To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas Electronics does not assume any liability for infringement of patents, copyrights, or other intellectual property rights of third parties by or arising from the use of Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: “Standard”, “High Quality”, and “Specific”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as “Specific” without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as “Specific” or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is “Standard” unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   “Standard”: Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; and industrial robots.

   “High Quality”: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; safety equipment; and medical equipment not specifically designed for life support.

   “Specific”: Aircraft; aerospace equipment; submersible repeaters; nuclear reactor control systems; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) “Renesas Electronics product(s)” means any product developed or manufactured by or for Renesas Electronics.

User's Manual

RX830 (μITRON Ver. 3.0)

Real-Time Operating System

Installation

Target Device
V830 Family™

Document No. U13151EJ2V1UM00 (2nd edition)
Date Published April 1999 J CP(K)

[MEMO]

The export of this product from Japan is regulated by the Japanese government. To export this product may be prohibited without governmental license, the need for which must be judged by the customer. The export or re-export of this product from a country other than Japan may also be prohibited without a license from that country. Please call an NEC sales representative.

- **The information in this document is subject to change without notice. Before using this document, please confirm that this is the latest version.**
- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Corporation. NEC Corporation assumes no responsibility for any errors which may appear in this document.
- NEC Corporation does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from use of a device described herein or any other liability arising from use of such device. No license, either express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Corporation or of others.
- Descriptions of circuits, software, and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software, and information in the design of the customer's equipment shall be done under the full responsibility of the customer. NEC Corporation assumes no responsibility for any losses incurred by the customer or third parties arising from the use of these circuits, software, and information.

M7A 98.8

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: 01-30-67 58 00
Fax: 01-30-67 58 99

NEC Electronics (France) S.A.
Spain Office
Madrid, Spain
Tel: 91-504-2787
Fax: 91-504-2860

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.
United Square, Singapore 1130
Tel: 65-253-8311
Fax: 65-250-3583

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

NEC do Brasil S.A.
Electron Devices Division
Rodovia Presidente Dutra, Km 214
07210-902-Guarulhos-SP Brasil
Tel: 55-11-6465-6810
Fax: 55-11-6465-6829

J99.1

PREFACE

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.

Operating systems are broadly classified into two types: program-development OSs and control OSs. Program-development OSs are to be found in those environments in which standard OSs (e.g., Windows 95 and UNIX OS) predominate because the hardware configuration to be used for development can be limited to some extent (e.g., personal computers).

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.

To satisfy these demands, NEC has developed and released not only the V800 Series™ V830 Family™ of microprocessors but also the RX830 operating system, which allows users to fully exploit the functions of these microprocessors and support systematic software creation.

RX830 is a control OS for real-time, multitasking processing; it has been developed to increase the application range of high-performance, multi-function microprocessors and further improve their generality.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

[MEMO]

CHAPTER 1 OVERVIEW

1.1 OUTLINE

RX830 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.
RX830 is a high-speed, compact OS capable of being stored in and run from the ROM of a target system.

1.2 FEATURES

RX830 has the following features:

(1) Conformity with μITRON 3.0 specification
RX830 is designed to conform with the μITRON 3.0 specification, that defines a typical built-in control OS architecture. RX830 implements μITRON 3.0 functions of up to level E.
The μITRON 3.0 specification applies to a built-in, real-time control OS.

(2) High generality
In addition to the 66 system calls specified by the μITRON 3.0 specification, RX830 provides seven original system calls specific to RX830, so that it can run on more generalized application systems.
RX830 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 during creating the system.

(3) Realization of real-time processing and multitasking
RX830 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
RX830 supports functions for disabling and resuming dispatching (task scheduling).
This allows the user to disable and resume a dispatching process from the processing program level.

(5) Compact design
RX830 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 V830 Family™ microprocessors, combined with V830 Series™ original instructions, enables high-speed processing.

(7) Utilization of internal RAM

By making use of the internal RAM (instruction memory and data memory) built into the V830 Family™, rapid instruction execution and data access are possible.

(8) Utility support

RX830 supports the following two utilities to aid in application system construction:

- Configurator CF830
- High-level language interface library

(9) C compiler

RX830 corresponds to the following V830 Family™ C compilers:

- CA830 (NEC Corporation)
- CCV830 (Green Hills Software Inc.)

1.3 EXECUTION ENVIRONMENT

This section explains the processing environment required by RX830.

- Processor
  V830 Family™

- Peripheral controller
  RX830 provides sample source files for that portion that is dependent on the hardware configuration of the execution environment (system initialization: boot processing and hardware initialization section).
  Therefore, simply rewriting the system initialization for each target system eliminates the need to use a specific peripheral controller.

- Required memory space
  The memory space required for RX830 installation and operation is as follows:

Space for nucleus text part:	approx. 4K to 11K bytes
Space for nucleus data:	approx. 1K to 2K bytes

  The maximum memory space above is required if "maximum priority assignment range" and "use of all functions (system calls) provided by RX830" are specified as the configuration settings. By restricting the priority assignment range and the RX830 functions, the required amount of memory space can be reduced.

1.4 DEVELOPMENT ENVIRONMENT

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

- Hardware environment
  - Host machine
    - * PC-9800 Series — Windows 95
    - * IBM-PC/AT-compatible machine — Windows 95
    - * SPARCstation™ — SunOS™ Rel 4.1.x
    - * SPARCstation™ — Solaris™ Rel 2.5.x
  - In-circuit emulators
    - * IE-705100-MC-EM1 — For V830
    - * IE-70000-MC-NW — For V831
  - Network modules
    - * IE-70000-MC-SV2
- Software environment
  - C compiler packages
    - * CA830 — NEC Corporation
    - * CCV830 — Green Hills Software Inc.
  - Debuggers
    - * ID830 — NEC Corporation
    - * MULTI — Green Hills Software Inc.
  - Task debugger
    - * RD830 — NEC Corporation
  - System performance analyzer
    - * AZ830 — NEC Corporation

CHAPTER 2 INSTALLATION

This chapter explains how to install the files, stored on the media supplied with RX830 into the user development environment.

2.1 OUTLINE

Two types of RX830 distribution medium are provided according to the user development environment (host machine); one for the Windows-based environment and the other for the UNIX-base environment.

Table 2-1 shows the correspondence between the user development environment and the RX830 distribution media.

Table 2-1. Correspondence between User Development Environment and RX830 Distribution Media

User development environment (host machine)	RX830 distribution media
Windows-base • PC-9800 series • IBM-PC/AT-compatible machine	Floppy disk (FD)
UNIX-base • SPARCstation™	Floppy disk (FD) or Cartridge magnetic tape (CGMT)

2.2 INSTALLATION PROCEDURE

The procedure for installing the files stored in the RX830 distribution medium varies depending on the user development environment.

The subsequent sections describe the different installation procedures for Windows and UNIX, on either of which the user development environment is assumed to be based.

2.3 WINDOWS-BASED INSTALLATION

For a Windows-based user development environment (PC-9800 Series or IBM-PC/AT-compatible machine), install RX830 by means of the following procedure:

In the input examples, "A>" indicates the shell prompt, "Δ" indicates one or more blanks, and "<cr>" indicates the return key.

(1) Starting Windows 95

Turn on the personal computer and peripheral devices to start Windows 95.

(2) Displaying the MS-DOS prompt

Display the MS-DOS prompt.

Note The procedure for installing RX830 is performed at the MS-DOS prompt.

(3) Changing the current directory

Using the cd command, change the current directory to the installation directory.

In the following example, root directory A:\ on drive A is specified as the directory to which the files will be installed.

Note Installation of RX830 requires approximately 1M byte of free disk space.

```
A> cdΔa:\  <cr>
```

(4) Loading the distribution media (floppy disk)

Load the floppy disk into the floppy disk drive.

Here, assume that the floppy disk is loaded into drive C.

(5) Installing a file group

Using the xcopy command, install the appropriate file group from the floppy disk into the user development environment.

The floppy disk contains the following two versions of RX830:

- RX830 for use with CA830
- RX830 for use with CCV830

That is, when using CA830 as the C compiler package, install the RX830 version for CA830. When using CCV830, install the RX830 version for CCV830.

[To copy the RX830 version for CA 830]

```
A> xcopyΔc:\nectoolsΔ.Δ /sΔ/eΔ/v  <cr>
```

[To copy the RX830 version for CCV830]

```
A> xcopy∆c:\ghstools∆.∆/s∆/e∆/v  <cr>
```

(6) Checking the file group

Using the `dir` command, check that the file group stored on the distribution media has been successfully copied to the user development environment.

See **Section 2.5** for details of each directory.

[For the RX830 version for CA830]

```
A> dir∆/s∆a:\nectools  <cr>
```

[For the RX830 version for CCV830]

```
A> dir∆/s∆a:\ghstools  <cr>
```

(7) Setting the command search path

Set the command search path for the RX830-supported utility (configurator CF830).

Here, examples of setting the command search path in the `path` variable in the `autoexec.bat` environment setting file are given.

[For the RX830 version for CCV830]

```
path = %path%;a:\ghstools\bin
```

Note Because the configurator of the version for CA830 starts from VSH, the operation for "command search path setting" is not necessary if C compiler package CA830 for NEC's V830 Family™ is used.

2.4 UNIX-BASED INSTALLATION

If the user development environment is based on UNIX (SPARCstation™), install RX830 according to the following procedure.

In the input examples, "%" indicates the shell prompt, "Δ" indicates one or more blanks, and "<cr>" indicates the return key.

(1) Logging into the development environment

Log into the development environment.

```
%
```

(2) Changing the current directory

Change the current directory to the installation directory by executing the `cd` command.

In the following example, `/usr/local` is specified as the directory into which the files will be installed.

Note Installation of RX830 requires approximately 1M byte of free disk space. Make sure that the permissions (`read`, `write`, and `execute`) for the installation directory have been granted.

```
% cdΔ/usr/local  <cr>
```

(3) Mounting the distribution medium

Mount the distribution medium (floppy disk or magnetic cartridge tape) in the floppy disk drive or the magnetic cartridge tape unit as appropriate.

(4) Installing a group of files

Install the group of files stored on the distribution medium into the user development environment by executing the `tar` command.

The distribution medium contains the following two versions of RX830:

- RX830 version for use with CA830
- RX830 version for use with CCV830

To use the CA830 C-compiler package, install the RX830 version for CA830. To use the CCV830 C-compiler package, install the RX830 version for CCV830.

In the following example, `/dev/rst8` is specified as the special file name.

[For the RX830 version for CA830]

```
% tarΔ-xvofΔ/dev/rst8/Δnectools  <cr>
```

[For the RX830 version for CCV830]

```
% tarΔ-xvofΔ/dev/rst8/Δghstools  <cr>
```

(5) Checking the file group

Make sure that the group of files stored on the distribution medium has been successfully installed in the user development environment by executing the `ls` command.

See **Section 2.5** for details of each directory.

[For the RX830 version for CA830]

```
% lsΔ-CFRΔ/user/local/nectools  <cr>
```

[For the RX830 version for CCV830]

```
% lsΔ-CFRΔ/user/local/ghstools  <cr>
```

(6) Setting the command search path

Set the command search path for the utility tool (configurator CF830) provided by RX830.

The following are examples of how to specify the setting in the `path` variable in the `.cshrc` environment setting file.

[For the RX830 version for CA830]

```
set path = ($path/usr/local/nectools/bin)
```

[For the RX830 version for CCV830]

```
set path = ($path/usr/local/ghstools/bin)
```

2.5 DIRECTORY STRUCTURE

The RX830 distribution formats are classified into a source file distribution format and object file distribution format. The directory structures of these formats differ.

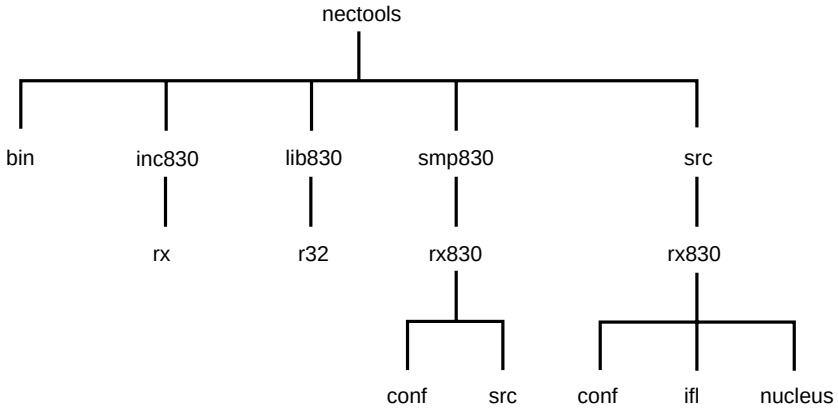
The following sections describe the directory structures of the source file distribution format and those of the object file distribution format.

2.6 SOURCE FILE DISTRIBUTION FORMAT

2.6.1 Source File Distribution Format for the RX830 Version for CA830

Figure 2-1 shows the source file directory structure (RX830 version for CA830).

