3.1 OUTLINE

System construction involves incorporating created load modules into a target system, using the file group copied from the RX850 distribution media to the user development environment (host machine).

The system construction procedure is outlined below.

(1) Creating a configuration file

(2) Creating an information file

- System information table (SIT)
- System information header file

The information table and header file are created by using the configurater.

(3) Creating system initialization

- Boot processing
- Initialization handler

(4) Creating an idle handler

(5) Creating processing programs

- Task
- Interrupt handler
- Cyclic handler

The programs are created by using C or assembly language.

(6) Creating an initialization data save area (Only when CA850 is used)

(7) Creating a link directive file (section map file)

(8) Creating a load module

(9) Incorporating the load module into the system

Figure 3-1 shows the procedure for organizing the system when NEC's V850 Family C compiler CA850 is used. Figure 3-2 shows the procedure for organizing the system when Green Hills Software's C cross V800 compiler CCV850 is used.

Figure 3-1. System Construction (CA850)

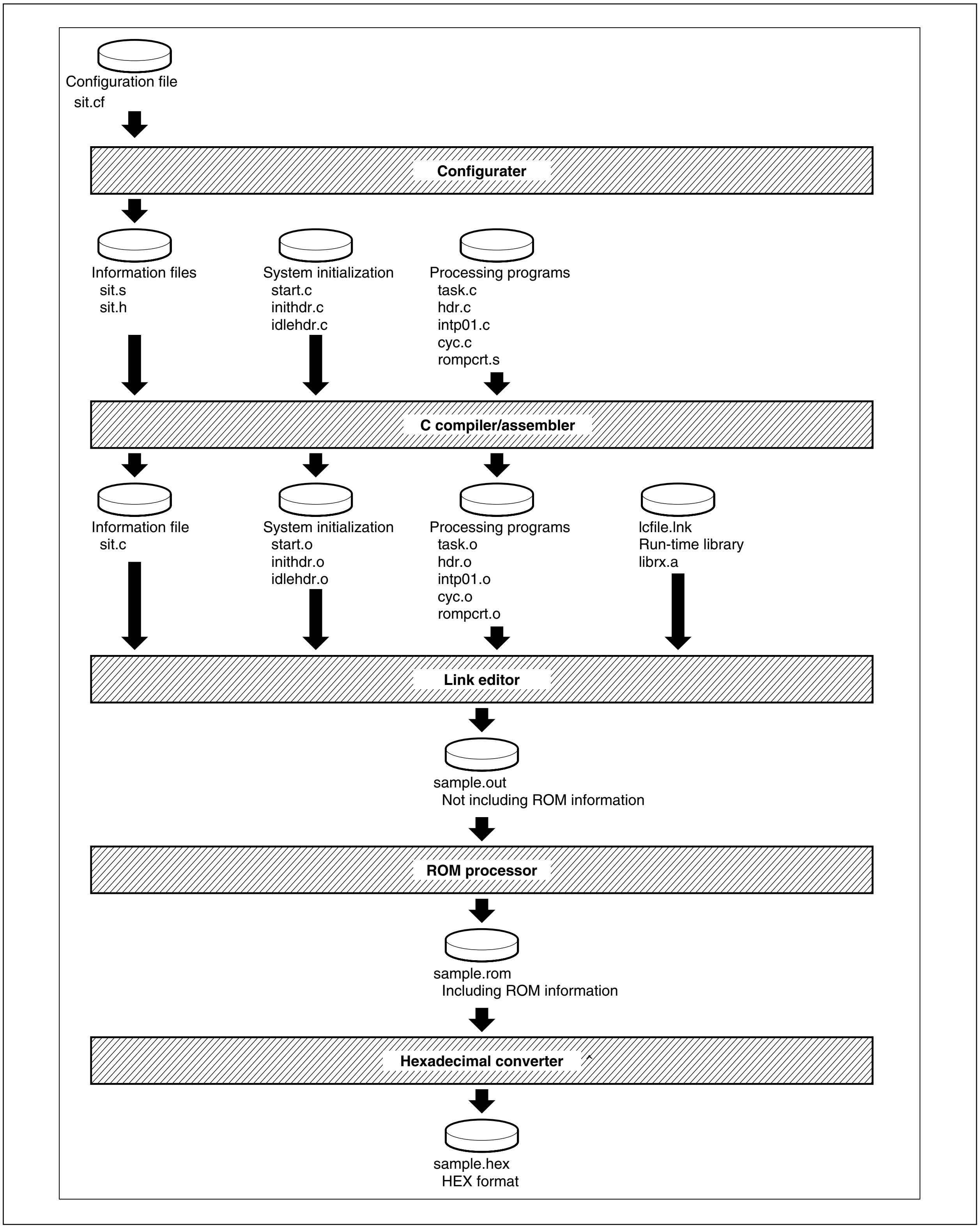
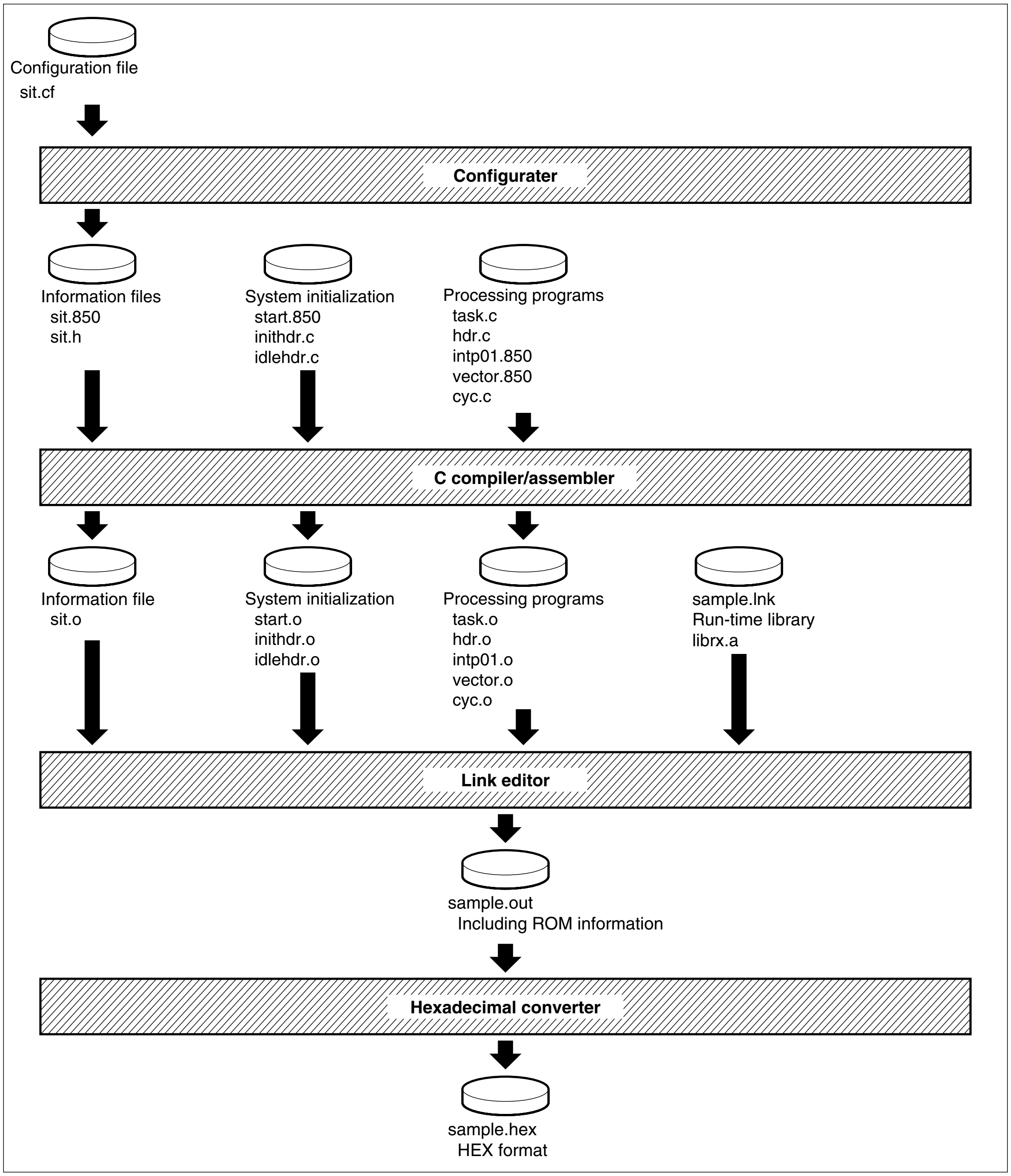


Figure 3-2. System Construction (CCV850)



The flow of organizing the system is explained based on the sample program supplied with the package. The program is stored in the following directory if the RX850 has been installed in directory nectools32.

Table 3-1. Sample Program Storage Directory

Compiler/CPU	Storage directory
For NEC/V850	nectools32\smp850\rx850\src
For GHS/V850	nectools32\smp850_ghs\rx850\src

Reference either of the above directories depending on the compiler being used.

The file extension of the sample program of the NEC version is .c even if the file is described in assembly language. This is because a macro description in C is used in an assembly source on the assumption that the program is converted into an .s file through preprocessor. For an explanation of how to start the preprocessor, refer to **User's Manual CA850 C Compiler Package – Operation (U14568E)**.

3.2 CREATING A CONFIGURATION FILE

Create an information table, called a configuration file that holds the various data used with the RX850.

This file is necessary for creating the following by using the configurater (CF850):

- System information table (SIT information)
- System information header file

For details on how to create the configuration file, see **Section 3.3**.

The "system information table" contains information on the resources of the RX850, such as tasks, semaphores, and memory pools. The "system information header file" has a description that makes the symbol names specified as resource IDs, such as those of tasks and semaphores created with the system information table, correspond to the actual symbol ID numbers, by using the #define instruction.