Figure 2-1. Source File Directory Structure (RX830 Version for CA830)



(1) `nectools`
Directory containing the RX830 version for CA830

(2) `nectools\bin`
Directory containing the utility (configurator CF830) supported by RX830

(3) `nectools\inc830`
Directory containing header files

`stdrx.h`: Standard header file (for C source file)
This header file includes all the header files required by RX830.
`stdrx.inc`: Standard header file (for assembly language source file)
This header file, when included, includes all the header files required for RX830 operation.

(4) `nectools\inc830\rx`
Directory containing RX830 header files

(5) `nectools\lib830`
Directory containing object and library files

(6) `nectools\lib830\r32`
Directory containing the object files (32-register mode) and library files (32-register mode) for RX830 as below:

`rxcore.o`: Common nucleus part
`librx.a`: Nucleus library
`libch.a`: System call interface library (with check function)
`libnc.a`: System call interface library (without check function)

(7) nectools\smp830

Directory containing sample programs

(8) nectools\smp830\rx830

Directory containing RX830 sample load programs

(9) nectools\smp830\rx830\conf

Directory containing the command file for generating load modules from sample programs.

Executing the command file contained in this directory creates the following load modules in current directory nectools\smp830\rx830\conf:

sample.out: Load module (not including ROM information)
sample.rom: Load module (including ROM information)
sample.hex: Load module (HEX format)

(10) nectools\smp830\rx830\src

Directory containing the source file for sample programs

(11) nectools\src

Directory containing source files

(12) nectools\src\rx830

Directory containing RX830 source files

(13) nectools\src\rx830\conf

Directory containing the command file to generate the object files (32-register mode) and library files (32-register mode) for RX830.

By executing the command file in this directory, the object files (32-register mode) and the library files (32-register mode) mentioned below are generated in the nectools\lib830\r32 directory where these files are stored.

rxcore.o: Common nucleus part
librx.a: Nucleus library
libch.a: System call interface library (with check function)
libnc.a: System call interface library (without check function)

(14) nectools\src\rx830\ifl

Directory containing the source files of the system call interface libraries (one with the check function and another without the check function)

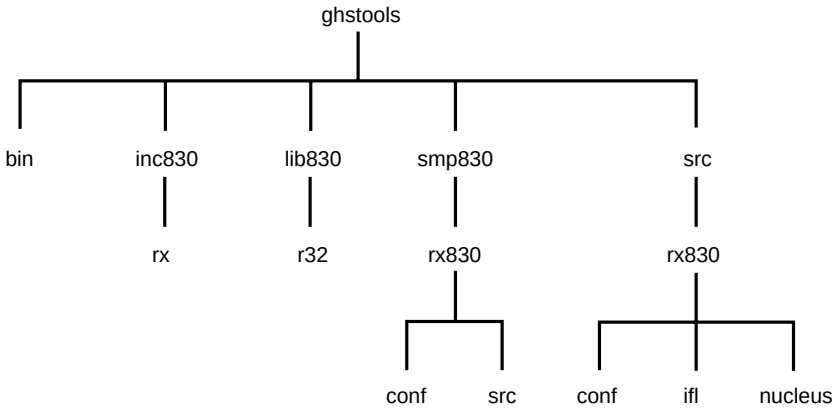
(15) nectools\src\rx830\nucleus

Directory containing the source file for the nucleus common part and nucleus libraries

2.6.2 Source File Distribution Format for the RX830 Version for CCV830

Figure 2-2 shows the source file directory structure (RX830 version for CCV830).

Figure 2-2. Source File Directory Structure (RX830 Version for CCV830)



(1) `ghstools`
Directory containing the version of RX830 corresponding to CCV830

(2) `ghstools\bin`
Directory containing the utility (configurator CF830) supported by RX830

(3) `ghstools\inc830`
Directory containing header files

`stdrx.h`: Standard header file
This header file includes all header files required by RX830.

(4) `ghstools\inc830\rx`
Directory containing RX830 header files

(5) `ghstools\lib830`
Directory containing nucleus libraries

(6) `ghstools\lib830\r32`
Directory containing the object files (32-register mode) and the library files (32-register mode) for RX830 as below:

`rxcore.o`: Common nucleus part
`librx.a`: Nucleus library
`libch.a`: System call interface library (with check function)
`libnc.a`: System call interface library (without check function)

(7) `ghstools\smp830`
Directory containing sample programs

(8) ghstools\smp830\rx830

Directory containing RX830 sample programs

(9) ghstools\smp830\rx830\conf

Directory containing the command file for generating load modules from sample programs.

Executing the command file contained in this directory creates the following load modules in current directory ghstools\smp830\rx830\conf:

sample.rom: Load module (including ROM information)
sample.hex: Load module (HEX format)

(10) ghstools\smp830\rx830\src

Directory containing the source file for sample programs

(11) ghstools\src

Directory containing source files

(12) ghstools\src\rx830

Directory containing RX830 source files

(13) ghstools\src\rx830\conf

Directory containing the command file to generate the object files (32-register mode) and the library files (32-register mode) for RX830.

By executing the command file in this directory, the object files (32-register mode) and library files (32-register mode) mentioned below are generated in the ghstools\lib830\r32 directory where these files are stored.

rxcore.o: Common nucleus part
librx.a: Nucleus library
libch.a: System call interface library (with check function)
libnc.a: System call interface library (without check function)

(14) ghstools\src\rx830\ifl

Directory containing the source files of the system call interface libraries (one with the check function and another without the check function)

(15) ghstools\src\rx830\nucleus

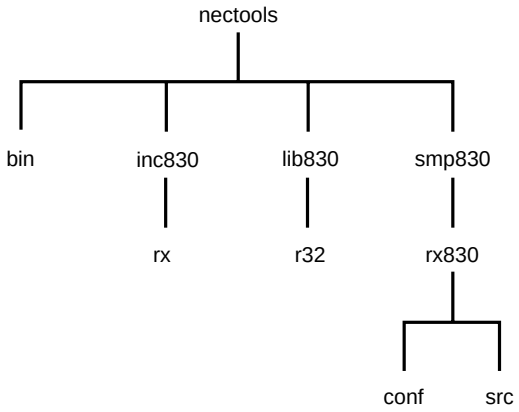
Directory containing the source file for the nucleus common part and nucleus libraries

2.7 OBJECT FILE DISTRIBUTION FORMAT

2.7.1 Object File Distribution Format for the RX830 Version for CA830

Figure 2-3 shows the object file directory structure (RX830 version for CA830).

Figure 2-3. Object File Directory Structure (RX830 Version for CA830)



(1) `nectools`
Directory containing the RX830 version for CA830

(2) `nectools\bin`
Directory containing the utility (configurator CF830) supported by RX830

(3) `nectools\inc830`
Directory containing header files

`stdrx.h`: Standard header file (for C source file)
This header file includes all the header files required by RX830.
`stdrx.inc`: Standard header file (for assembly language source file)
This header file, when included, includes all the header files required for the RX830 operation.

(4) `nectools\inc830\rx`
Directory containing RX830 header files

(5) `nectools\lib830`
Directory containing object and library files

(6) `nectools\lib830\r32`
Directory containing the object files (32-register mode) and the library files (32-register mode) for RX830 as below:

`rxcore.o`: Common nucleus part
`librx.a`: Nucleus library
`libch.a`: System call interface library (with check function)
`libnc.a`: System call interface library (without check function)

(7) `nectools\smp830`
Directory containing sample programs

(8) `nectools\smp830\rx830`
Directory containing RX830 sample programs

(9) `nectools\smp830\rx830\conf`
Directory containing the command file for generating load modules from sample programs.
Executing the command file contained in this directory creates the following load modules in current directory `nectools\smp830\rx830\conf`:

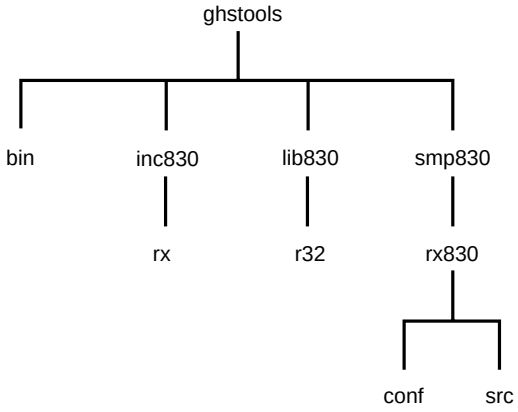
`sample.out`: Load module (not including ROM information)
`sample.rom`: Load module (including ROM information)
`sample.hex`: Load module (HEX format)

(10) `nectools\smp830\rx830\src`
Directory containing the source file for sample programs

2.7.2 Object File Distribution Format for the RX830 Version for CCV830

Figure 2-4 shows the object file directory structure (RX830 version for CCV830).

Figure 2-4. Object File Directory Structure (RX830 Version for CCV830)



(1) `ghstools`
Directory containing the version of RX830 corresponding to CCV830

(2) `ghstools\bin`
Directory containing the utility (configurator CF830) supported by RX830

(3) `ghstools\inc830`
Directory containing header files

`stdrx.h`: Standard header file
This header file includes all header files required by RX830.

(4) `ghstools\inc830\rx`

Directory containing RX830 header files

(5) `ghstools\lib830`

Directory containing object and library files

(6) `ghstools\lib830\r32`

Directory containing the object files (32-register mode) and the library files (32-register mode) for RX830 as below:

`rxcore.o`:	Common nucleus part
`librx.a`:	Nucleus library
`libch.a`:	System call interface library (with check function)
`libnc.a`:	System call interface library (without check function)

(7) `ghstools\smp830`

Directory containing sample programs

(8) `ghstools\smp830\rx830`

Directory containing RX830 sample programs

(9) `ghstools\smp830\rx830\conf`

Directory containing the command file for generating load modules from sample programs.

Executing the command file contained in this directory creates the following load modules in current directory `ghstools\smp830\rx830\conf`:

`sample.rom`:	Load module (including ROM information)
`sample.hex`:	Load module (HEX format)

(10) `ghstools\smp830\rx830\src`

Directory containing the source file for sample programs

CHAPTER 3 SYSTEM CONSTRUCTION

This chapter explains how to construct an application system using RX830.

3.1 OUTLINE

System construction involves incorporating created load modules into a target system, using the file group copied from the RX830 distribution media to the user development environment (host machine).

The system construction procedure is outlined below.

(1) Creating a CF definition file

(2) Generating the user own coding part

- Boot processing
- Hardware initialization section
- Software initialization section
- Interrupt/exception entry
- Clock interrupt post-processing
- I/O port operation

(3) Creating processing programs

- Task
- Directly activated interrupt handler
- Indirectly activated interrupt handler
- Cyclic handler
- Extended SVC handler
- Extended SVC handler interface library

(4) Creating an initialization data save area

(5) Creating a link directive file

(6) Creating a load module

- Not including ROM information
- Including ROM information
- HEX format

(7) Incorporating the load module into the target system

Note When compiler package CCV830 for C cross V800™, produced by Green Hills Software Inc., is used, an initialization data save area need not be created.

Figures 3-1 and 3-2 show the system construction procedures.

Figure 3-1 illustrates the system construction procedure when V830 Family™ C compiler package CA830 (NEC Corporation) is used. Figure 3-2 illustrates the system construction procedure when compiler package CCV830 for C cross V800™ (Green Hills Software Inc.) is used.

Figure 3-1. System Construction When Using CA830

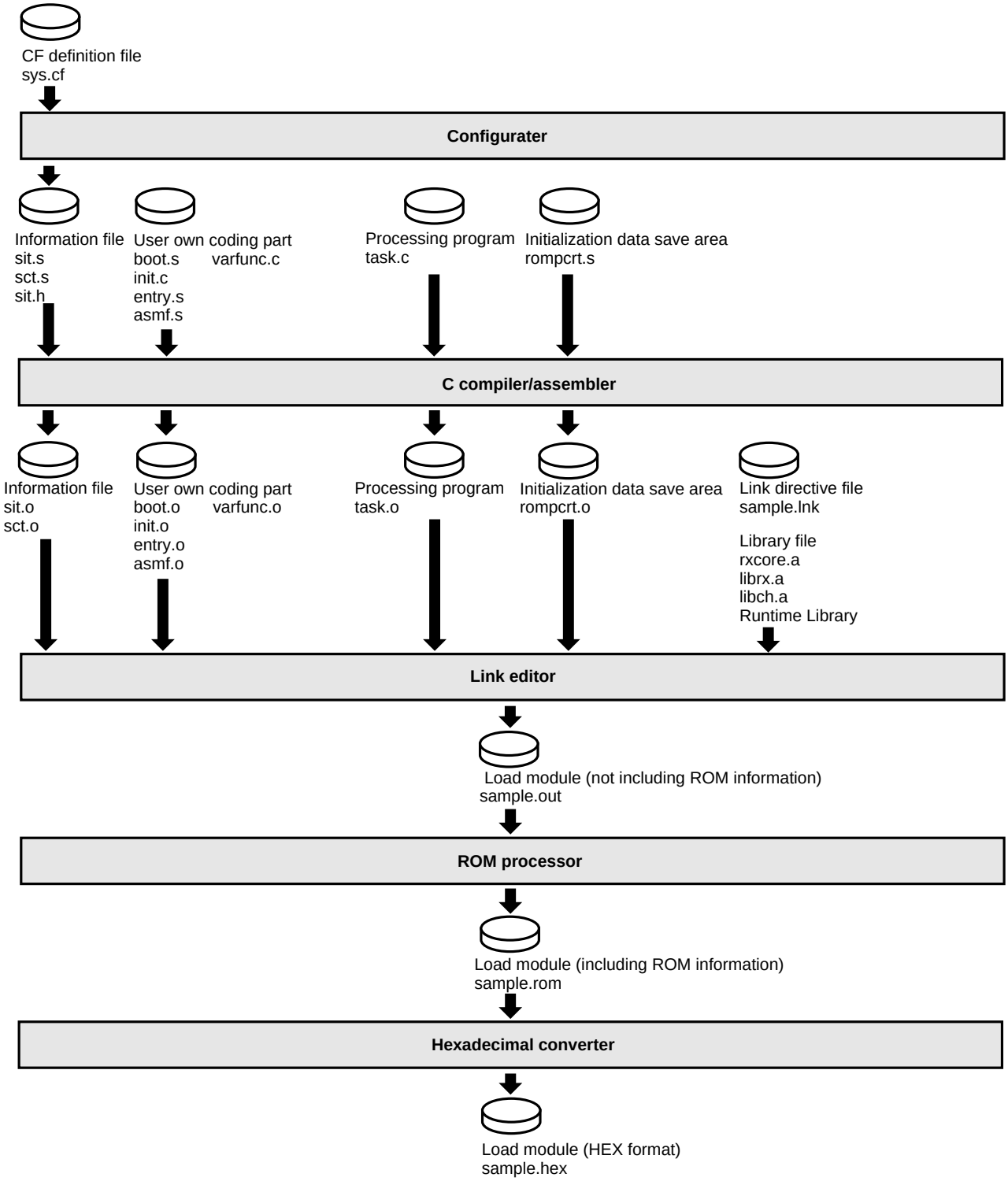
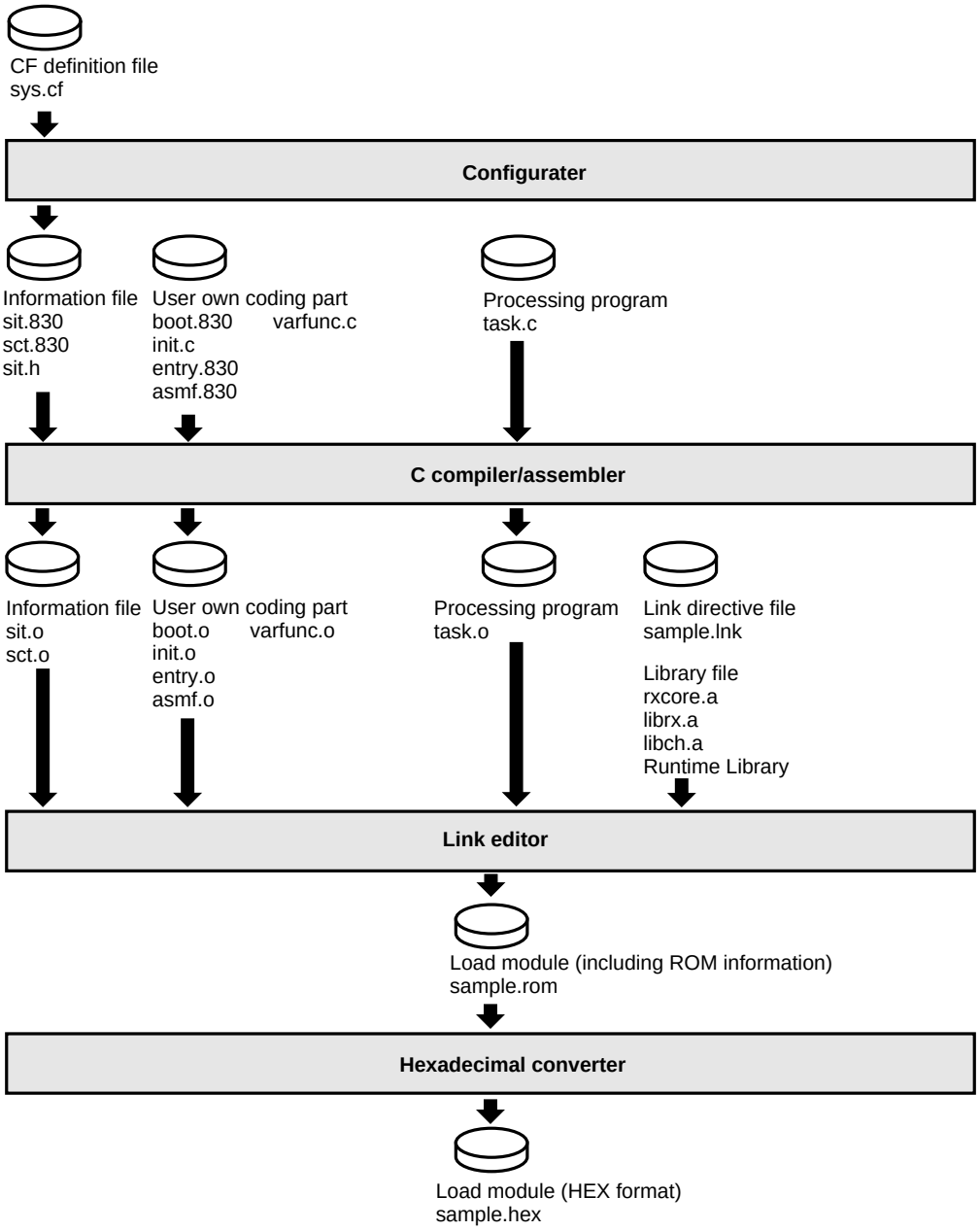


Figure 3-2. System Construction When Using CCV830



3.2 CREATING A CF DEFINITION

Create a file (CF definition file) containing data to be distributed to RX830.
The following data is required to create the CF definition file.

- Real-time OS information
  - RX series information

- SIT information
  - System information
  - System maximum value information
  - Memory information
  - Task information
  - Semaphore information
  - Event flag information
  - Mailbox information
  - Interrupt handler information
  - Memory pool information
  - Cyclic handler information
  - Extended SVC handler information
  - Initialization handler information

- SCT information
  - Task management/task-associated synchronization function information
  - Task communication (semaphore) function information
  - Task communication (event flag) function information
  - Task communication (mailbox) function information
  - Interrupt management function information
  - Memory pool management function information
  - Time management function information
  - System management function information

Notes 1. The standard CF definition file is stored at the following location:

`nectools\smp830\rx830\src\sys.cf` For CA830

`ghstools\smp830\rx830\src\sys.cf` For CCV830

2. See **Chapter 8** for details of the CF definition file description format.

3.3 GENERATING THE USER OWN CODING PART

Generate the functions for processing the part that depends on the environment (of the target application system) for running RX830, set apart from the RX830 main program operation. Furthermore, generate those functions that should be provided to make the user software environment more useful and for other specific purposes.

- Boot processing
  This function is allocated to the reset entry of the V830 Family™.

- Hardware initialization section
  This function is provided to initialize the hardware in the environment (of the target application system) for running RX830.

- Software initialization section
  This function is provided to make the user software environment more useful.

- Interrupt/exception entry
  This function is provided as a routine that is dedicated to the processing of the branch to the interrupt/exception processing management, activated when an interrupt/exception occurs.

- Clock interrupt post-processing
  This function is provided as a routine that is dedicated to the post-processing of the interrupt handler (clock handler) for time management, activated when a clock interrupt occurs.

- I/O port operation
  This function is provided to implement the read/write operation at the C level of the data from a port address.

Notes 1. The coding that is provided as standard for boot processing is found in the following file (directory):
`nectools\smp830\rx830\src\boot.s` for the RX830 version for CA830
`ghstools\smp830\rx830\src\boot.830` for the RX830 version for CCV830

2. The coding that is provided as standard for the hardware initialization section, software initialization section, and clock interrupt post-processing is found in the following file (directory):
`nectools\smp830\rx830\src\init.c` for the RX830 version for CA830
`ghstools\smp830\rx830\src\init.c` for the RX830 version for CCV830

3. The coding that is provided as standard for interrupt/exception entry is found in the following file (directory):
`nectools\smp830\rx830\src\entry.s` for the RX830 version for CA830
`ghstools\smp830\rx830\src\entry.830` for the RX830 version for CCV830

4. The coding that is provided as standard for I/O port operation is found in the following file (directory):
`nectools\smp830\rx830\src\asmf.s` for the RX830 version for CA830
`ghstools\smp830\rx830\src\asmf.830` for the RX830 version for CCV830

5. See **Chapter 4** for details of the user own coding part.

3.4 CREATING PROCESSING PROGRAMS

Create the processing (processing programs) to be performed by the system.
The processing programs are classified as follows, based on their usage.

- Task
  Minimum unit in which processing programs can be executed under RX830 control

- Directly activated interrupt handler
  A directly activated interrupt handler is a routine dedicated to interrupt processing. When the maskable interrupt occurs, this handler is initiated without the intervention of RX830 and handled independently of all other tasks. Upon the maskable interrupt occurrence, thus, control is passed to the directly activated interrupt handler, stopping the processing of the task being executed, even if that task has the highest priority in the system.
  A directly activated interrupt handler is a routine dedicated to interrupt processing, initiated without the intervention of RX830. This handler thus offers an excellent response time, approximating to the maximum that can be expected from the hardware.

- Indirectly activated interrupt handler
  An indirectly activated interrupt handler is a routine dedicated to interrupt processing. When the maskable interrupt occurs, this handler is activated upon the completion of the preprocessing by RX830 (such as saving the contents of the registers or switching the stack) and handled independently of all other tasks. Upon the maskable interrupt occurrence, thus, control is passed to the indirectly activated interrupt handler, stopping the processing of the task being executed, even if that task has the highest priority in the system.
  Although the response time of an indirectly activated interrupt handler is longer than that of a directly activated interrupt handler, the interrupt pre-processing by RX830 makes the processing done by the interrupt handler simple.

- Cyclic handler
  A cyclic handler is a routine dedicated to cyclic processing. Every time the specified time elapses, this handler is activated immediately. This routine is handled independently of tasks. At the cyclic activation time, thus, control is passed to the cyclic handler, stopping the processing of the task being executed, even the task of the highest priority in the system.
  A cyclic handler incurs a smaller overhead before the start of execution, relative to any other cyclic processing programs written by the user.

- Extended SVC handler
  An extended SVC handler is a processing routine registered with RX830 to call a function coded by the user as an extended system call.
  RX830 positions the extended SVC handler as the continuation of an execution procedure from which the extended SVC handler has been called. Therefore, if an extended SVC handler is called from a task, it is possible to issue only those system calls that can be issued from the task. If an extended SVC handler is called from a non-task, it is possible to issue only those system calls that can be issued from the non-task. Pay careful attention to this point.

Notes 1. The standard processing programs (task and indirectly activated interrupt handler) are stored at the following location:

`nectools\smp830\rx830\src\task.c` For CA830

`ghstools\smp830\rx830\src\task.c` For CCV830

2. Refer to the **RX830 User's Manual, Basic** for details of these processing programs' format.

3.5 CREATING AN INITIALIZATION DATA SAVE AREA

Create the save area for the initialization data (sections such as `.data`, `.sdata`, and `.itext` sections) which is required for loading the system into ROM.

Notes 1. The standard initialization data save area is contained in the following location:

`nectools\smp830\rx830\src\rompcrt.s` For CA830

2. When compiler package CCV830 for C cross V800™, produced by Green Hills Software Inc., is used, an initialization data save area need not be created.

3.6 CREATING A LINK DIRECTIVE FILE

Create the link directive file used to fix the address assignment to be performed by the link editor.

RX830 provides the whole text divided into three sections that can be stored in different memory regions. A description of each section is provided below. This allows memory allocation in such a manner that the standard processing part, which requires a large memory space, is stored into external RAM, while the interrupt processing part and the scheduling part, which require rapid memory access, are stored into the internal instruction RAM.

- Standard processing part (`.system` section)
  This part consists of the RX830 main processing programs (for task management, synchronous communication, etc.).

  Note The memory space required for the standard processing part is larger than the capacity of the internal instruction RAM. Thus, it is impossible to allocate the standard processing part to internal RAM.

- Interrupt processing part (`.system_int` section)
  This part consists of the interrupt pre-processing program which is executed to pass control to an indirectly activated interrupt handler, and the interrupt post-processing program which is executed for the return from that handler.
  Allocation of the interrupt processing part to the internal instruction RAM reduces the response time from the occurrence of a maskable interrupt to the activation of the indirectly activated interrupt handler.