The sample configuration file is

- sit.cf

For the contents and syntax of the configuration file, see **Section 5.2**.

3.3 CREATING INFORMATION FILES

Create the “system information table (SIT information)” and “system information header file” from the configuration file (sit.cf) created as described in Section 3.2 above, by using the configurater (cf850).

The following filenames are recommended:

[System information table]

- NEC version: sit.s
- GHS version: sit.850

[System information header file]

- sit.h

To organize an application using RX850, assemble sit.s (sit.850) and link the created object. The C source must include sit.h.

For details on how to use the configurater (cf850.exe) that is used to create these files, see **Chapter 6**.

3.4 CREATING SYSTEM INITIALIZATION BLOCK

The system initialization block is a function consisting of program segments that are dependent upon the user's target system. This function is used to facilitate transplantation and customization.

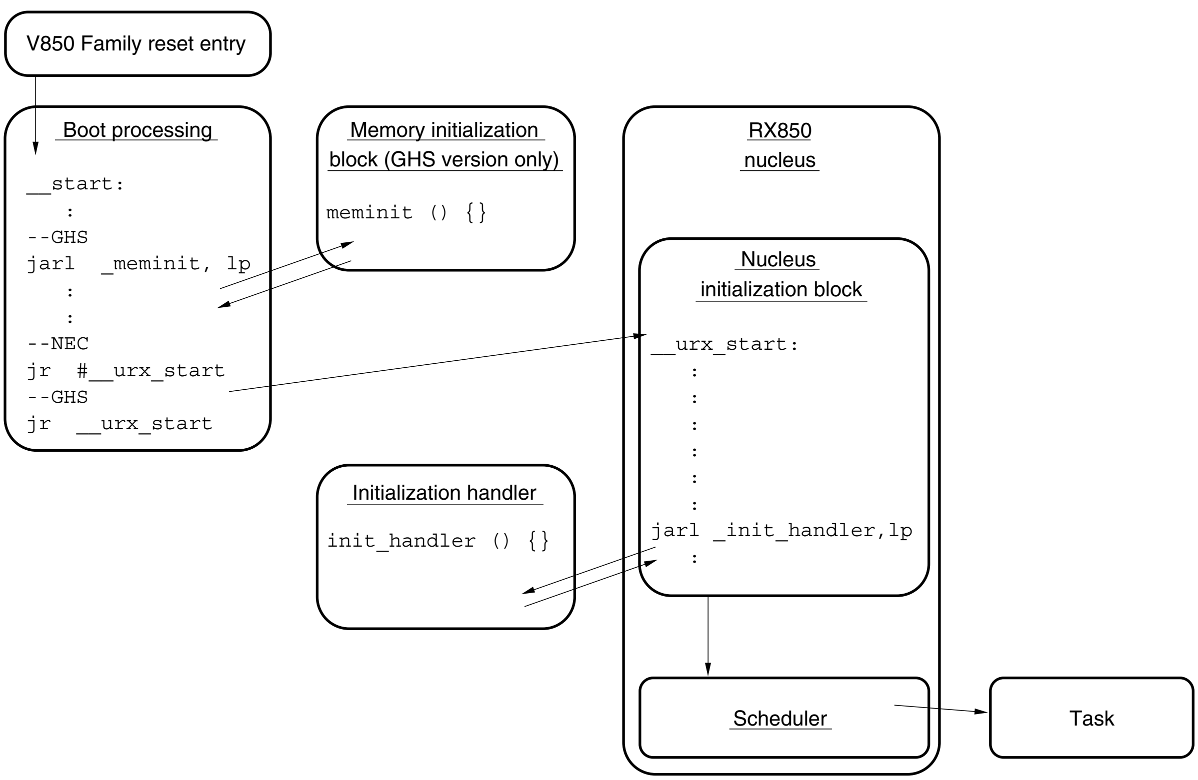
The sample file is as follows:

Table 3-2. Configuration of System Initialization Block

Sample file name	Type	Function name	Feature
start.c (NEC version) start.850 (GHS version)	Boot processing	`start`	Boot processing of system
inithdr.c	Initialization handler	`init_handler`	Initialization processing of hardware, etc.
vector.850 (GHS version)	Interrupt entry	None	Processing to branch to interrupt processing

The rough flow of the system initialization block is illustrated below.

Figure 3-3. Flow of System Initialization Block



Each processing is explained next.

3.4.1 Boot Processing

The boot processing is assigned to the set entry (handler address: 0x0) of the V850 Family and is the system initialization processing that is executed first.

The description following label "`__start`" in sample file start.c (start.850) is the entity of the boot processing. The instructions that cause execution to jump from the reset entry to this label are as follows. These instructions are in the same start.c file for the NEC version and in the vector.850 file for the GHS version.

[NEC version]

```
    .section    "RESET"
    jr          __start
```

[GHS version]

```
    .org      0x00000000
    .grobl    __reset
__reset:
    .extern   __start
    jr     __start
```

The lowest line of the instructions is assigned to the handler address [0x0]. When reset is executed, therefore, these instructions are executed, execution jumps to `__start`, and the boot processing is executed.

As part of the boot processing, the following must be performed.

1. Setting of tp (text pointer), gp (global pointer), and ep (element pointer)
2. Setting of sp (stack pointer) used for boot processing
3. Transferring control to the RX850 nucleus initialization block by jumping to the `_urx_start` symbol by using the jr instruction

In addition to the above, processing (`jarl_meminit, lp`) that causes execution to jump to the `meminit ()` function, which is a "memory initialization block", is executed between 2 and 3 with the sample of the GHS version. This `meminit ( )` function initializes the bss area and copies the default value data. With the sample of the NEC version, the bss area on RAM is initialized (cleared to 0) in start.c. With the NEC version, the default value data is copied by creating an area of the default value data (rompcrt.s) and by using function `_rcopy ( )`. For details, refer to **User's Manual CA850 C Compiler Package – Operation (U14568E)**.

The sample of the GHS version sets the stack pointer as in 2 above, though this is not performed with the sample of the NEC version. The stack pointer to be set is independent of the stack for tasks and interrupt handlers. After the RX850 has been started, the stack used by tasks and interrupt handlers is managed by the RX850 itself, by using the system information table (SIT), and the stack pointer is automatically switched by means of task switching or interrupts. Therefore, the stack pointer specified in the boot processing is used before the RX850 is started. This stack pointer is used, for example, when execution jumps to a function and that function has data to be saved to the stack. This stack pointer is used if it is necessary to use the stack with the `meminit ( )` function of the sample.

At the end of the boot processing, processing 3 is necessary. Perform the following processing.

[NEC version]

```
.extern   __urx_start
jr        __urx_start
```

[GHS version]

```
.extern   __urx_start
jr        __urx_start
```

The description in the RX850 following symbol "__urx_start" is the nucleus initialization processing of the RX850. After the boot processing has been completed, transfer control to the nucleus initialization processing by using the jr instruction. The initialization processing creates resources and executes initialization based on the "system information table (SIT)" created from the configuration file.

NEC recommends changing the description of the boot processing to the environment suitable for the user, based on the boot processing of the sample.

3.4.2 Nucleus Initialization Block

The nucleus initialization block is an internal routine of the RX850 that is executed after completion of the boot processing. This block creates the RX850 system management block, and creates and initializes information on things such as tasks, semaphores, and memory pools, based on the "system information table (SIT)" created from the configuration file.

Once initialization has been completed in the nucleus initialization block, an initialization handler is called. The function name of the initialization handler of the RX850 is determined to be `init_handler` (label "`_init_handler`" if the description is made in an assembly language). It is therefore necessary to create a function with this name. For details of this function, see **Section 3.4.3**.

When control has been returned from the initialization handler, the scheduler is started, and then RX850 is started.

3.4.3 Initialization Handler

The initialization handler is a function (handler) that is called from the nucleus initialization block. Describe the processing to be performed before starting RX850 and the hardware processing in this handler. In the initialization handler, it is possible to issue system calls that can be issued by an interrupt handler or cyclic handler.

The initialization handler is a function of FP type without an argument. Its function name is determined to be `init_handler` (label "`_init_handler`" if the description is made in assembly language). It is therefore necessary to create a function with this name. Even if the processing is not necessary, create the function as a function that performs no processing. Upon the termination of the handler, return control to the nucleus initialization processing by using the return instruction.

The initialization handler of the sample initializes the peripheral I/O, sets the interrupt control register, and starts tasks. With the NEC version, the default data value can be copied into the initialization handler. For details of how to copy the default value data, refer to **User's Manual CA850 C Compiler Package – Operation (U14568E)**.

3.4.4 Interrupt Entry

An interrupt entry is an instruction that is executed if an interrupt occurs, and is assigned to the “interrupt handler address” of the V850 Family. The interrupt entry must be defined for all the interrupts used by the user, and must be described in assembly language. The interrupt handler of the sample is described in “start.c” for the NEC version, and in “vector.850” for the GHS version.

The interrupts of the RX850 are handled by two types of handlers: “directly activated interrupt handler” and “indirectly activated interrupt handler”. Describe the interrupt entry only when the directly activated interrupt handler is used.

In the entry, describe a branch instruction in the same manner as an ordinary interrupt entry. In the sample, interrupt “INTP01 (handler address: 0x130)” is the example of the directly activated interrupt handler. With the NEC version, the .section pseudo instruction is used. With the GHS version, the .org instruction is used. For details of each instruction, refer to **User’s Manual CA850 – Assembly Language (U14567E)** for the NEC version. For the GHS version, refer to the manual related to the GHS language. The entry of the directly activated interrupt handler is as follows:

[NEC version]

```
.section   "INTP01"
jr   entry_P01
```

[GHS version]

```
.org   0x00000130
  jr   entry_P01
```

If jump destination label “`entry_P01`” is defined in a separate file, use the .extern instruction.