- Scheduling part (`.system_cmn` section)
  This part consists of the task-wakeup processing and task scheduling.
  Allocation of the scheduling part to the internal instruction RAM enables more rapid processing of task wake-up and task scheduling. Besides, system calls that must be scheduled can be processed more rapidly.

Notes 1. The standard link directive file is stored at the following location:

`nectools\smp830\rx830\src\sample.lnk` For CA830

`ghstools\smp830\rx830\src\sample.lnk` For CCV830

2. For details of the link directive file's format, refer to the user's manual for the C compiler package being used.

3.7 CREATING A LOAD MODULE

The processing flow for creating a load module after the necessary files described in **Sections 3.2** to **3.6** have been created is shown below.

(1) Creating an information file

Create the information file (including the system information table, the branch table, and the system information header file) by applying the configurator to the CF definition file.

(2) Creating object files

Create object files by executing the C compiler or assembler for the following source files:

- Information file
  - System information table
  - Branch table
- System initialization
  - Boot processing
  - Hardware initialization section
  - Interrupt/exception entry
  - Clock interrupt post-processing
  - I/O port operation
  - Software initialization section
- Processing programs
  - Task
  - Directly activated interrupt handler
  - Indirectly activated interrupt handler
  - Cyclic handler
  - Extended SVC handler
  - Extended SVC handler interface library
- Initialization data save area

(3) Creating a load module (not including ROM information)

Create a load module (not including ROM information) by executing the link editor for the following object files, link directive file, and library files:

- Information file
  - System information table
  - Branch table
- System initialization
  - Boot processing
  - Hardware initialization section
  - Interrupt/exception entry
  - Clock interrupt post-processing
  - I/O port operation
  - Software initialization section

- Processing programs
  - Task
  - Directly activated interrupt handler
  - Indirectly activated interrupt handler
  - Cyclic handler
  - Extended SVC handler
  - Extended SVC handler interface library

- Initialization data save area

- Link directive file

- Library files
  - Common nucleus part
  - Nucleus library
  - System call interface library
  - Runtime library

(4) Creating a load module (including ROM information)

Create a load module (including ROM information) by executing the ROM processor for the load module (not including ROM information).

(5) Creating a load module (in HEX format)

Create a load module in HEX format by executing the hexadecimal converter for the load module (including ROM information).

Note When compiler package CCV830 for C cross V800™, produced by Green Hills Software Inc., is used, an initialization data save area need not be created.

[MEMO]

CHAPTER 4 USER OWN CODING PART

This chapter describes the functions and design of the user own coding part.

4.1 OUTLINE

The user own coding part is a group of functions provided for processing that part that varies with the user operating environment (of the target application system), set apart from the RX830 main program operation, thus improving the portability of the RX830, and making the RX830 easy to customize.

As the standard provision, the user own coding part has been coded, assuming that the V830 Family™ board [DVE-V830/20], supplied by DENSAN Co., Ltd, is used as the user operating environment. If the user operating environment is not the standard configuration or if a specific hardware mode setting is changed, the user own coding part must be rewritten.

Table 4-1 lists the entities that constitute the user own coding part.

Table 4-1. Structure of the User Own Coding Part

Entity	Function name	Provided function
Boot processing	`boot`	System boot processing
Hardware initialization section	`reset`	Hardware initialization
Software initialization section	`varfunc`	Adaptive software environment setup
Interrupt/exception entry	`entry`	Processing of the branch to the interrupt/exception processing management
Clock interrupt post-processing	`clkhdr`	Post-processing of the interrupt handler for time management

I/O port operation	`inpb`
	`inph`
	`inpw`
	`outpb`
	`outph`
	`outpw`
		Byte data read
		Half-word data read
		Word data read
		Byte data write
		Half-word data write
		Word data write

Header file	—	Definition of port addresses, etc.

4.2 BOOT PROCESSING

Boot processing is the function allocated to the reset entry of the V830 Family™. Boot processing is performed as the first procedure of the system initialization processing.

As provided as standard, the boot processing (function name: `boot`) executes the following:

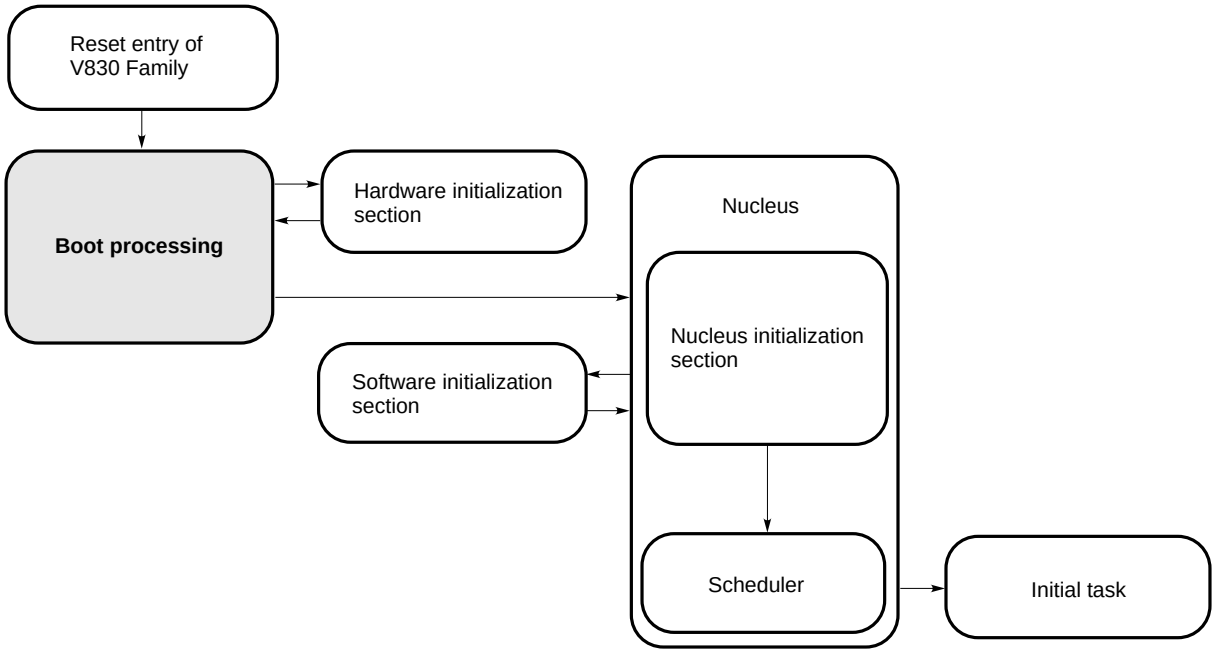
- Setting up the `gp` and `tp` registers;
- Initializing the memory areas, clearing to the state without initial values;
- Invoking the hardware initialization section; and
- Transferring control to the nucleus initialization section.

Note The coding as the standard provision for boot processing is found in the following file (directory):

`nectools\smp830\rx830\src\boot.s`	For CA830
`ghstools\smp830\rx830\src\boot.830`	For CCV830

Figure 4-1 shows the position of the boot processing.

Figure 4-1. Position of Boot Processing



4.3 HARDWARE INITIALIZATION SECTION

The hardware initialization section is provided to initialize the hardware in the operating environment (of the target application system), which is invoked from the boot processing.

As the standard provision, the hardware initialization section (function name: `reset`) executes the following:

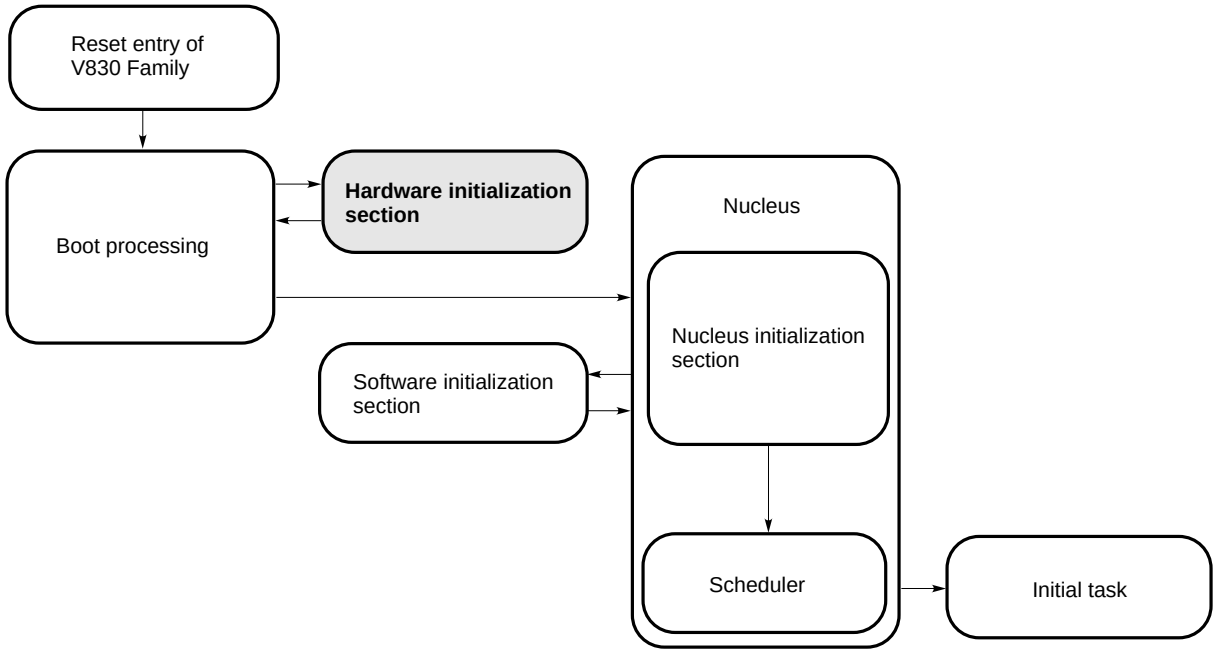
- Initializing the internal units;
  - Initializing the interrupt controller
  - Initializing the clock controller
- Initializing the peripheral controllers; and
- Returning control to the boot processing.

Note The standard hardware initialization section is stored at the following location:

`nectools\smp830\rx830\src\init.c`	For CA830
`ghstools\smp830\rx830\src\init.c`	For CCV830

Figure 4-2 shows the positioning of the hardware initialization section.

Figure 4-2. Positioning of Hardware Initialization Section



4.4 SOFTWARE INITIALIZATION SECTION

The software initialization section is provided to make the user software environment more useful, which is invoked from the nucleus initialization section.

In the standard configuration, the software initialization section (function name: `varfunc`) executes the following:

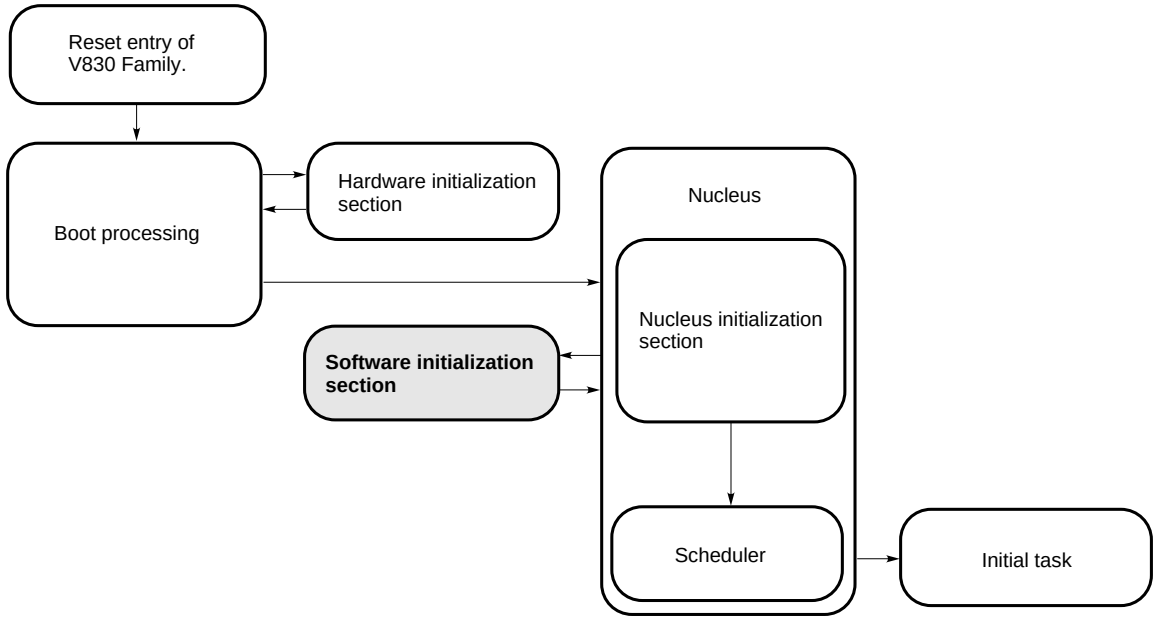
- Copying of the initialization data;
- Enabling the clock interrupt; and
- Returning control to the nucleus initialization section.

Note The coding provided as standard for the software initialization part is found in the following file (directory):

`nectools\smp830\rx830\src\init.c`	For CA830
`ghstools\smp830\rx830\src\init.c`	For CCV830

Figure 4-3 shows the position of the software initialization section.

Figure 4-3. Position of Software Initialization Section



4.5 INTERRUPT/EXCEPTION ENTRY

The interrupt/exception entry is provided as the routine dedicated to the instruction of the branch to the interrupt/exception processing management, activated when an interrupt/exception occurs. It is allocated to the handler address held by the V830 Family™.

In the standard configuration, the interrupt/exception entry (function name: `entry`) executes the following:

- Jump to the entry of the interrupt/exception processing management (in the RX830).

Note The coding provided as standard for interrupt/exception entry is found in the following file (directory):

`nectools\smp830\rx830\src\entry.s`	For CA830
`ghstools\smp830\rx830\src\entry.830`	For CCV830

4.6 CLOCK INTERRUPT POST-PROCESSING

The clock interrupt post-processing is provided as a routine that is dedicated to the instruction for branching to the interrupt handler (clock handler) for time management, activated when a clock interrupt occurs. It is allocated to the handler address (the entry for clock interrupt) held by the V830 Family™.

In the standard configuration, the clock interrupt post-processing (function name: `clkhdr`) executes the following:

- Clearing the clock interrupt latch;
- Issuing the interrupt terminating command; and
- Transferring the control to the interrupt handler (clock handler) for time management.

Note The coding provided as standard for clock interrupt post-processing is found in the following file (directory):

`nectools\smp830\rx830\src\init.c`	For CA830
`ghstools\smp830\rx830\src\init.c`	For CCV830

4.7 I/O PORT OPERATION

I/O port operation is provided to implement reading/writing, in C, of data from a port address.

In the standard configuration, the I/O port operation (function name: `inpb`, `inph`, `inpw`, `outpb`, `outph`, `outpw`) executes the following:

- `inpb`
  Reads byte data from the port address specified by parameter.

- `inph`
  Reads half-word data from the port address specified by parameter.

- `inpw`
  Reads word data from the port address specified by parameter.

- `outpb`
  Writes byte data into the port address specified by parameter.

- `outph`
  Writes half-word data into the port address specified by parameter.

- `outpw`
  Writes word data into the port address specified by parameter.

The function interface specifications are described on the subsequent pages.

Note The coding provided as standard for I/O operation is found in the following file (directory):

`nectools\smp830\rx830\src\asmf.s`	For or CA830
`ghstools\smp830\rx830\src\asmf.830`	For or CCV830

inpb

Overview

Reads byte data.

Format

```
int    inpb(int prtadr);
```

Parameter

I/O	Parameter	Description
I	`int prtadr;`	Port address from which the data is read

Explanation

This function reads byte data from the port address assigned for *prtadr*.
The read byte data is returned as a return value.

Return value

Read byte data

inph

Overview

Reads half-word data.

Format

```
int    inph(int prtadr);
```

Parameter

I/O	Parameter	Description
I	`int prtadr;`	Port address from which the data is read

Explanation

This function reads half-word data from the port address assigned for *prtadr*.
The read half-word data is returned as a return value.

Return value

Read half-word data

inpw

Overview

Reads word data.

Format

```
int    inpw(int prtadr);
```

Parameter

I/O	Parameter	Description
I	int *prtadr*;	Port address from which the data is read

Explanation

This function reads word data from the port address assigned for *prtadr*.
The read word data is returned as a return value.

Return value

Read word data

outpb

Overview

Writes byte data.

Format

```
void    outpb(int prtadr, int data);
```

Parameter

I/O	Parameter	Description
I	`int prtadr;`	Port address from which the data is written
I	`int data;`	Data to be written

Explanation

This function writes the byte data assigned for *data* into the port address assigned for *prtadr*.

Return value

None.

outph

Overview

Writes half-word data.

Format

```
void    outph(int prtadr, int data);
```

Parameter

I/O	Parameter	Description
I	int *prtadr*;	Port address from which the data is written
I	int *data*;	Data to be written

Explanation

This function writes the half-word *data* assigned for data into the port address assigned for *prtadr*.

Return value

None.

outpw

Overview

Writes word data.

Format

```
void    outpw(int prtadr, int data);
```

Parameter

I/O	Parameter	Description
I	int *prtadr*;	Port address from which the data is written
I	int *data*;	Data to be written

Explanation

This function writes the word data assigned for *data* into the port address assigned for *prtadr*.

Return value

None.

4.8 HEADER FILE

RX830 includes the header file in which constants, such as port addresses, used by the functions in the user own coding part, are macro defined.

In the standard configuration, the header file stores the following macro definitions:

- Port address of VME bus interface gate array
- Port address of board control register
- Tick timer control commands
- Interrupt control commands
- Port address of interrupt acknowledge cycle register

Note The coding provided as standard for the header file is found in the following file (directory):

`nectools\smp830\rx830\src\user.h`	For CA830
`ghstools\smp830\rx830\src\user.h`	For CCV830

[MEMO]

CHAPTER 5 INTERFACE LIBRARIES

5.1 OUTLINE

Processing programs (tasks, non-tasks) coded in C are generated in the external function format which is used to issue system calls and call extended SVC handlers. The format for issue (to the nucleus), which can be recognized by the nucleus, however, differs from the external function format.

Thus, it is necessary to convert system calls or extended SVC handlers issued in external function format into the format for issue to the nucleus. Interface libraries perform this conversion, acting as the interface between the processing program and the nucleus.

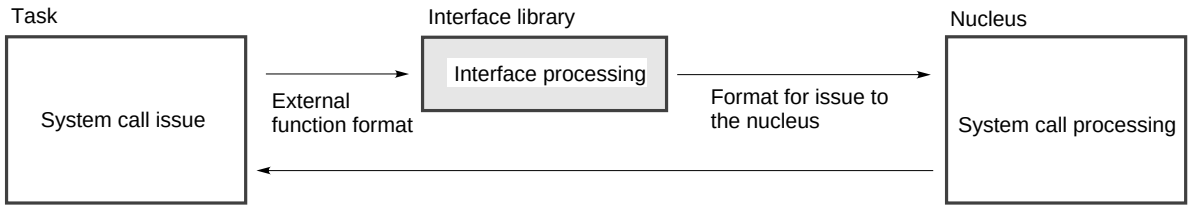
That is, the interface libraries are positioned between the processing program and the nucleus and operate to set the different types of information required for the nucleus to process the issued system call or extended SVC handler and pass the control to the nucleus.

Interface libraries provided by RX830 are available with those for C compiler CA830 for V830 Family™, supplied by NEC, and those for C cross V800™ compiler CCV830, supplied by Green Hills Software, Inc., USA.

Also, available are two types of system call interface libraries provided by RX830: one with a parameter check function to check system call parameters and another without the parameter check function. Either may be used to satisfy your system application requirements.

Figure 5-1 shows the interface library position.

Figure 5-1. Position of Interface Library



5.2 SYSTEM CALL INTERFACE LIBRARY

The main operation of the system call interface library is detailed below. The method of system call parameter hand-over, however, complies with the C compiler used.

- Saves the function code setting of the system call into the `r10` register.
- Saves the address setting for the return from the system call into the `lp` register.
- Checks the system call parameters.
- Acquires the address of the system call entry (the value of the `hp` register + address `0x100`).
- Jumps to the system call entry.

If an error is detected as the result of the system call parameter check, the interface library executes the following:

- Saves the error code associated with the detected error into the `r10` register.
- Jumps to the return address of the system call exception process.

Figure 5-2 shows an example of the coding of a system call interface library.

Figure 5-2. System Call Interface Library

```
            .text
            .globl      _syscall_name
            .align      2
_syscall_name
            -- Set the function code to be saved.
            mov         func_code, r10
            -- Parameter check.
            .....
            .....
            .....
            jz          _syscall_err
            -- Acquire the address of the system call entry.
            ld.w        0x100[hp], r12
            -- Jump to the system call entry.
            jmp         [r12]


-- Processing upon error detection
_syscall_err
            -- Set the error code to be saved.
            movea       lo(err_code), r0, r10
            -- Jump to the return address of the system call exception process.
            jmp         [lp]
```

5.3 EXTENDED SVC HANDLER INTERFACE LIBRARY

The main operation of the extended SVC handler interface library is detailed below. The method of extended SVC handler parameter hand-over, however, complies with the C compiler being used.

- Saves the function code setting of the extended SVC handler into the `r10` register.
- Saves the address setting for the return from the extended SVC handler into the `lp` register.
- Checks the extended SVC handler parameters.
- Saves the setting of the extended SVC handler parameter area size into the `r11` register.
  Example: In the case of four parameters of `int` type, `0x10` is saved into the `r11` register.
- Acquires the address of the extended SVC handler entry (the value of the `hp` register + address `0x108`).
- Jumps to the extended SVC handler entry.

If an error is detected as a result of the extended SVC handler parameter check, the interface library executes the following:

- Saves the error code associated with the detected error into the `r10` register.
- Jumps to the return address of the extended SVC handler exception process.

Figure 5-3 shows an example of coding an extended SVC handler interface library.

Figure 5-3. Extended SVC Handler Interface Library

```
            .text
            .globl      _svchdr_name
            .align      2
_svchdr_name
            -- Set the function code to be saved.
            mov         func_code, r10
            -- Parameter check.
            .....
            .....
            .....
            jz          _svchdr_err
            -- Set the parameter area size to be saved.
            mov         prm_siz, r11
            -- Acquire the address of the extended SVC handler entry.
            ld.w        0x108[hp], r12
            -- Jump to the extended SVC handler entry.
            jmp         [r12]


-- Processing upon error detection
_svchdr_err
            -- Set the error code to be saved.
            movea       lo(err_code), r0, r10
            -- Jump to the return address of the extended SVC handler exception process.
            jmp         [lp]
```

[MEMO]

CHAPTER 6 REQUIRED MEMORY SPACE ESTIMATES

This chapter provides information to help you to estimate the total memory space required for the RX830 management region (system memory region and user memory region).

6.1 OUTLINE

The RX830 management region is divided into two regions: system memory and user memory.

- System memory region
  The system memory region consists of System Memory Pool 0 and System Memory Pool 1. The management objects, the stack area for tasks, and the stack area for interrupt handlers are allocated to this region.

  Note For RX830, the allocation of management objects is limited to System Memory Pool 0.

- User memory region
  The user memory region consists of User Memory Pool 0 and User Memory Pool 1, to which memory pools are allocated.

6.2 REQUIRED MEMORY SPACE CALCULATION FORMULAS

6.2.1 Management Objects

The formulas provided in this section are used to calculate the memory space (in units of bytes) required for the management objects.

The whole management object area, which should be allocated to the system memory, is estimated as the sum of the values calculated using these formulas.

- Operating system management table
  OBT_SIZ = `align8(744 + align32 (`*`pri_lvl`*` + 4)/8) + align8(`*`pri_lvl`*` x 2 + 8)`

  Note *`pri_lvl`* corresponds to the value assigned when you define the task priority range in the system maximum value information as one of the RX830 configuration settings.

- Task management block
  TSK_SIZ = *`tsk_cnt`* `x 56`

  Note *`tsk_cnt`* corresponds to the value assigned when you define the maximum number of tasks that can be created in the system maximum value information as one of the RX830 configuration settings.

- Semaphore management block
  SEM_SIZ = *sem_cnt* x 20

  Note *sem_cnt* corresponds to the value assigned when you define the maximum number of semaphores that can be created in the system maximum value information as one of the RX830 configuration settings.

- Event flag management block
  FLG_SIZ = *flg_cnt* x 20

  Note *flg_cnt* corresponds to the value assigned when you define the maximum number of event flags that can be created in the system maximum value information as one of the RX830 configuration settings.

- Mailbox management block
  MBX_SIZ = *mbx_cnt* x 20

  Note *mbx_cnt* corresponds to the value assigned when you define the maximum number of mailboxes that can be created in the system maximum value information as one of the RX830 configuration settings.

- Memory pool management block
  MPL_SIZ = *mpl_cnt* x 24

  Note *mpl_cnt* corresponds to the value assigned when you define the maximum number of memory pools that can be created in the system maximum value information as one of the RX830 configuration settings.

- Time management block for cyclic wake-up
  CYC_SIZ = *cyc_cnt* x 40

  Note *cyc_cnt* corresponds to the value assigned when you define the maximum number of cyclic handlers that can be registered in the system maximum value information as one of the RX830 configuration settings.

- Extended SVC handler management block
  SVC_SIZ = svc_cnt x 16

  Note *svc_cnt* corresponds to the value assigned when you define the maximum number of extended SVC handlers that can be registered in the system maximum value information as one of the RX830 configuration settings.

6.2.2 Stack Area for Tasks

The formula of calculating the memory space (in units of bytes) required for the stack area per task is given below.