Label “`entry_P01`” is defined in “`intp01.c`” of the NEC version or “`intp01.850`” of the GHS version, and causes execution to jump to the handler entity (`inthdrP01 ( )`) after the preprocessing and post-processing of the directly activated interrupt handler are described (in macro). For details of how to describe a directly activated interrupt handler, refer to **User’s Manual RX850 – Basics (U13430E)**.

3.5 CREATING IDLE HANDLER

The idle handler is a function (handler) that is started when there is no task to be scheduled in an application. By using this function, the idle status of the system can be checked and the power-saving function of the CPU can be used. For example, if processing that places the CPU in HALT mode is described in this idle handler, the CPU can be placed in HALT mode in the idle status of the system.

The idle handler is a function of FP type without an argument and its function name is determined to be `idle_handler` (label "`_idle_handler`" if the description is made in assembly language). Therefore, create a function with this name. Create the idle function as a function that executes nothing even if idle processing is not needed. However, because an interrupt is used to exit from the handler, the processing that enables the interrupt (EI instruction) must be described in the handler.

The idle handler of the sample is described in the following file:

- idle.c

The V850 Family supports "HALT mode", "IDLE mode", and "STOP mode" as power-saving functions. To place the CPU in each of these modes, particular processing is required. Therefore, a sample corresponding to each mode is supplied. The correspondence between each mode, sample file, and function name is as shown below.

Table 3-3. Sample of Power-Saving Function

Mode	Sample file	Function name
HALT mode	halt.c (NEC version)/halt.850 (GHS version)	`__halt()`
IDLE mode	idle.c (NEC version)/idle.850 (GHS version)	`__idle()`
STOP mode	stop.c (NEC version)/stop.850 (GHS version)	`__stop()`

To compile and assemble these files with the NEC version, a register mode must be specified (the .option pseudo instruction between "#if" and "#elif or #endif" at the beginning of the file is the instruction needed for selecting a register mode). The default register mode is 32-register mode. To select the 22- or 26-register mode, specify the following option for assembly.

Register mode	Assemble option (NEC version)
22-register mode	–D__850_22__
26-register mode	–D__850_26__

With the GHS version, files do not select a register mode.

For the details of the power-saving functions of the V850 Family, refer to the User's Manual – Hardware for each device.

3.6 CREATING PROCESSING PROGRAM

Create a processing program, i.e., application.
The processing units of the application necessary for the RX850 are broadly classified into the following:

- Task
- Directly activated interrupt handler
- Indirectly activated interrupt handler
- Cyclic handler

The contents of the sample are shown below.

Table 3-4. Configuration of Processing Program

Sample file name	Type	Function name	Feature
task.c	Task	task1 task2	Task entity
handler.c	Cyclic handler Indirectly activated interrupt handler Directly activated interrupt handler	cychdr0 cychdr1 inthdrP00 inthdrP01	Each handler processing

If a processing program described in C issues a system call, include header file “stdrx850.h” supplied by the RX850. This file contains the definition necessary for using the system call.

The header file “sample.h” (in nectools32\smp850\rx850\include) of the sample includes port.h that defines the above stdrx850.h and the peripheral I/O area of the V850. As necessary, define the constants used by a function in the header file, and include the header file in the program.

3.7 CREATING INITIALIZATION DATA SAVING AREA

When NEC's "CA850" C compiler is used, it is necessary to create an area for saving the initialization data. This is because it is necessary to store the initialization data to ROM and to copy the default values of the data to RAM before executing a program. Creating a saving area for the initialization data involves reserving a ROM area to which the initialization data is to be stored.

For details of how to create this area, see the description of "ROM-embeddable processor" in **User's Manual CA850 C Compiler Package – Operation (U14568E)**.

With the GHS version, this processing is performed in function `meminit ( )` of the sample.

3.8 CREATING LINK DIRECTIVE FILE (SECTION MAP FILE)

Create a link directive file (section map file) containing the "section information" and "address information" referenced by the linker when it links modules. The following sample files are link directive files.

- lcfile (NEC version)
- sample.lnk (GHS version)

With the Windows version, only filename "sample" is displayed because the extension of sample.lnk is the same as that of the short-cut.

The sections listed in the table below are essential for the RX850.

Table 3-5. Essential Sections of RX850

Section name	Type of area
`.sit`	System information area
`.text`	RX850 system call location area
`.pool0`	System memory pool 0
`.pool1`	System memory pool 1

`.sit` section and `.text` section are text-attribute sections. The `.pool0` and `.pool1` sections are located in the RAM area. This information must be defined in the link directive file (section map file).

Creating the `.sit` section and `.pool0` section is essential. Because the RX850 is designed to access these two sections at address 0 with a single instruction, these sections must be located in a range of ±32 Kbytes from address 0 (`0xffff8000` to `0x7fff`). If these sections are not within this range, the management block of the RX850 cannot be accessed correctly. The architecture of the V850 Family recommends locating `.pool0` in the internal RAM. For details of the `.pool0` and `.pool1` sections, see **Section 4.1**.

As described in the sample file, define the information on the location of these sections. The part of the sample that describes the location of these sections is shown below.

[NEC version]

```
TEXT     : !LOAD ?RX {
         .sit     = $PROGBITS     ?AX.sit;
         .text    = $PROGBITS     ?AX.text;
};
                  :
                  :
EDATA    : !LOAD ?RW V0x00100000{
         .pool1   = $NOBITS       ?AW .pool1;
};
                  :
                  :
IDATA    : !LOAD ?RW {
         .pool0   = $NOBITS       ?AW .pool0;
         .data    = $PROGBITS     ?AW .data;
         .sdata   = $PROGBITS     ?AWG .sdata;
         .sbss    = $NOBITS       ?AWG .sbss;
         .bss     = $NOBITS       ?AW .bss;
};
                  :
```

[GHS version]

```
-sec {
         .sit     0x0000160        :
         .text                     :
                  :
                  :
         .pool1   0x0100000        :
         .pool0   0x0ffe000        :
                  :
```

In addition, define sections related to the RAM area, such as `.data/.bss` section, and those related to the ROM area, such as the `const` section, as necessary. NEC recommends changing the description of the link directive file (section map file) in the environment suitable for the user.

For details on how to describe the link directive file, refer to **User's Manual CA850 C Compiler Package – Operation (U14568E)**. For details on how to describe the section map file, refer to the manual related to the GHS language.

3.9 CREATING LOAD MODULE

Next, create a load module, i.e., executable module.

Link the objects (.o file) created by compiling and assembling the C source file and assemble source file, based on the link directive file (section map file) created in Section 3.8.

It is necessary to reference the following library when linking applications using the RX850.

Library	Contents
librx.a	Nucleus library

A separate library is provided for each of the 22-, 26-, and 36-register modes. Use the appropriate library for the register mode being used.

When link is successful, an executable module (.out file) is created. At this stage, the executable module can be read to the debugger to execute the application.

The load module file created by the linker correctly locates the initialization data in RAM. If initialization data exists in the application of the NEC version, a module that reserves an initialization data saving area and which incorporates a copy routine must be created. In this case, a load module via a ROM-embeddable processor must be created for the load module created by the linker.

For details on how to use the ROM-embeddable able processor and for the details of the copy routine, refer to “ROM-embeddable processor” in **User’s Manual CA850 C Compiler Package – Operation (U14568E)**.

3.10 EMBEDDING IN SYSTEM

Embed the completed load module file in the system.

To do so, the load module file created in Section 3.9 must be converted into a hex file.
By using the hex converter provided with each of the NEC and GHS versions, create a hex file of the necessary format. Then, embed this file into the system by using a ROM writer.

CHAPTER 4 MEMORY AND ESTIMATING ITS CAPACITY

This chapter explains how to manage the memory (RAM) of the RX850 and the capacity of the memory used.

4.1 `.pool0` and `.pool1`

The RAM area used for the RX850 consists of two sections defined in RAM: "`.pool0` section" called system memory pool 0 and "`.pool1` section" called system memory pool 1. These sections are defined in RAM by the link directive file (section map file).

The information allocated to the `.pool0` section and `.pool1` section is as follows:

Table 4-1. Information Allocated to `.pool0` and `.pool1` Sections

Section	Allocated information
`.pool0`	System base table (SBT) Ready queue Each management block Task stack Interrupt handler stack (system stack) Variable-length memory pool Fixed-length memory pool
`.pool1`	Task stack Interrupt handler stack (system stack) Variable-length memory pool Fixed-length memory pool

Because the system information of the RX850 is allocated to the `.pool0` section, creating this section is essential. The `.pool1` section can be used as a stack or memory pool. If the `.pool0` section suffices, however, `.pool1` does not have to be created.

From which of `.pool0` or `.pool1` the task stack, interrupt handler stack, and memory pool are to be reserved is specified by the configuration file. For details on how to specify this, see **Chapter 5**.

The location of the `.pool0` section is limited. It must be located in a range of ±32 Kbytes from address 0 (`0xffff8000` to `0x7fff`) because the RX850 accesses `.pool0` section at address 0 with a single instruction. Unless this section is located within this range, the management blocks of the RX850 cannot be accessed correctly. Because the `.pool0` section must be located in RAM, it is recommended that it be located in the internal RAM area in this range. This limitation does not apply to the `.pool1` section.

4.2 MEMORY CAPACITY OF MANAGEMENT AREA

This section explains the size of the system base table, ready queue, and each management block of the RX850. These are reserved from `.pool0` section. Table 4-2 shows the size of the management area used by each of the objects and how to estimate the area size.