The entire stack area for tasks, which should be allocated to the system memory, is estimated as the sum of the stack areas for all tasks, each of which is calculated by using the following formula:

STK_SIZ = `align8(`*`stk_siz`* `+ 88)`

Note *`stk_siz`* corresponds to the value assigned when you define the stack data for a task in the stack information as one of the RX830 configuration settings, or the value assigned when you define the stack size as the task generation data at *`cre_tsk`* system call issue. *`stk_siz`* includes the size of the stack for interrupts (76 bytes).

6.2.3 Stack Area for Interrupt Handlers

The following formula should be used to calculate the memory space (in units of bytes) required for the stack area for interrupt handlers:

INTSTK_SIZ = *`intstk_siz`*

Note *`intstk_siz`* corresponds to the value assigned when you define the stack data for the interrupt handler in the system information as one of the RX830 configuration settings.

6.2.4 Memory Pools

The formula of calculating the memory space (in units of bytes) required for each memory pool is given below.

The entire memory pool area, which should be allocated to the user memory, is estimated as the sum of the memory pool areas, each of which is calculated by using the following formula:

MPL_SIZ = *`mpl_siz`*

Note *`mpl_siz`* corresponds to the value assigned when you define the memory pool information as one of the RX830 configuration settings, or the value assigned when you define the memory pool size in the memory pool information at *`cre_mpl`* system call issue.

6.3 EXAMPLE ESTIMATE OF REQUIRED MEMORY SPACE

This section describes how to estimate the memory space required for the RX830 management region (system memory region and user memory region).

[Prerequisites]

- System information
  Stack data for interrupt handlers: Memory area of `0x100` bytes must be reserved in System Memory Pool 0.

- System maximum value information

Task priority range:	`0xf`
Maximum number of tasks that can be created:	`0x2`
Maximum number of semaphores that can be created:	`0x1`
Maximum number of event flags that can be created:	`0x2`
Maximum number of mailboxes that can be created:	`0x3`
Maximum number of memory pools that can be created:	`0x2`
Maximum number of cyclic handlers that can be registered:	`0x1`
Maximum number of extended SVC handlers that can be registered:	`0x1`

- Task information
  Stack data for task: A memory area of `0x100` bytes must be reserved in System Memory Pool 0.
  Stack data for task: A memory area of `0x100` bytes must be reserved in System Memory Pool 0.

- Memory pool information
  Memory pool information: A memory area of `0x1000` bytes must be reserved in User Memory Pool 0.
  Memory pool information: A memory area of `0x2000` bytes must be reserved in User Memory Pool 0.

Note It is assumed that the `cre_tsk` and `cre_mpl` system calls have not been issued in the user processing program.

[Estimating method]

- Management objects

```
OBJ_SIZ = {align8(744 + align32(0xf + 4)/8) + align8(0xf x 2 + 8)}
          + {0x2 x 56} + {0x1 x 20} + {0x2 x 20} + {0x3 x 20}
          + {0x2 x 24} + {0x1 x 40} + {0x1 x 20}
        = 1,288
```

- Stack area for tasks

```
STK_SIZ = align8(0x100 + 88) + align8(0x100 + 88)
        = 688
```

- Stack area for interrupt handlers
  INTSTK_SIZ = 256

- Memory pools

  MPL_SIZ = 0x1000 + 0x2000
  = 12,288

From the above calculation, the following RX830 management region is estimated to be required:

System memory region
  System Memory Pool 0: 2,072 bytes
  System Memory Pool 1: 0 bytes

User memory region
  User Memory Pool 0: 12,288 bytes
  User Memory Pool 1: 0 bytes

[MEMO]

CHAPTER 7 CONFIGURATOR CF830

This chapter explains the role and operating environment of configurator CF830.

7.1 OUTLINE

When RX830 is used to construct a system, information files (system information table, branch table, and system information header file) containing data to be distributed to RX830 become necessary.

Because the data arrangement in these information files conforms to a prescribed format they can, in principle, be described using any standard editor. Because, however, the files are relatively difficult to describe and subsequently understand, considerable skill is required to describe them.

To overcome this difficulty, therefore, RX830 provides a utility tool that converts a file (CF definition file) created in a proprietary coding format with improved coding ease, and decodes into an information file.

This utility tool is called the configurator. The configurator reads a CF definition file created in a proprietary coding format as the input and outputs an information file, such as a system information table, branch table, and system information header file.

The configurator outputs the following information files:

- System information table
  This information file contains the required data (such as basic clock cycle and the task priority range) for RX830 operation.

- Branch table
  This information file contains the data of the optional setting of whether each function (system call) is used by the system.
  By restricting the functions used by the system, the amount of memory required for RX830 can be reduced.

- System information header file
  This information file defines the correspondence between object names described in the CF definition file (such as task names and semaphore names) and their ID numbers.

7.2 OPERATING ENVIRONMENT

Table 7-1 defines the configurator operating environment.

Table 7-1. Configurator Operating Environment

User development environment (host machine)	Operating system
PC-9800 Series	Windows 95 (MS-DOS prompt)
IBM-PC/AT-compatible machine	Windows 95 (MS-DOS prompt)
SPARCstation™	SunOS™ Rel 4.1.x
SPARCstation™	Solaris™ Rel 2.5.x

CHAPTER 8 DESCRIBING A CF DEFINITION FILE

This chapter describes how to code a CF definition file that is required to create information files (system information table, branch table, and system information header file) using configurator CF830.

8.1 DECLARATION

The following shows how to make the necessary declaration when describing a CF definition file.

(1) Character codes

A CF definition file is described using ASCII code.

The system does not differentiate between upper-case and lower-case characters.

Character strings (values, symbol names, keywords) are delimited using either a space or a tab. Statements are delimited by an carriage return.

Note Japanese-language text, in either EUC or shift-JIS code, can be described only within a comment.

(2) Values

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

(3) Symbol names

Symbol names consisting of up to 256 alphanumeric characters can be specified.

Note, however, that the first character of a symbol name must be an alphabetic character or an underline.

(4) Comments

Within a CF definition file, all text between "—" and the end of the line is handled as a comment.

(5) Continuation lines

Within a CF definition 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 character strings shown below are reserved key words for use with the configurator.

The use of these character strings for any other purpose is prohibited.

clktim	cyc	defstk	flg	flgsvc
ini	inthdr	intstk	intsvc	maxcyc
maxflg	maxmbx	maxmpl	maxpri	maxsem
maxsvc	maxtsk	mbx	mbxsvc	mem
mpl	mplsvc	no_use	prtflg	prtmbx
prtmpl	prtsem	prttsk	RX830	rxsers
sct_def	sem	semsvc	ser_def	sit_def
SPOL0	SPOL1	svc	syssvc	TA_ASM
TA_DISINT	TA_ENAINT	TA_HLNG	TA_MFIFO	TA_MPRI
TA_TFIFO	TA_TPRI	TA_WMUL	TA_WSGL	TCY_OFF
TCY_ON	timsvc	tsk	tsksvc	TTS_DMT
TTS_RDY	UPOL0	UPOL1	V300	

8.2 CONFIGURATION INFORMATION

The configuration information that is described in a CF definition file is divided into the following three 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 RX830.

- System Call Table (SCT) information
  Data that is used to select whether a system function (system call) is to be used.

8.2.1 Real-time OS Information

The real-time OS information that is described in a CF definition 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

8.2.2 SIT Information

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

(1) System information

The following data is defined as system information.

- Basic clock cycle
- Default stack size
- Stack information for interrupt handler
  - Stack size for interrupt handler
  - Type of system memory that is assigned to the interrupt handler stack
- Range of protected task ID numbers
- Range of protected semaphore ID numbers
- Range of protected event flag ID numbers
- Range of protected mailbox ID numbers
- Range of protected memory pool ID numbers

(2) System maximum value information

The following data is defined as system maximum value information.

- Task priority range
- Maximum number of tasks that can be created
- Maximum number of semaphores that can be created
- Maximum number of event flags that can be created
- Maximum number of mailboxes that can be created
- Maximum number of memory pools that can be created
- Maximum number of cyclic handlers that can be registered
- Maximum number of extended SVC handlers that can be registered

(3) Memory information

For each system memory (System Memory Pool 0, System Memory Pool 1, User Memory Pool 0, User Memory Pool 1), the following data is defined.

- Types of system memory
- Start address of system memory
- Size of system memory

(4) Task information

For each task, the following data is defined.

- Task ID
- Task initial status
- Task activation code
- Task extended information
- Language used to describe the task
- Task activation address
- Task initial priority (assigned upon activation)
- Interrupt status
- Status of stack used by task
  - Size of stack used by task
  - Type of system memory allocated to stack used by task
- GP register-specific value for a task
- TP register-specific value for a task
- Key ID number of a task

(5) Semaphore information

For each semaphore, the following data is defined.

- Semaphore ID
- Semaphore extended information
- Method used to queue tasks
- Initial resource count for semaphore
- Maximum resource count for semaphore
- Key ID number for semaphore

(6) Event flag information

For each event flag, the following data is defined.

- Event flag ID
- Event flag extended information
- Whether waiting for multiple tasks is enabled/prohibited
- Initial bit pattern of event flag
- Key ID number of event flag

(7) Mailbox information

For each mailbox, the following data can be defined.

- Mailbox ID
- Mailbox extended information
- Method used to queue tasks
- Method used to queue messages
- Key ID number of mailbox

(8) Interrupt handler information

For each interrupt handler, the following data is defined.

- Interrupt level
- Language used to describe the interrupt handler
- Activation address of interrupt handler
- GP register-specific value for interrupt handler
- TP register-specific value for interrupt handler

(9) Memory pool information

For each memory pool, the following data is defined.

- Memory pool ID
- Memory pool extended information
- Method used to queue tasks
- Memory pool information
  - Memory pool size
  - Type of system memory allocated to memory pool
- Key ID number of memory pool

(10) Cyclic handler information

For each cyclic handler, the following data is defined.

- Specification number of cyclic handler
- Extended information for cyclic handler
- Language used to describe the cyclic handler
- Activation address of the cyclic handler
- Initial activation status of cyclic handler
- Activation interval for cyclic handler
- GP register-specific value for cyclic handler
- TP register-specific value for cyclic handler

(11) Extended SVC handler information

For each extended SVC handler, the following data is defined.

- Function number of extended SVC handler
- Language used to describe the extended SVC handler
- Activation address of extended SVC handler
- GP register-specific value for extended SVC handler
- TP register-specific value for extended SVC handler

(12) Initialization handler information

The following data is defined as initialization handler information.

- Language used to describe the initialization handler
- Activation address of initialization handler
- GP register-specific value for initialization handler
- TP register-specific value for initialization handler

8.2.3 SCT Information

The SCT information that is described in a CF definition file consists of the following eight items.

(1) Task management/task-associated synchronization management system call information

Defines, as the task management/task-associated synchronization management system call information, the task management/task-associated synchronization management system calls used by a user program.

The task management/task-associated synchronization management system calls supported by RX830 are listed below.

```
cre_tsk     del_tsk     sta_tsk     ext_tsk     exd_tsk
ter_tsk     dis_dsp     ena_dsp     chg_pri     rot_rdq
rel_wai     get_tid     ref_tsk     vget_tid    sus_tsk
rsm_tsk     frsm_tsk    slp_tsk     tslp_tsk    wup_tsk
can_wup
```

(2) Synchronous communication (semaphore) management system call information

Defines, as the semaphore management system call information, the semaphore management system calls used by a user program.

The semaphore management system calls supported by RX830 are listed below.

```
cre_sem     del_sem     sig_sem     preq_sem    wai_sem
twai_sem    ref_sem     vget_sid
```

(3) Synchronous communication (event flag) management system call information

Defines, as the event flag management system call information, the event flag management system calls used by a user program.

The event flag management system calls supported by RX830 are listed below.

```
cre_flg     del_flg     set_flg     clr_flg     wai_flg
pol_flg     twai_flg    ref_flg     vget_fid
```

(4) Synchronous communication (mailbox) management system call information

Defines, as the mailbox management system call information, the mailbox management system calls used by a user program.

The mailbox management system calls supported by RX830 are listed below.

```
cre_mbx     del_mbx     snd_msg     rcv_msg     prcv_msg
trcv_msg    ref_mbx     vget_mid
```

(5) Interrupt processing management system call information

Defines, as the interrupt processing management system call information, the interrupt processing management system calls used by a user program.

The interrupt processing management system calls supported by RX830 are listed below.

```
def_int     ret_int     ret_wup     vret_clk    loc_cpu
unl_cpu     chg_ilv     ref_ilv
```

(6) Memory pool management system call information

Defines, as memory pool management system call information, the memory pool management system calls used by a user program.

The memory pool management system calls supported by RX830 are listed below.

```
cre_mpl     del_mpl     get_blk     pget_blk    tget_blk
rel_blk     ref_mpl     vget_pid
```

(7) Time management system call information

Defines, as the time management system call information, the time management system calls used by a user program.

The time management system calls supported by RX830 are listed below.

```
set_tim     get_tim     dly_tsk     def_cyc     act_cyc
ref_cyc
```

(8) System management system call information

Defines, as the system management system call information, the system management system calls used by a user program.