Table 4-2. Size of Object Management Area

Object	Size of management area used (per object)	Calculating size
System base table (SBT)	20 bytes	Fixed size (20 bytes) regardless of the status of the system (number of tasks).
Ready queue	2 to 32 bytes	Number of levels of priority specified by configuration file + 1 (unit: bytes)
Task execution right	18 bytes	Extra usable area is added in the following cases: 1. If the task execution right group is used • Estimating the additional size: Number of execution right groups specified as "having a possibility of waiting for task execution right" × 1 (units: bytes) 2. If either or both of the variable-size memory pool and event flag are used • Estimating additional size: Number of all task execution rights × 8 (units: bytes)
Task	9 bytes	–
Event flag	5 bytes	–
1-bit event flag	2 bytes	–
Semaphore	2 bytes	–
Mailbox	8 bytes	When a mailbox having the `TA_MPRI` attribute is used, a 1-byte area is used for each mailbox having the `TA_MPRI` attribute.
Variable-size memory pool	16 bytes	A memory pool area is necessary in addition to management area.
Fixed-size memory pool	4 bytes	A memory pool area is necessary in addition to management area.
Cyclic handler	10 bytes	–

4.3 MEMORY CAPACITY OF STACK

4.3.1 Task Stack Area

The stack area for each task is allocated by adding the following size (1) through (3) to the size of the area used if a task is considered to be a normal C function (such as a save area for register variables).

(1) Context area

If dispatching takes place by issuing a system call, etc., the stack of the size is consumed according to each of the following register modes.

32-register mode: 56 bytes
26-register mode: 44 bytes
22-register mode: 36 bytes

(2) Variable-size memory pool work area

Although it is assumed that the RX850 does not use the stack while a system call is being processed, it uses up to 12 bytes of the task stack only when a variable-size memory pool is being used.
The system calls that uses a variable-size memory pool are shown below.

`get_blk, pget_blk, tget_blk, rel_blk, ter_tsk, rel_wai`

(3) Temporary register save area, used if an interrupt occurs

As an area used to save the temporary registers if an interrupt occurs, the stack of the given size is consumed according to each of the following register modes.

32-register mode: 68 bytes
26-register mode: 56 bytes
22-register mode: 48 bytes

If the directly activated interrupt handler from which execution is returned by `reti (RTOS_IntExit)` is described, the size of the stack consumed by the handler must be added.

4.3.2 Stack (system stack) Area for Interrupt Handler

The RX850 switches the stack area from the task stack to the handler stack when each handler (interrupt handler, cyclic handler, initialization handler, or idle handler) is activated. The size of the stack necessary at this time is calculated as follows:

(1) Initialization handler

By considering the initialization handler as a normal C function, the size of the stack consumed by that function is necessary.

Because the interrupts are disabled while the initialization handler is executed, and because the processing is not switched to the other handlers or tasks while the handler is being executed, the size of the stack does not have to be considered if the size of the stack consumed by the interrupt handler is sufficiently large.

(2) Idle handler

By regarding the idle handler as a normal C function, the size of the stack consumed by that function is necessary. Because it is assumed that an interrupt occurs while the idle handler is being executed, the size must be allocated separately from the stack consumed by the interrupt handler or cyclic handler.

(3) Interrupt handler

The interrupt handler consumes a handler stack consisting of 8 bytes when the handler is activated. If the interrupt handler is nested, another 8-byte stack is consumed each time the handler is nested. Therefore, "the maximum number of times of nesting of the interrupt handler × 8" bytes of stack area is necessary in addition to the size of the stack (total size in case of nesting) consumed by the interrupt handler as a C function.

(4) Timer handler

The timer handler is provided as an indirectly activated interrupt handler, and 20 bytes of the stack are consumed when the timer handler is activated.

However, even if the timer handler is nested while the timer handler is being executed, a new 20-byte stack area is not necessary.

(5) Cyclic handler

The cyclic hander is provided as a C function that is called from the timer handler. Therefore, a stack of the size consumed by this handler is consumed.

(6) Variable-size memory pool

If a system call related to the variable-size memory pool is issued in the same manner as the task stack, a stack of up to 12 bytes is consumed. If system calls are issued from the handler, these system calls may be issued again from a nested handler. Therefore, up to "maximum number of nested interrupt handlers × 12 bytes" must be added.

CHAPTER 5 CONFIGURATION FILE

This chapter explains the configuration file and how to describe it.

5.1 CONFIGURATION FILE

To organize a system using the RX850, information holding the necessary data (such as system information and resource information) is necessary. This information is called a system information table (SIT).

The system information table is written in assembly language and is an enumeration of data in a specified format. It is possible to describe the system information table by using an editor. This however, takes time and effort because modifying or adding new data to the system information table is extremely difficult.

Therefore, an application “configurater (CF850)” is supplied.

This application converts a “configuration file” in which the information on the system and resources of the RX850 is described in an original format into a system information file. The user can obtain the system information table by creating a configuration file and by using the configurater.

The configurater outputs two files from the configuration file: “system information table” and “system information header file”. The “system information header file” describes the correspondence between the symbol names specified as resource IDs, such as created tasks and semaphore, and actual ID numbers with #define instruction.

For details on how to start the configurater, see **Chapter 6**.

How to describe the configuration file is explained next.

5.2 DESCRIBING A CONFIGURATION FILE

This section explains how to describe a configuration file that is to be input to the configurater.

(1) Character code

Create the configuration file using ASCII code.
The system distinguishes between lowercase and uppercase letters.
Use a space or tab to delimit words (e.g., numerics, symbol names, and keywords). Enter a line feed (LF) to delimit statements.

Caution For Japanese language coding, EUC codes and shift JIS codes can be used only for comments.

(2) Numeric

Unless otherwise specified, any 32-bit value (`0x0` to `0xffffffff`) can be specified for a numeric.

(3) Symbol name

A symbol name can be coded using alphanumeric characters (up to 31 characters).
The symbol name, however, must begin with _ or an alphabetic character.

(4) Comment

In the configuration file, the portion from -- to the end of a line is handled as a comment.

(5) Continuation lines

In the configuration file, a backslash coded at the end of a line indicates that that line is continued on the next line.
Note that the character immediately before the backslash must be either a space or a tab.

(6) Keywords

The configurater reserves the following character strings as keywords.
These character strings must not be used for other purposes.

```
auto        clkhdr      cyc         di          ei
flg         flg1        inthdr      intstk      maxpri
mbx         mpf         mpl         no_use      no_wait
pool0       pool1       RX850       rxsers      sem
ser_def     sit_def     TA_MFIFO    TA_MPRI     TCY_ULNK
TCY_OFF     TCY_ON      trace       tsk         tskgrp
TTS_DMT     TTS_RDY     V310
```

5.3 CONFIGURATION INFORMATION

The configuration information that is described in a configuration file is divided into the following two main types.

- Real-time OS information
  Data relating to the real-time OS being used.
- System Information Table (SIT) information
  Data that is necessary to the operation of RX850.

5.3.1 Real-Time OS Information

The real-time OS information that is described in a configuration file consists of the following item.

(1) RX series information

The following data is described as RX series information.

- Real-time OS name
- Version number

5.3.2 SIT Information

The SIT information that is described in a configuration file consists of the following twelve items.

(1) System information

Define the following items as system information:

- System stack information
- Clock interrupt source
- Trace information

(2) System maximum value information

Define the following item as system maximum value information:

- Task priority range

(3) Task execution right group information

Define the following items of task execution right group information for each task execution right group:

- Name of task execution right group
- Stack information for task execution right group
- Wait state information for task execution right group

(4) Task information

Define the following items of task information for each task:

- Task name
- Start address of task
- Task stack information
- Initial priority of task
- Initial state of task
- Task activation code
- Interrupt state

(5) Semaphore information

Define the following items of semaphore information for each semaphore:

- Semaphore name
- Initial resource count for semaphore

(6) Event flag information

Define the following item as event flag information for each event flag:

- Event flag name

(7) One-bit event flag information

Define the following item for each one-bit event flag:

- One-bit event flag name

(8) Mailbox information

Define the following items of mailbox information for each mailbox:

- Mailbox name
- Message queuing method

(9) Indirectly activated interrupt handler information

Define the following items of indirectly activated interrupt handler information for each indirectly activated interrupt handler:

- Interrupt source
- Start address of indirectly activated interrupt handler

(10) Fixed-size memory pool information

Define the following items as fixed-size memory pool information for each fixed-size memory pool:

- Fixed-size memory pool name
- Memory block information
- Total number of memory blocks

(11) Variable-size memory pool information

Define the following items as variable-size memory pool information for each variable-size memory pool:

- Variable-size memory pool name
- Variable-size memory pool information

(12) Cyclic handler information

Define the following items of cyclic handler information for each cyclic handler:

- Cyclic handler name
- Start address of cyclic handler
- Initial activity state of cyclic handler
- Activation interval for cyclic handler

5.4 SPECIFICATION FORMAT FOR REAL-TIME OS INFORMATION

The following shows the specification format that must be observed when describing real-time OS information in a configuration file.

In the following explanation, courier text indicates **a reserved word**, while text in italics indicate **a value**, **symbol name**, or **keyword to be supplied by the user**.

5.4.1 RX Series Information

RX series information defines the name and version of the real-time OS being used.

For a configuration file, the specification of RX series information is required.

Figure 5-1 shows the format of the RX series information.

Figure 5-1. RX Series Information Format

```
rxsers  rtos_nam  rtos_ver
```

The items to be coded as RX series information are as explained below.

rtos_nam Specifies the name of the real-time OS.
RX850 is the only keyword that can be specified for *rtos_nam*.

rtos_ver Specifies the version of the real-time OS.
The keyword that can be specified as *rtos_ver* is V31x (x is any number).

5.5 SPECIFICATION FORMAT FOR SIT INFORMATION

The following shows the specification format that must be observed when describing SIT information in a configuration file.

In the following explanation, courier text indicates **a reserved word**, while text in italics indicate **a value**, **symbol name**, or **keyword to be supplied by the user**.

5.5.1 System Information

The system information defines the system stack information, clock interrupt source, and trace information.

For a configuration file, the specification of the system stack information is required.

Figure 5-2 shows the format of the system information.

Figure 5-2. System Information Format

```
intstk    intstk_siz : mem_nam
clkhdr    int_nam
trace     tracer
```

The items to be coded as system information are explained below.

intstk_siz : *mem_nam*

Specifies the size of the system stack (in bytes), and the type of system memory to be allocated to the system stack.

A value of between `0x1` and `0xfffffffc`, aligned with a four-byte boundary, can be specified for *intstk_siz*. For *mem_nam*, either of keywords `pool0` or `pool1` can be specified.

`pool0`: Allocates the system stack to system memory pool 0 (`.pool0` section).

`pool1`: Allocates the system stack to system memory pool 1 (`.pool1` section).

int_nam

Specifies the clock interrupt source used as the system clock.

The value or keyword that can be specified for *int_nam* varies with the C compiler package being used.

For CA850:

The interrupt source name defined in the device file or keyword `no_use` can be specified

For CCV850:

The value calculated from (exception code of interrupt request number - `0x80`)/`0x10` or keyword `no_use` can be specified.

Cautions 1. When `clkhdr` definition is omitted, "`clkhdr no_use`" is assumed.

2. Specifying `no_use` results in the timer operation function (cyclic wake-up, delayed wake-up, timeout, and so forth) provided by RX850 not being used.

tracer Specifies the type of the tool that uses the trace information.
If the keyword that can be specified as *tracer*, and the definition of trace are omitted, the operation to be performed differs depending on the cross tool being used.

For CA850:
As the keyword, only az can be specified.

az: Uses the trace function supplied by the AZ850.

If the definition of trace is omitted, it is assumed that "trace az" is described.

For CCV850:
The keyword that can be specified is az or multi.

az: Uses the trace function supplied by the AZ850.
multi: Uses the task debug function supplied by MULTI.

If the definition of trace is omitted, it is assumed that "trace multi" is described.

5.5.2 System Maximum Value Information

The system maximum value information defines a value for the task priority range.
Figure 5-3 shows the format of the system maximum value information.

Figure 5-3. Format of System Maximum Value Information

```
maxpri  pri_lvl
```

The items to be coded as system maximum value information are explained below.

pri_lvl Specifies the priority range for the task.
A value of between `0x1` and `0x1f` or keyword `auto` can be specified for *pri_lvl*.

Cautions 1. When `maxpri` definition is omitted, "`maxpri auto`" is assumed.
2. When `auto` is specified, the configurater references the initial priority specified in the task information, explained in Section 5.5.4, and outputs the relevant priority range.

5.5.3 Task Execution Right Group Information

Define items such as "name of task execution right group," "stack information for task execution right group," and "wait state information for task execution right group" as task execution right group information for each task execution right group.

However, the number of items that can be defined as task execution right group information is limited to between 0 and 127.

Figure 5-4 shows the format of the task execution right group information.

Figure 5-4. Format of Task Execution Right Group Information

```
tskgrp  tskgrp_id  stk_siz : mem_nam  wait
```

The items to be coded as task execution right group information are explained below.

tskgrp_id

Specifies the name of the task execution right group.
Only symbol name can be specified for *tskgrp_id*.

stk_siz : *mem_nam*

Specifies the size of the task execution right group stack (in bytes), and the type of system memory to be allocated to the task execution right group stack.
A value of between `0x1` and `0xfffffffc`, aligned with a four-byte boundary, can be specified for *stk_siz*. For *mem_nam*, either of keywords `pool0` or `pool1` can be specified.

`pool0`: Allocates the task execution right group stack to system memory pool 0 (`.pool0` section).
`pool1`: Allocates the task execution right group stack to system memory pool 1 (`.pool1` section).

wait

Specifies whether a task execution right wait state is allowed to occur.
The keyword that can be specified for *wait* is `no_wait`.

`no_wait`: When the task is started, the task execution right group wait state will not arise.

Caution When the specification of this item is omitted, the task execution right group wait state may arise when the task is started.

5.5.4 Task Information

Define items such as "task name," "start address of task," "stack information for task," "initial priority of task," "initial state of task," "activation code," and "interrupt state" as task information for each task.

For a configuration file, the specification of at least one item of task information is required.

However, the number of items that can be defined as task information is limited to between 1 and 127.

Figure 5-5 shows the task information format.

Figure 5-5. Task Information Format

```
tsk  tsk_id  sta_adr  tskgrp_id | stk_siz : mem_nam  pri  sts  sta_code  intr
```

The items to be coded as task information are explained below.

tsk_id

Specifies a task name.
Only a symbol name can be specified for *tsk_id*.

Caution The configurater outputs the relationship between *tsk_id* and task ID number to the system information header file in the following format:

`#define` *tsk_id* task-ID-number

sta_adr

Specifies the start address of the specified task.
A value of between `0x0` and `0xfffffe`, aligned with a 2-byte boundary, can be specified for *sta_adr*. Alternatively, a symbol name can be specified.

Caution When specifying a symbol name for *sta_adr*, specify a label name which is used for coding in assembly language.

tskgrp_id
stk_siz:mem_nam

Specifies the stack size (in bytes) to be used by a task, and the type of the system memory to be allocated to that stack.
Only a task execution right group name specified in **Section 5.5.3** can be specified for *tskgrp_id*.
A value of between `0x1` and `0xfffffffc`, aligned with a 4-byte boundary, can be specified for *stk_siz*. The keywords that can be specified for *mem_nam* are `pool0` and `pool1`.

`pool0`: The task stack is allocated to system memory pool 0 (`.pool0` section).
`pool1`: The task stack is allocated to system memory pool 1 (`.pool1` section).

pri

Specifies the initial priority of the task.
A value of between `0x1` and `0x1f` can be specified for *pri*.

sts

Specifies the initial state of a task.
The keywords that can be specified for *sts* are `TTS_DMT` and `TTS_RDY`.

`TTS_DMT`: The system enters the `dormant` state upon being activated.
`TTS_RDY`: The system enters the `ready` state upon being activated.

sta_code

Specifies the task activation code.
A value of between `0x0` and `0xffffffff`, or a symbol name, can be specified for *sta_code*.

Caution ***sta_code*** **is valid only when `TTS_RDY` is specified for *sts*. It is invalid when `TTS_DMT` is specified for *sts*.**

intr

Specifies the interrupt state.
The keywords that can be specified for *intr* are `ei` and `di`.

`ei`: All interrupts are enabled at task activation.
`di`: All interrupts are disabled at task activation.

Caution **Omitting this item results in all interrupts being enabled when the task is activated.**

5.5.5 Semaphore Information

Define items such as "semaphore name" and "initial resource count for semaphore" as the semaphore information for each semaphore.

However, the number of items that can be defined as semaphore information is limited to between 0 and 127.

Figure 5-6 shows the semaphore information format.

Figure 5-6. Semaphore Information Format

```
sem  sem_id  init_cnt
```

The items to be coded as semaphore information are explained below.

sem_id

Specifies a semaphore name.
Only a symbol name can be specified for *sem_id*.

Caution The configurater outputs the relationship between *sem_id* and semaphore ID number to the system information header file in the following format:

`#define` *sem_id* semaphore-ID-number

init_cnt

Specifies an initial resource count for semaphore.
A value of between `0x0` and `0x7f` can be specified for *init_cnt*.

5.5.6 Event Flag Information

Define items such as "event flag name" as event flag information for each event flag.

However, the number of items that can be defined as event flag information is limited to between 0 and 127.

Figure 5-7 shows the event flag information format.

Figure 5-7. Event Flag Information Format

```
flg flg_id
```

The items to be coded as event flag information are explained below.

flg_id Specifies an event flag name.
Only a symbol name can be specified for *flg_id*.

Caution The configurater outputs the relationship between *flg_id* and event flag ID number to the system information header file in the following format:

`#define` *flg_id* event-flg-ID-number

5.5.7 One-Bit Event Flag Information

Define an item "one-bit event flag name" as one-bit event flag information for each one-bit event flag.
The number of items that can be defined as one-bit event flag information is limited to between 0 and 127.
Figure 5-8 shows the format of the one-bit event flag information.

Figure 5-8. Format of One-Bit Event Flag Information

```
flg1  flg1_id
```

The item to be coded as one-bit event flag information is explained below.

flg1_id Specifies a one-bit event flag name.
Only a symbol name can be specified for *flg1_id*.

Caution The configurater outputs the relationship between *flg1_id* and ID number to the system information header file in the following format:

`#define` *flg1_id* one-bit-event-flag-ID-number

5.5.8 Mailbox Information

Define items such as "mailbox name" and "message queuing method" as mail box information for each mailbox.
However, the number of items that can be defined as mailbox information is limited to between 0 and 127.
Figure 5-9 shows the mailbox information format.

Figure 5-9. Mailbox Information Format

```
mbx  mbx_id  mwai_opt
```

The items to be coded as mailbox information are explained below.

mbx_id

Specifies a mailbox name.
Only a symbol name can be specified for *mbx_id*.

Caution The configurater outputs the relationship between *mbx_id* and mailbox ID number to the system information header file in the following format.

`#define` *mbx_id* mailbox-ID-number

mwai_opt

Specifies the message queuing method.
The keywords that can be specified for *mwai_opt* are `TA_MFIFO` and `TA_MPRI`.

`TA_MFIFO` : Messages are queued in the same order as that in which they are transmitted.
`TA_MPRI` : Messages are queued according to their priority.

5.5.9 Indirectly Activated Interrupt Handler Information

Define items such as "interrupt source" and "start address of indirectly activated interrupt handler" as indirectly activated interrupt handler information for each indirectly activated interrupt handler.

The number of items that can be defined as indirectly activated interrupt handler information is limited to one for each interrupt source.

Figure 5-10 shows the format of the indirectly activated interrupt handler information.