The system management system calls supported by RX830 are listed below.

```
get_ver     ref_sys     def_svc     viss_svc
```

8.3 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 CF definition file.

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

8.3.1 RX Series Information

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

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

Figure 8-1 shows the specification format for the RX series information.

Figure 8-1. RX Series Information Specification Format

```
rxsers      rtos_nam      rtos_ver
```

The items constituting the RX series information are as follows.

rtos_nam	Specifies the name of the real-time OS. The keywords that can be specified for *rtos_nam* are limited to those supported by RX830.
rtos_ver	Specifies the version number of the real-time OS. The keywords that can be specified for *rtos_ver* are limited to those supported by the V300.

8.4 SPECIFICATION FORMAT FOR SIT INFORMATION

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

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

8.4.1 System Information

The system information defines the basic clock cycle, default stack size, stack information, and the protected range of ID numbers.

For a CF definition file, the specification of the system information is required.

Figure 8-2 shows the specification format for the system information.

Figure 8-2. System Information Specification Format

```
clktim    time
defstk    stk_siz
intstk    intstk_siz:mem_nam
prttsk    tsk_idlmt
prtsem    sem_idlmt
prtflg    flg_idlmt
prtmbx    mbx_idlmt
prtmpl    mpl_idlmt
```

The items constituting the system information are as follows.

time — Specifies the basic clock cycle (ms).
A value of between `0x1` and `0xffff` can be specified for *time*.

Remark The basic clock cycle is that cycle at which RX830 generates the clock interrupts necessary to realize the time management function (cycle rise, delay rise, time-out).

stk_siz — Specifies the default stack size (bytes).
A value of between `0x0` and `0x7ffffffc`, aligned with a 4-byte boundary, can be specified for *stk_siz*.

Remark The default stack size is the smallest task stack size that can exist within the system. If, therefore, at system activation, a static task is generated or if an active task is generated as a result of a `cre_tsk` system call, and a stack size smaller than the default is specified, that specification is ignored and the default size is used.

intstk_siz:mem_nam Specifies the size of the interrupt handler stack (bytes), and the type of system memory to be allocated to the interrupt handler stack.
A value of between 0x0 and 0x7fffffac, aligned with a four-byte boundary, can be specified for *intstk_siz*.
For *mem_nam*, either of keywords SPOL0 or SPOL1 can be specified.

SPOL0: Allocates the interrupt handler stack to System Memory Pool 0.

SPOL1: Allocates the interrupt handler stack to System Memory Pool 1.

tsk_idlmt Specifies the range of protected ID numbers when a task is generated with no ID number specified.
A value of between 0x0 and *tsk_cnt* can be specified for *tsk_idlmt*.

Note When 0x0 is specified, no protections are imposed on the ID number.
The value defined for the maximum number of tasks that can be created, maxtsk, in the system maximum value information, is used for *tsk_cnt*.

sem_idlmt Specifies the range of protected ID numbers when a semaphore is generated with no ID number specified.
A value of between 0x0 and *sem_cnt* can be specified for *sem_idlmt*.

Note When 0x0 is specified, no protections are imposed on the ID number.
The value defined for the maximum number of semaphores that can be created, maxsem, in the system maximum value information, is used for *sem_cnt*.

flg_idlmt Specifies the range of protected ID numbers when an event flag is generated with no ID number specified.
A value of between 0x0 and flg_cnt can be specified for *flg_idlmt*.

Note When 0x0 is specified, no protections are imposed on the ID number.
The value defined for the maximum number of event flags that can be created, maxflg, in the system maximum value information, is used for *flg_cnt*.

mbx_idlmt

Specifies the range of protected ID numbers when a mailbox is generated with no ID number specified.
A value of between 0x0 and *mbx_cnt* can be specified for *mbx_idlmt*.

Note When 0x0 is specified, no protections are imposed on the ID number.
The value defined for the maximum number of mailboxes that can be created, maxmbx, in the system maximum value information, is used for *mbx_cnt*.

mpl_idlmt

Specifies the range of protected ID numbers when a memory pool is generated with no ID number specified.
A value of between 0x0 and *mpl_cnt* can be specified for *mpl_idlmt*.

Note When 0x0 is specified, no protections are imposed on the ID number.
The value defined for the maximum number of memory pools that can be created, maxmpl, in the system maximum value information, is used for *mpl_cnt*.

8.4.2 System Maximum Value Information

The system maximum value information defines values for the task priority range (priority value), maximum number of objects that can be created (tasks, etc.), and maximum number of objects that can be registered (cyclic handler, etc.) that are used by the system.

For a CF definition file, the description of the system maximum value information is required.

Figure 8-3 shows the specification format for the system maximum value information.

Figure 8-3. System Maximum Value Information Specification Format

```
maxpri     pri_lvl
maxtsk     tsk_cnt
maxsem     sem_cnt
maxflg     flg_cnt
maxmbx     mbx_cnt
maxmpl     mpl_cnt
maxcyc     cyc_cnt
maxsvc     svc_cnt
```

The items constituting the system maximum value information are as follows.

pri_lvl
Specifies the priority range (priority values) for the task.
A value of between 0x1 and 0xfc can be specified for *pri_lvl*.

tsk_cnt
Specifies the maximum number of tasks that can be created.
A value of between 0x1 and 0x7fff can be specified for *tsk_cnt*.

sem_cnt
Specifies the maximum number of semaphores that can be created.
A value of between 0x0 and 0x7fff can be specified for *sem_cnt*.

flg_cnt
Specifies the maximum number of event flags that can be created.
A value of between 0x0 and 0x7fff can be specified for *flg_cnt*.

mbx_cnt
Specifies the maximum number of mailboxes that can be created.
A value of between 0x0 and 0x7fff can be specified for *mbx_cnt*.

mpl_cnt
Specifies the maximum number of memory pools that can be created.
A value of between 0x0 and 0x7fff can be specified for *mpl_cnt*.

cyc_cnt
Specifies the maximum number of cyclic handlers that can be registered.
A value of between 0x0 and 0x7fff can be specified for *cyc_cnt*.

svc_cnt
Specifies the maximum number of extended SVC handlers that can be registered.
A value of between 0x0 and 0x7fff can be specified for *svc_cnt*.

8.4.3 Memory Information

As the memory information define, for each system memory (System Memory Pool 0, System Memory Pool 1, User Memory Pool 0, User Memory Pool 1), data that indicates the type, first address, and size.

For a CF definition file, the specification of the data for System Memory Pool 0 is required.

Also, for a CF definition file, when the data for User Memory Pool 1 is specified, the data for User Memory Pool 0 is required.

Note Each system memory can be divided into multiple memory areas, and those areas assigned as necessary.

Figure 8-4 shows the specification format for the memory information.

Figure 8-4. Memory Information Specification Format

```
mem   mem_id   mem_adr   mem_siz
```

The items constituting the memory information are as follows.

mem_id Specifies the type of the system memory.
The keywords that can be specified for *mem_id* are SPOL0, SPOL1, UPOL0, and UPOL1.

- SPOL0: System Memory Pool 0 is set as the system memory type.
- SPOL1: System Memory Pool 1 is set as the system memory type.
- UPOL0: User Memory Pool 0 is set as the system memory type.
- UPOL1: User Memory Pool 1 is set as the system memory type.

mem_adr Specifies the start address of the system memory.
A value of between 0x0 and 0xfffffffc, aligned with a 4-byte boundary, can be specified for *mem_adr*.

mem_siz Specifies the size of the system memory (bytes).
A value of between 0x100 and 0x7ffffffc, aligned with a 4-byte boundary, can be specified for *mem_siz*.

Note For details of how to estimate the system memory, see **Chapter 6**.

8.4.4 Task Information

As the task information define, for each task, data that indicates the ID, initial status, activation code, extended information, description language, activation address, activation priority, interrupt status, stack information, GP register-specific value, TP register-specific value, and key ID number.

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

The number of task information items that can be specified is defined as being within the range of 1 to the maximum number of tasks that can be created, *tsk_cnt*, as set in the system maximum value information.

Figure 8-5 shows the specification format for the task information.

Figure 8-5. Task Information Specification Format

```
tsk tsk_id sts sta_code ext_inf lang sta_adr pri intr stk_siz:
mem_nam data text key_id
```

The items constituting the task information are as follows.

tsk_id

Specifies the ID of the task.
A value of between 0x0 and *tsk_cnt*, or a symbol name, can be specified for *tsk_id*.

Note When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *tsk_idlmt* and *tsk_cnt*.
The value defined for the task ID number protected range, prttsk, in the system information is set as *tsk_idlmt*.
The value defined for the maximum number of tasks that can be created, maxtsk, in the system maximum value information is set as *tsk_cnt*.

sts

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

TTS_DMT: The system enters the dormant status upon being activated.

TTS_RDY: The system enters the ready status 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*.

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

ext_inf

Specifies the extended information for a task.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant task. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_tsk system call by a processing program (task/nontask).

lang

Specifies the language used to describe a task
The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

- TA_HLNG: A task is described in C.
- TA_ASM: A task is described in assembly language.

sta_adr

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

pri

Specifies the activation priority (initial priority) for a task.
A value of between 0x0 and *pri_lvl* can be specified for *pri*.

Note The value defined for the task priority range, maxpri, in the system maximum value information is set as *pri_lvl*.

intr

Specifies the interrupt status.
The keywords that can be specified for *intr* are TA_ENAINT and TA_DISINT.

- TA_ENAINT: All interrupts are enabled at task activation.
- TA_DISINT: All interrupts are disabled at task activation.

stk_siz:mem_nam

Specifies the stack size (bytes) to be used by a task, and the type of the system memory to be allocated to that stack.
A value of between 0x0 and 0x7ffffffc, aligned with a 4-byte boundary, can be specified for *stk_siz*.
The keywords that can be specified for *mem_nam* are SPOL0 and SPOL1.

- SPOL0: The task stack is allocated to System Memory Pool 0.
- SPOL1: The task stack is allocated to System Memory Pool 1.

data

Specifies a GP register-specific value for a task.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* while, as a keyword, no_use can be specified.

- no_use: At system activation, a GP register-specific value is not set.

text

Specifies a TP register-specific value for a task.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* while, as a keyword, no_use can be specified.

- no_use: At system activation, a TP register-specific value is not set.

key_id

Specifies the key ID number for a task.
A value of between 0x0 and 0x7fff can be specified for *key_id*.

Note When 0x0 is specified, the configurator does not perform key ID number assignment.

8.4.5 Semaphore Information

As the semaphore information define, for each semaphore, data that indicates the ID, extended information, task queuing method, initial resource count, maximum resource count, and key ID number.

The number of semaphore information items that can be specified is defined as being within the range of 0 to the maximum number of semaphores that can be created, *sem_cnt*, as set in the system maximum value information.

Figure 8-6 shows the specification format for the semaphore information.

Figure 8-6. Semaphore Information Specification Format

```
sem   sem_id   ext_inf   twai_opt   init_cnt   max_cnt   key_id
```

The items constituting the semaphore information are as follows.

sem_id

Specifies the ID of a semaphore.
A value of between 0x0 and *sem_cnt*, or a symbol name, can be specified for *sem_id*.

Note When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *sem_idlmt* and *sem_cnt*.
The value defined for the semaphore ID number protected range, prtsem, in the system information is set as *sem_idlmt*.
The value defined for the maximum number of semaphores that can be created, maxsem, in the system maximum value information is set as *sem_cnt*.

ext_inf

Specifies the extended information for the semaphore.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant semaphore. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_sem system call by a processing program (task/nontask).

twai_opt

Specifies the task queuing method.
The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

TA_TFIFO: Tasks are queued in the same order as that in which resource allocation is requested.

TA_TPRI: Tasks are queued in order of their priority.

init_cnt Specifies the initial resource count for a semaphore.
A value of between 0x0 and max_cnt can be specified for *init_cnt*.

max_cnt Specifies the maximum resource count for a semaphore.
A value of between 0x1 and 0x7fffffff can be specified for *max_cnt*.

key_id Specifies the key ID number for a semaphore.
A value of between 0x0 and 0x7fff can be specified for *key_id*.

Note When 0x0 is specified, the configurator does not assign a key ID number.

8.4.6 Event Flag Information

As the event flag information define, for each event flag, data that indicates the ID, extended information, whether wait for multiple tasks is enabled/prohibited, the initial bit pattern, and key ID number.

The number of event flag information items that can be specified is defined as being within the range of 0 to the maximum number of event flags that can be created, *flg_cnt*, as set in the system maximum value information.

Figure 8-7 shows the specification format for the event flag information.

Figure 8-7. Event Flag Information Specification Format

```
flg   flg_id   ext_inf   twai_opt   init_ptn   key_id
```

The items constituting the event flag information are as follows.

flg_id

Specifies an ID for an event flag.
A value of between 0x0 and *flg_cnt*, or a symbol name, can be specified for *flg_id*.