Figure 5-10. Format of Indirectly Activated Interrupt Handler Information

```
inthdr int_nam hdr_adr
```

The items to be coded as indirectly activated interrupt handler information are explained below.

int_nam

Specifies an interrupt source.
The value or keyword that can be specified for *int_nam* varies with the C compiler package being used.

For CA850:
Interrupt source name specified with a device file
For CCV850:
Value calculated using "(exception code - `0x80`)/`0x10`"

hdr_adr

Specifies the start address of the indirectly activated interrupt handler.
A value of between `0x0` and `0xfffffe`, aligned with a 2-byte boundary, can be specified in *hdr_adr*. Alternatively, a symbol name can be specified.

Caution When specifying a symbol name for *hdr_adr*, specify a label name which is used for coding in assembly language.

5.5.10 Fixed-Size Memory Pool Information

Define items such as "memory pool name," "memory block information," and "total number of memory blocks" as fixed-size memory pool information for each fixed-size memory pool.

However, the number of items that can be defined as fixed-size memory pool information is limited to between 0 and 127.

Figure 5-11 shows the fixed-size memory pool information format.

Figure 5-11. Fixed-Size Memory Pool Information Format

```
mpf  mpf_id  blk_siz : mem_nam  blk_cnt
```

The items to be coded as fixed-size memory pool information are explained below.

mpf_id

Specifies a fixed-size memory pool name.
Only a symbol name can be specified for *mpf_id*.

Caution The configurater outputs the relationship between *mpf_id* and memory pool ID number to the system information header file in the following format:

`#define` *mpf_id* fixed-size-memory-pool-ID-number

blk_siz:mem_nam

Specifies the size of a memory block (basic block size in bytes), and the type of the system memory to be allocated to that fixed-size memory pool.
A value of between `0x4` and `0xfffffffc`, aligned with a 4-byte boundary, can be specified for *blk_siz*. The keywords that can be specified for *mem_nam* are `pool0` and `pool1`.

`pool0`: The fixed-size memory pool is allocated to system memory pool 0 (`.pool0` section).
`pool1`: The fixed-size memory pool is allocated to system memory pool 1 (`.pool1` section).

blk_cnt

Specifies the total number of memory blocks.
A value greater than or equal to `0x1` can be specified for *blk_cnt*.

5.5.11 Variable-Size Memory Pool Information

Define items such as "memory pool name," and "memory block information" as variable-size memory pool information for each variable-size memory pool.

However, the number of items that can be defined as variable-size memory pool information is limited to between 0 and 127.

Figure 5-12 shows the variable-size memory pool information format.

Figure 5-12. Variable-Size Memory Pool Information Format

`mpl` *mpl_id* *mpl_siz* : *mem_nam*

The items to be coded as variable-size memory pool information are explained below.

mpl_id

Specifies a variable-size memory pool name.
Only a symbol name can be specified for *mpl_id*.

Caution The configurater outputs the relationship between *mpl_id* and memory pool ID number to the system information header file in the following format:

`#define` *mpl_id* variable-size-memory-pool-ID-number

mpl_siz : *mem_nam*

Specifies the size of a variable-size memory pool (units: bytes), and the type of the system memory to be allocated to that variable-size memory pool.
A value of between `0x8` and `0xfffffffc`, aligned with a 4-byte boundary, can be specified for *mpl_siz*. The keywords that can be specified for *mem_nam* are `pool0` and `pool1`.

`pool0`: The variable-size memory pool is allocated to system memory pool 0 (`.pool0` section).
`pool1`: The variable-size memory pool is allocated to system memory pool 1 (`.pool1` section).

Caution The CF850 outputs an error message and stops processing if a value other than a four-byte boundary value is specified as *mpl_siz*.

5.5.12 Cyclic Handler Information

Define items such as "cyclic handler name," "start address of cyclic handler," "initial activity state of cyclic handler," and "activation interval of cyclic handler" as cyclic handler information for each cyclic handler.

However, the number of items that can be defined for each cyclic handler is limited to between 0 and 127.

Figure 5-13 shows the cyclic handler information format.

Figure 5-13. Cyclic Handler Information Format

```
cyc cyc_no hdr_adr act intvl
```

The items to be coded as cyclic handler information are explained below.

cyc_no

Specifies a cyclic handler name.
Only a symbol name can be specified for *cyc_no*.

Caution The configurater outputs the relationship between *cyc_no* and cyclic handler ID number to the system information header file in the following format:

`#define` *cyc_no* cyclic-handler-specification-number

hdr_adr

Specifies the start address of the specified cyclic handler.
A value of between `0x0` and `0xfffffe`, aligned with a 2-byte boundary, can be specified for *hdr_adr*. Alternatively, a symbol name can be specified.

Caution When specifying a symbol name for *hdr_adr*, specify a label name which is used for coding in assembly language.

act

Specifies the initial activity state of the cyclic handler.
Only `TCY_ON`, `TCY_OFF`, and `TCY_ULNK` can be specified as keywords for *act*.

`TCY_ON` : At system activation, the activity state is set to ON.
`TCY_OFF` : At system activation, the activity state is set to OFF.
`TCY_ULNK` : The cyclic handler is dequeued from the timer queue.

intvl

Specifies the activation interval for the cyclic handler (units: basic clock cycles).
A value of between `0x1` and `0xffffffff` can be specified for *intvl*.

5.6 CAUTIONS

In a configuration file, describe the configuration information (real-time OS information and SIT information) in the following order.

<1> Declaration of the start of the real-time OS information description
<2> Real-time OS information description
<3> Declaration of the start of the SIT information description
<4> SIT information description

Figure 5-14 illustrates how a configuration file is described.

Figure 5-14. Describing a Configuration File

```
--------------------------------------------------------------------------------
-- Declaration of start of real-time OS information description
--------------------------------------------------------------------------------
ser_def

-----------------------------------------------------
-- Real-time OS information description
-----------------------------------------------------
..............................................
..............................................
..............................................

--------------------------------------------------------------------
-- Declaration of start of SIT information description
--------------------------------------------------------------------
sit_def

----------------------------------------
-- SIT information description
----------------------------------------
..............................................
..............................................
..............................................
```

5.7 DESCRIPTION EXAMPLES

Figures 5-15 and 5-16 show examples of configuration file description when the V851 is used.

Figure 5-15 is an example of the configuration file when the CA850 is used. Figure 5-16 shows an example when the CCV850 is used.

In the examples shown in Figures 5-15 and 5-16, the following data is coded:

(1) RX series information

Real-time OS name : RX850
Version No. : V313

(2) System information

System stack information : 0x100 bytes of system memory pool 1 are allocated.
Clock interrupt source : Exception code
For CA850 : INTCM4
For CCV850 : 0x5
Trace information : For CA850 : az
For CCV850 : multi

(3) System maximum value information

Task priority range : 0x1f

(4) Task execution right group information

Task execution right group name : txcb0
Stack information : 0x100 bytes of system memory pool 0 are allocated.
Wait information : The task execution right group wait state may occur.

Task execution right group name : txcb1
Stack information : 0x200 bytes of system memory pool 1 are allocated.
Wait information : The task execution right group wait state will not occur.

(5) Task information

Task name : task1
Start address : label-name _task1
Stack information : 0x100 bytes of system memory pool 1 are allocated.
Initial priority of task : 0x1
Initial status : Ready
Activation code : 0xa
Interrupt status : All interrupts are disabled.

Task name : task2
Start address : label-name _task2
Stack information : Task execution right group txcb0 is used.
Initial priority of task : 0x2
Initial status : Ready
Activation code : 0x14
Interrupt status : All interrupts are enabled.

Task name : task3
Start address : label-name _task3
Stack information : Task execution right group txcb1 is used.
Initial priority of task : 0x3
Initial status : Dormant
Activation code : 0x1e
Interrupt status : All interrupts are enabled.

(6) Semaphore information

Semaphore name : sem1
Initial resource count : 0x0

Semaphore name : sem2
Initial resource count : 0x0

Semaphore name : sem3
Initial resource count : 0x1

Semaphore name : sem4
Initial resource count : 0x7f

(7) Event flag information

Event flag name : evf32_1

(8) One-bit event flag information

One-bit event flag name : evf1_1

(9) Mailbox information

Mailbox name : mbx1
Queuing method : FIFO

Mailbox name : mbx2
Queuing method : FIFO

Mailbox name : mbx3
Queuing method : FIFO

Mailbox name : mbx4
Queuing method : FIFO

Mailbox name : pmbx1
Queuing method : According to priority

Mailbox name : pmbx2
Queuing method : According to priority

Mailbox name : pmbx3
Queuing method : According to priority

Mailbox name : pmbx4
Queuing method : According to priority

(10) Indirectly activated interrupt handler information

Interrupt source : Exception code
For CA850 : INTP00
For CCV850 : 0xa
Start address : Label name _inthdr

(11) Fixed-size memory pool information

Memory pool name : mpf1
Memory block information : A memory block of the basic block size (0x8 bytes) is allocated from system memory pool 1.
Total number of memory blocks : 0xa

(12) Variable-size memory pool information

Memory pool name : mpl1
Memory pool size information : 0x20 bytes is allocated from system memory pool 1.

(13) Cyclic handler information

Cyclic handler name : cychdr1
Start address : label-name _cychdr1
Initial activity state : Off
Cycle activation interval : 0x1 (units: Clock interrupt cycles)

Cyclic handler name : cychdr2
Start address : label-name _cychdr2
Initial activity state : On
Cycle activation interval : 0x2 (units: Clock interrupt cycles)

Cyclic handler name : cychdr3
Start address : label-name _cychdr3
Initial activity state : The cyclic handler is dequeued from the timer queue.
Cycle activation interval : 0x3 (units: Clock interrupt cycles)

Figure 5-15. Example of Configuration File (for CA850) (1/2)

```
--------------------------------------------------------------------------------
-- Declaration of start of real-time OS information description
--------------------------------------------------------------------------------
ser_def

------------------------------------------------------
-- Real-time OS information description
------------------------------------------------------
-- RX series information
rxsers RX850 V313

--------------------------------------------------------------------
-- Declaration of start of SIT information description
--------------------------------------------------------------------
sit_def

----------------------------------------
-- SIT information description
----------------------------------------
-- System information
intstk 0x100:pool1
clkhdr INTCM4
trace  az

-- System maximum value information
maxpri 0x1f

-- Task execution right group information
tskgrp txcb0  0x100:pool0
tskgrp txcb1  0x200:pool1     no_wait

-- Task information
tsk    task1  _task1  0x100:pool1    0x1   TTS_RDY 0xa   di
tsk    task2  _task2  txcb0          0x2   TTS_RDY 0x14  ei
tsk    task3  _task3  txcb1          0x3   TTS_DMT 0x1e
```

Figure 5-15. Example of Configuration File (for CA850) (2/2)

```
-- Semaphore information
sem   sem1      0x0
sem   sem2      0x0
sem   sem3      0x1
sem   sem4      0x7f

-- Event flag information
flg   exv32_1

-- 1-bit event flag information
flg1  exv1_1

-- Mailbox information
mbx   mbx1      TA_MFIFO
mbx   mbx2      TA_MFIFO
mbx   mbx3      TA_MFIFO
mbx   mbx4      TA_MFIFO
mbx   pmbx1     TA_MPRI
mbx   pmbx2     TA_MPRI
mbx   pmbx3     TA_MPRI
mbx   pmbx4     TA_MPRI

-- Indirectly activated interrupt handler information
inthdr INTP00   _inthdr

-- Fixed-size memory pool information
mpf   mpf1      0x8:pool1   0xa

-- Variable-size memory pool information
mpl   mpl1      0x20:pool1

-- Cyclic handler information
cyc   cychdr1 _cychdr1     TCY_OFF    0x1
cyc   cychdr2 _cychdr2     TCY_ON     0x2
cyc   cychdr3 _cychdr3     TCY_ULNK   0x3
```

Figure 5-16. Example of Configuration File (for CCV850) (1/2)

```
-------------------------------------------------------------------------------
-- Declaration of start of real-time OS information description
-------------------------------------------------------------------------------
ser_def

-----------------------------------------------------
-- Real-time OS information description
-----------------------------------------------------
-- RX series information
rxsers RX850 V313

--------------------------------------------------------------------
-- Declaration of start of SIT information description
--------------------------------------------------------------------
sit_def

----------------------------------------
-- SIT information description
----------------------------------------
-- System information
intstk 0x100:pool1
clkhdr 0x5
trace  multi

-- System maximum value information
maxpri 0x1f

-- Task execution right group information
tskgrp txcb0 0x100:pool0
tskgrp txcb1 0x200:pool1     no_wait

-- Task information
tsk    task1 _task1  0x100:pool1    0x1   TTS_RDY 0xa    di
tsk    task2 _task2  txcb0          0x2   TTS_RDY 0x14   ei
tsk    task3 _task3  txcb1          0x3   TTS_DMT 0x1e
```

Figure 5-16. Example of Configuration File (for CCV850) (2/2)

```
-- Semaphore information
sem     sem1     0x0
sem     sem2     0x0
sem     sem3     0x1
sem     sem4     0x7f

-- Event flag information
flg     exv32_1

-- 1-bit event flag information
flg1    exv1_1

-- Mailbox information
mbx     mbx1     TA_MFIFO
mbx     mbx2     TA_MFIFO
mbx     mbx3     TA_MFIFO
mbx     mbx4     TA_MFIFO
mbx     pmbx1    TA_MPRI
mbx     pmbx2    TA_MPRI
mbx     pmbx3    TA_MPRI
mbx     pmbx4    TA_MPRI

-- Indirectly activated interrupt handler information
inthdr 0xa      _inthdr

-- Fixed-size memory pool information
mpf     mpf1     0x8:pool1     0xa

-- Variable-size memory pool information
mpl     mpl1     0x20:pool1

-- Cyclic handler information
cyc     cychdr1 _cychdr1      TCY_OFF     0x1
cyc     cychdr2 _cychdr2      TCY_ON      0x2
cyc     cychdr3 _cychdr3      TCY_ULNK    0x3
```

[MEMO]

★

CHAPTER 6 OPERATING CONFIGURATER (CF850)

This chapter explains how to create an information file (system information table and system information header file) from the configuration file, using the configurater.

6.1 OUTLINE

To create an information file (system information table and system information header file) from the configuration file, activate the configurater from the command line. Start the configurater under MS-DOS™ when the Windows-based RX850 is used.

The activation of the configurater is explained below.

In the examples below, "C>" (this should be read as "%" for a UNIX-based OS) indicates the shell prompt, and "↵" indicates pressing of the return key. The options enclosed in "[ ]" may be omitted.

[For CA850]

```
C>  cf850 -cpu name [-devpath = path] [-i sit_file] [-ni] [-d h_file]
      [-nd] [-V] [-help] cf_file ↵
```

[For CCV850]

```
C>  cf850 [-i sit_file] [-ni] [-d h_file] [-nd] [-V] [-help] cf_file ↵
```

6.2 ACTIVATION OPTIONS

The configurater activation options are explained below:

`-cpu` *name* Specifies a target device.
Default : This activation option cannot be omitted.
Note : This activation option can be specified only for CA850.

`-devpath=`*path* Retrieves the device file from the *path* directory.
Default : The device file is retrieved from the current directory (.\).
Note : This activation option can be specified only for CA850.

`-i` *sit_file* Specifies the system information table name to be output.
Default : The system appends an "`i`" to the end of the configuration file name, specified with *cf_file*, changes the extension to `.tbl`, and outputs the file as the system information table.
Note : When both this activation option and the `-ni` option are specified at the same time, only that which was input last is effective.

`-ni` Disables output of the system information table.
Default : The system appends an "`i`" to the end of the configuration file name, specified with *cf_file*, changes the extension to `.tbl`, and outputs the file as the system information table.
Note : When both this activation option and the `-i` *sit_file* option are specified at the same time, only that which was input last is effective.

`-d` *h_file* Specifies the system information header file name to be output.
Default : The system changes the extension of the configuration file name, specified with *cf_file*, to `.h`, and outputs the file as the system information header file.
Note : When both this activation option and the `-nd` option are specified at the same time, only that which was input last is effective.

`-nd` Disables output of the system information header file.
Default : The system changes the extension of the configuration file name, specified with *cf_file*, to `.h`, and outputs the file as the system information header file.
Note : When both this activation option and the `-d` *h_file* option are specified at the same time, only that which was input last is effective.

`-V` Outputs version information for the configurater to the standard output.
Default : Version information for the configurater is not output.
Note : Specifying this activation option nullifies all other activation options.

`-help` Outputs the usage of the activation options for the configurater to the standard output.
Default : The usage of the activation options for the configurater is not output.
Note : Specifying this activation option nullifies all other activation options.

cf_file Specifies the configuration file name that input to the configurater.
Default : This activation option cannot be omitted.

6.3 COMMAND INPUT EXAMPLES

6.3.1 Command Input for Configurater for CA850

Examples of command input for the configurater for CA850 are given below. In this example, the V851 is used as the target.

- `cf850 -cpu 3000 -devpath=..\dev -i sitfile.tbl -d hfile.h cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `sitfile.tbl` and system information header file `hfile.h`.

- `cf850 -cpu 3000 -devpath=..\dev -i sitfile.tbl cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `sitfile.tbl` and system information header file `cffile.h`.

- `cf850 -cpu 3000 -devpath=..\dev -d hfile.h cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffile.tbl` and system information header file `hfile.h`.

- `cf850 -cpu 3000 -devpath=..\dev cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffilei.tbl` and system information header file `cffile.h`.

- `cf850 -cpu 3000 -nd cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffilei.tbl`.
  The system information header file is not output.

- `cf850 -cpu 3000 -ni cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information header file `cffile.h`.
  The system information table is not output.

- `cf850 -V`
  This command outputs version information for the configurater to the standard output.

- `cf850 -help`
  This command outputs the usage of the activation options for the configurater to the standard output.

6.3.2 Command Input for Configurater for CCV850

Examples of command input for the configurater for CCV850 are given below. In this example, the V851 is used as the target.

- `cf850 -i sitfile.tbl -d hfile.h cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `sitfile.tbl` and system information header file `hfile.h`.

- `cf850 -i sitfile.tbl cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `sitfile.tbl` and system information header file `cffile.h`.

- `cf850 -d hfile.h cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffilei.tbl` and system information header file `hfile.h`.

- `cf850 cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffilei.tbl` and system information header file `cffile.h`.

- `cf850 -nd cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information table `cffilei.tbl`.
  The system information header file is not output.

- `cf850 -ni cffile.cf`
  This command reads configuration file `cffile.cf`, then outputs system information header file `cffile.h`.
  The system information table is not output.

- `cf850 -V`
  This command outputs version information for the configurater to the standard output.

- `cf850 -help`
  This command outputs the usage of activation options for the configurater to the standard output.

6.4 MESSAGES

Upon the detection of description errors such as "incorrect definition in the configuration file" during processing, the configurater generates messages and outputs them to the standard output.

Messages are classified into three levels: fatal errors, non-fatal errors, and warning errors. The configurater adds an alphabetic character representing the error level to the beginning of the message to be output.

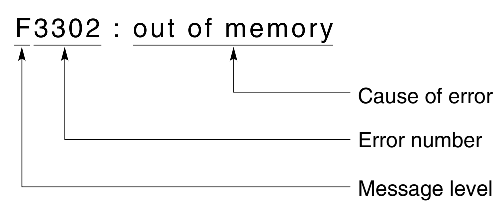
F: Fatal error
If a fatal error occurs, the configurater outputs a message then stops configuration processing.
Example Insufficient memory area.

E: Non-fatal error
If a non-fatal error occurs, the configurater outputs a message then stops configuration processing.
Example Duplicated definition.

W: Warning error
If a warning error occurs, the configurater outputs a message and continues configuration processing.
Example The description of a parameter has been omitted.

Figure 6-1 shows the message format.

Figure 6-1. Message Format



6.4.1 Fatal Errors

This section explains the messages output when fatal errors occur.

In the following messages, those items in italics (e.g., *file_name*) are determined if the associated fatal error occurs:

F2001: Usage:cf850 -cpu <name>[devpath=<path>][-i <sit_file>][-ni][-d<head_file>][-nd][-V][-help]<cf_file>

Activation option specification is invalid for the configurater for CA850.

F2001: Usage:cf850[-i <sit_file>][-ni][-d <head_file>][-nd][-V][-help]<cf_file>

Activation option specification is invalid for the configurater for CCV850.

F2002: Can't allocate memory.

Insufficient memory

F2003: Can't open file "*file_name*".

File *file_name* cannot be opened.

F2004: Out of memory.

Insufficient memory

F2005: Can't open device file.

The device file cannot be opened.

F2006: Can't ready device file.

The device file cannot be read.

F2007: Unknown device file format

A device file that is not supported has been specified.

6.4.2 Non-Fatal Errors

This section explains the messages output if non-fatal errors occur.

In the following messages, those items in italics (e.g., *name*) are determined if the associated non-fatal error occurs:

`E2001: ser_def not defined.`

The declaration of the start of real-time OS information, `ser_def`, is not found on the first line.

`E2002: Illegal ser_def.`

The declaration of the start of real-time OS information, `ser_def`, is encountered in other than the correct location.

`E2003: ser_def already defined.`

The declaration of the start of real-time OS information, `ser_def`, is defined twice.

`E2004: sit_def not defined.`

The declaration of the start of SIT information, `sit_def`, is not found.

`E2005: Illegal sit_def.`

The declaration of the start of SIT information, `sit_def`, is encountered in other than the correct location.

`E2006: sit_def already defined.`

The declaration of the start of SIT information, `sit_def`, is defined twice.

`E2007: Out of sit_def division.`

Data which is part of SIT information is defined before the declaration of the start of SIT information, `sit_def`.

`E2012: rxsers not defined.`

RX series information `rxsers` is not defined.

`E2013: Illegal rxsers.`

RX series information `rxsers` is defined in other than the correct location.

`E2014: rxsers already defined.`

RX series information `rxsers` is defined twice.

`E2101: Integer overflow.`

A numeric value exceeds the 32-bit data range.

`E2102: Syntax error.`

The configuration coding format is illegal.

`E2103: Word too long.`

The symbol name exceeds the maximum number of characters.

`E2104: Address out of range.`

A specified address falls outside the allowable range.

`E2105: Address must be aligned by 2.`

Specify an address on a 2-byte boundary.

`E2106: Stack size out of range.`

A stack size exceeding the allowable range is specified.

`E2107: Symbol "`*name*`" already defined.`

The symbol *name* is defined twice.

`E2201: System stack size not defined.`

System stack information `intstk` is not defined.

`E2202: System stack size already defined.`

System stack information `intstk` is defined twice.

`E2203: Interrupt level "`*name*`" already defined.`

Two or more indirectly activated interrupt handlers are registered for the same interrupt source "*name*".

`E2204: Interrupt level of system clock already defined.`

Clock interrupt source `clkhdr` is defined twice.

`E2206: Priority level already defined.`

System maximum value information `maxpri` is defined twice.

`E2207: Priority level out of range.`

System maximum value information `maxpri` falls outside the allowable range.

`E2208: Taskgroup "`*name*`" already defined.`

The task execution right group *name* is defined twice.

`E2209: Task not defined.`

Task information `tsk` is not defined.

`E2210: Too many tasks.`

There are 128 or more task information `tsk` definitions.

`E2211: Task "`*name*`" already defined.`

The task *name* is defined twice.

`E2212: Taskgroup "`*name*`" in task not defined.`

The task execution right group *name* is not defined.

`E2213: Task priority out of range.`

The initial priority *pri* is not within the range of `0x1` to `0x1f`.

`E2214: Too many eventflags.`

There are 128 or more event flag information `flg` definitions.

`E2215: Eventflag "`*name*`" already defined.`

The event flag *name* is defined twice.

`E2216: Too many 1bit eventflags.`

There are 128 or more one-bit event flag information `flg1` definitions.

`E2217: 1bit eventflag "`*name*`" already defined.`

The one-bit event flag *name* is defined twice.

`E2218: Too many Semaphores.`

There are 128 or more semaphore information `sem` definitions.

`E2219: Semaphore "`*name*`" already defined.`

The semaphore *name* is defined twice.

`E2220: Semaphore resource count out of range.`

The initial resource count *init_cnt* is not within the range of `0x0` to `0x7f`.

`E2221: Too many mailboxes.`

There are 128 or more mailbox information `mbx` definitions.

`E2222: Mailbox "`*name*`" already defined.`

The mailbox *name* is defined twice.

`E2223: Too many fixed-size memorypools.`

There are 128 or more fixed-size memory pool information `mpf` definitions.

`E2224: Memorypool "`*name*`" already defined.`

The fixed-size memory pool *name* is defined twice.

`E2225: Memory block size of block count is 0.`

Basic block size *blk_siz* or total number of memory blocks *blk_cnt* is defined with `0x0`.

`E2226: Memory block size must be aligned by 4.`

Basic block size *blk_siz* must be on a four-byte boundary.

`E2227: Memory total size exceeds 4Gbyte.`

The total size of system memory exceeds 4G bytes.

`E2228: Too many cyclic handlers.`

There are 128 or more cyclic handler information `cyc` definitions.

`E2229: Cyclic handler "`*name*`" already defined.`

The cyclic handler *name* is defined twice.

`E2230: Interval time out of range.`

The activation interval for the cyclic handler *intvl* falls outside the range of `0x1` to `0xffffffff`.

`E2231: Too many variable-size memorypools.`

There are 128 or more variable-size memory pool information `mpl` definitions.

`E2232: Variable-size memorypool` *name* `already defined.`

The variable-size memory pool *name* is defined twice.

`E2233: Too small variable-size memorypool size.`

There are 128 or more variable-size memory pool information `mpl` definitions.

`E2234: Variable-size memorypool size must be aligned by 4.`

The size of variable-size memorypool must be on a four-byte boundary.

`E2235: Illegal trace.`

Trace information `trace` is defined in other than the correct location.

`E2236: Trace already defined`

Trace information `trace` is defined twice.

6.4.3 Warning Errors

This section explains the messages output if warning errors occur.

In the following messages, those items in italics (e.g., *name*) are determined if the associated warning error occurs:

`W2201: Use "`*pri1*`" priority level, but priority "`*pri2*`" task is defined, define "`*pri2*`" priority level are used.`

Although *pri* is defined as the task priority range, *pri* is specified for the initial task priority.

The configurater assumes *pri2* to be specified as the task priority range, and continues processing.

`W2202: Taskgroup "`*name*`" is not used in task, ignored.`

Although *name* is defined for a task execution right group, it is not specified in the task information.

The configurater assumes the task execution right group *name* to be undefined, and continues processing.

`W2203: Stack size must be aligned by 4, round up stack size.`

The defined stack size is not on a four-byte boundary.

The configurater rounds up the size so that the stack is aligned with the nearest four-byte boundary, and continues processing.

[MEMO]

APPENDIX A INDEX

[U]

[V]

[W]

★

APPENDIX B REVISION HISTORY

A history of revisions up to this edition is shown below. "Applied to:" indicates the chapters to which the revision was applied.

Edition	Contents	Applied to:
2nd edition	Modification of description of target CPU of execution environment	**Chapter 1**
	Modification of description of hardware environment and software environment of development environment	
	Modification of description of install/uninstall	**Chapter 2**
	Modification of description of system construction procedure	**Chapter 3**
	Modification of description of memory management and memory capacity to be used	**Chapter 4**
	Modification of description of outline of configuration file	**Chapter 5**
	Modification of description of outline of configurater (CF850)	**Chapter 6**

[MEMO]

Facsimile Message

From:

Name

Company

Tel. FAX

Address

Although NEC has taken all possible steps to ensure that the documentation supplied to our customers is complete, bug free and up-to-date, we readily accept that errors may occur. Despite all the care and precautions we've taken, you may encounter problems in the documentation. Please complete this form whenever you'd like to report errors or suggest improvements to us.

Thank you for your kind support.

North America
NEC Electronics Inc.
Corporate Communications Dept.
Fax: +1-800-729-9288
+1-408-588-6130

Europe
NEC Electronics (Europe) GmbH
Technical Documentation Dept.
Fax: +49-211-6503-274

South America
NEC do Brasil S.A.
Fax: +55-11-6462-6829

Hong Kong, Philippines, Oceania
NEC Electronics Hong Kong Ltd.
Fax: +852-2886-9022/9044

Korea
NEC Electronics Hong Kong Ltd.
Seoul Branch
Fax: +82-2-528-4411

Taiwan
NEC Electronics Taiwan Ltd.
Fax: +886-2-2719-5951

Asian Nations except Philippines
NEC Electronics Singapore Pte. Ltd.
Fax: +65-250-3583

Japan
NEC Semiconductor Technical Hotline
Fax: +81- 44-435-9608

I would like to report the following error/make the following suggestion:

Document title: ____________________

Document number: ____________________ Page number: ____________________

If possible, please fax the referenced page or drawing.

Document Rating	Excellent	Good	Acceptable	Poor
Clarity	❏	❏	❏	❏
Technical Accuracy	❏	❏	❏	❏
Organization	❏	❏	❏	❏

CS 01.2