Note When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *flg_idlmt* and *flg_cnt*.
The value defined for the event flag ID number protected range, prtflg, in the system information is set as *flg_idlmt*.
The value defined for the maximum number of event flags that can be created, maxflg, in the system maximum value information is set as *flg_cnt*.

ext_inf

Specifies the extended information for the event flag.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant event flag. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_flg system call by a processing program (task/nontask).

twai_opt

Specifies whether wait for multiple tasks is enabled/prohibited.
The keywords that can be specified for *twai_opt* are TA_WSGL and TA_WMUL.

TA_WSGL: Wait for multiple tasks is prohibited.

TA_WMUL: Wait for multiple tasks is enabled.

init_ptn

Specifies the initial bit pattern for the event flag.
A value of between 0x0 and 0xffffffff can be specified for *init_ptn*.

key_id

Specifies the key ID number for an event flag.
A value of between 0x0 and 0x7fff can be specified for *key_id*.

Note When 0x0 is specified, the configurator does not assign a key ID number.

8.4.7 Mailbox Information

As the mailbox information define, for each mailbox, data that indicates the ID, extended information, task queuing method, message queuing method, and key ID number.

The number of mailbox information items that can be specified is defined as being within the range of 0 to the maximum number of mailboxes that can be created, *mbx_cnt*, as set in the system maximum value information.

Figure 8-8 shows the specification format for the mailbox information.

Figure 8-8. Mailbox Information Specification Format

```
mbx   mbx_id   ext_inf   twai_opt   mwai_opt   key_id
```

The items constituting the mailbox information are as follows.

mbx_id

Specifies an ID for a mailbox.
A value of between 0x0 and *mbx_cnt*, or a symbol name, can be specified for *mbx_id*.

Note When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *mbx_idlmt* and *mbx_cnt*.
The value defined for the mailbox ID number protected range, prtmbx, in the system information is set as *mbx_idlmt*.
The value defined for the maximum number of mailboxes that can be created, maxmbx, in the system maximum value information is set as *mbx_cnt*.

ext_inf

Specifies the extended information for the mailbox.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant mailbox. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_mbx system call by a processing program (task/nontask).

twai_opt

Specifies the task queuing method.
The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

TA_TFIFO: Tasks are queued in the same order as that in which message reception is requested.

TA_TPRI: Tasks are queued according to their priority.

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.

key_id

Specifies the key ID number for a mailbox.
A value of between 0x0 and 0x7fff can be specified for *key_id*.

Note When 0x0 is specified, the configurator does not assign a key ID number.

8.4.8 Interrupt Handler Information

As the interrupt handler information define, for each interrupt handler, data that indicates the interrupt level, description language, activation address, GP register-specific value, and TP register-specific value.

For each interrupt level, a single item of interrupt handler information can be defined.

Figure 8-9 shows the specification format for the interrupt handler information.

Figure 8-9. Interrupt Handler Information Specification Format

```
inthdr   int_lvl   lang   hdr_adr   data   text
```

The items constituting the interrupt handler information are as follows.

int_lvl
Specifies the interrupt level.
A value of between 0x0 and 0xf can be specified for *int_lvl*.

lang
Specifies the language used to describe the interrupt handler.
The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

- TA_HLNG: The interrupt handler is described in C.
- TA_ASM: The interrupt handler is described in assembly language.

hdr_adr
Specifies the activation address of the interrupt handler.
A value of between 0x0 and 0xfffffffe, aligned with a two-byte boundary, or a symbol name, can be specified for *hdr_adr*.

data
Specifies the GP register-specific value for the interrupt handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* while, as a keyword, no_use can be specified.

- no_use: At system activation, a GP register-specific value is not set.

text
Specifies the TP register-specific value for the interrupt handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* while, as a keyword, no_use can be specified.

- no_use: At system activation, a TP register-specific value is not set.

8.4.9 Memory Pool Information

As the memory pool information define, for each memory pool, data that indicates the ID, extended information, task queuing method, memory pool information, and key ID number.

The number of memory pool information items that can be specified is defined as being within the range of 0 to the maximum number of memory pools that can be created, *mpl_cnt*, as set in the system maximum value information.

Figure 8-10 shows the specification format for the memory pool information.

Figure 8-10. Memory Pool Information Specification Format

```
mpl   mpl_id   ext_inf   twai_opt   mpl_siz:mem_nam   key_id
```

The items constituting the memory pool information are as follows.

mpl_id

Specifies an ID for a memory pool.
A value of between 0x0 and *mpl_cnt*, or a symbol name, can be specified for *mpl_id*.

Note When 0x0 or a symbol name is specified, the configurator automatically assigns an unused ID number between *mpl_idlmt* and *mpl_cnt*.
The value defined for the memory pool ID number protected range, prtmpl, in the system information is set as *mpl_idlmt*.
The value defined for the maximum number of memory pools that can be created, maxmpl, in the system maximum value information is set as *mpl_cnt*.

ext_inf

Specifies the extended information for the memory pool.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant memory pool. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_mpl system call by a processing program (task/nontask).

twai_opt

Specifies the task queuing method.
The keywords that can be specified for *twai_opt* are TA_TFIFO and TA_TPRI.

TA_TFIFO: Tasks are queued in the same order as that in which the allocation of memory blocks is requested.

TA_TPRI: Tasks are queued according to their priority.

mpl_siz:mem_nam

Specifies the memory pool size (bytes), and the type of the system memory to be allocated to that memory pool.
A value of between 0x100 and 0x7ffffffc, aligned with a four-byte boundary, can be specified for *mpl_siz*.
The keywords that can be specified for *mem_nam* are UPOL0 and UPOL1.

- UPOL0: The memory pool is allocated to User Memory Pool 0
- UPOL1: The memory pool is allocated to User Memory Pool 1

key_id

Specifies the key ID number for a memory pool.
A value of between 0x0 and 0x7fff can be specified for *key_id*.

Note When 0x0 is specified, the configurator does not assign a key ID number.

8.4.10 Cyclic Handler Information

As the cyclic handler information define, for each cyclic handler, data that indicates the specification number, extended information, description language, activation address, initial activation status, activation interval, GP register-specific value, and TP register-specific value.

The number of cyclic handler information items that can be specified is defined as being within the range of 0 to the maximum number of cyclic handlers that can be registered, *cyc_cnt*, as set in the system maximum value information.

Figure 8-11 shows the specification format for the cyclic handler information.

Figure 8-11. Cyclic Handler Information Specification Format

```
cyc  cyc_no  ext_inf  lang  hdr_adr  act  intvl  data  text
```

The items constituting the cyclic handler information are as follows.

cyc_no Specifies the specification number for the cyclic handler.
A value of between 0x1 and *cyc_cnt*, or a symbol name, can be specified for *cyc_no*.

Note When a symbol name is specified, the configurator automatically assigns an unused specification number between 0x1 and *cyc_cnt*.
The value defined for the maximum number of cyclic handlers that can be registered, maxcyc, in the system maximum value information is set as *cyc_cnt*.

ext_inf Specifies the extended information for the cyclic handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *ext_inf*.

Note *ext_inf* is provided to enable the specification of user own information for the relevant cyclic handler. The user can specify it as necessary.
The value specified for *ext_inf* can be dynamically allocated upon the issue of a ref_cyc system call by a processing program (task/nontask).

lang Specifies the language used to describe the cyclic handler.
The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

- TA_HLNG: The cyclic handler is described in C.
- TA_ASM: The cyclic handler is described in assembly language.

hdr_adr Specifies the activation address of the cyclic handler.
A value of between 0x0 and 0xfffffffe, aligned with a two-byte boundary, or a symbol name, can be specified for *hdr_adr*.

act Specifies the initial activation status of the cyclic handler.
The keywords that can be specified for *act* are TCY_ON and TCY_OFF.

- TCY_ON: At system activation, the activation status is set to ON.
- TCY_OFF: At system activation, the activation status is set to OFF.

intvl Specifies the activation interval (in basic clock cycles) for the cyclic handler.
A value of between 0x1 and 0x7fffffff can be specified for *intvl*.

data Specifies a GP register-specific value for the cyclic handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* while, as a keyword, no_use can be specified.

- no_use: At system activation, a GP register-specific value is not set.

text Specifies a TP register-specific value for the cyclic handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* while, as a keyword, no_use can be specified.

- no_use: At system activation, a TP register-specific value is not set.

8.4.11 Extended SVC Handler Information

As the extended SVC handler information define, for each extended SVC handler, data that indicates the function number, description language, activation address, GP register-specific value, and TP register-specific value.

The number of extended SVC handler information items that can be specified is defined as being within the range of 0 to the maximum number of extended SVC handlers that can be registered, *svc_cnt*, as set in the system maximum value information.

Figure 8-12 shows the specification format for the extended SVC handler information.

Figure 8-12. Extended SVC Handler Information Specification Format

```
svc   svc_no   lang   hdr_adr   data   text
```

The items constituting the extended SVC handler information are as follows.

svc_no

Specifies the function number for the extended SVC handler.
A value of between 0x1 and *svc_cnt*, or a symbol name, can be specified for *svc_no*.

Note When a symbol name is specified, the configurator automatically assigns an unused function number between 0x1 and *svc_cnt*.
The value defined for the maximum number of extended SVC handlers that can be registered, maxsvc, in the system maximum value information is set as *svc_cnt*.

lang

Specifies the language used to describe the extended SVC handler.
The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

TA_HLNG: The extended SVC handler is described in C.

TA_ASM: The extended SVC handler is described in assembly language.

hdr_adr

Specifies the activation address of the extended SVC handler.
A value of between 0x0 and 0xfffffffe, aligned with a two-byte boundary, or a symbol name, can be specified for *hdr_adr*.

data

Specifies a GP register-specific value for the extended SVC handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* while, as a keyword, no_use can be specified.

no_use: At system activation, a GP register-specific value is not set.

text

Specifies a TP register-specific value for the extended SVC handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* while, as a keyword, no_use can be specified.

no_use: At system activation, a TP register-specific value is not set.

8.4.12 Initialization Handler Information

As the initialization handler information define, for each initialization handler, data that indicates the description language, activation address, GP register-specific value, and TP register-specific value.

For a CF definition file, the specification of the initialization handler information is required.

Figure 8-13 shows the specification format for the initialization handler information.

Figure 8-13. Initialization Handler Information Specification Format

```
ini   lang   hdr_adr   data   text
```

The items constituting the initialization handler information are as follows.

lang

Specifies the language used to describe the initialization handler.
The keywords that can be specified for *lang* are TA_HLNG and TA_ASM.

- TA_HLNG: The initialization handler is described in C.
- TA_ASM: The initialization handler is described in assembly language.

hdr_adr

Specifies the activation address of the initialization handler.
A value of between 0x0 and 0xfffffffe, aligned with a two-byte boundary, or a symbol name, can be specified for *hdr_adr*.

data

Specifies a GP register-specific value for the initialization handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *data* while, as a keyword, no_use can be specified.

- no_use: At system activation, a GP register-specific value is not set.

text

Specifies a TP register-specific value for the initialization handler.
A value of between 0x0 and 0xffffffff, or a symbol name, can be specified for *text* while, as a keyword, no_use can be specified.

- no_use: At system activation, a TP register-specific value is not set.

8.5 SPECIFICATION FORMAT FOR SCT INFORMATION

The following describes the format that must be observed when describing the SCT information in the CF definition file.

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

8.5.1 Task Management/Task-Associated Synchronization Management System Call Information

As the task management/task-associated synchronization management system call information define, for each system call, data that indicates the task management/task-associated synchronization management system calls that are used by a user processing program.

Figure 8-14 shows the specification format for the task management/task-associated synchronization management system call information.

Figure 8-14. Task Management/Task-Associated Synchronization Management System Call Information Specification Format

```
tsksvc   svc_nam
```

The items constituting the task management/task-associated synchronization management system call information are as follows.

svc_nam Specifies a task management/task-associated synchronization management system call name.
The following keywords can be specified for *svc_nam*.

cre_tsk	del_tsk	sta_tsk	ext_tsk
exd_tsk	ter_tsk	dis_dsp	ena_dsp
chg_pri	rot_rdq	rel_wai	get_tid
ref_tsk	vget_tid	sus_tsk	rsm_tsk
frsm_tsk	slp_tsk	tslp_tsk	wup_tsk
can_wup			

8.5.2 Synchronous Communication (Semaphore) Management System Call Information

As the semaphore management system call information define, for each system call, data that indicates the semaphore management system calls that are used by a user processing program.

Figure 8-15 shows the specification format for the semaphore management system call information.

Figure 8-15. Semaphore Management System Call Information Specification Format

```
semsvc   svc_nam
```

The items constituting the semaphore management system call information are as follows.

svc_nam Specifies a semaphore management system call name.
The following keywords can be specified for *svc_nam*.

cre_sem	del_sem	sig_sem	wai_sem
preq_sem	twai_sem	ref_sem	vget_sid

8.5.3 Synchronous Communication (Event Flag) Management System Call Information

As the event flag management system call information define, for each system call, data that indicates the event flag management system calls that are used by a user processing program.

Figure 8-16 shows the specification format for the event flag management system call information.

Figure 8-16. Event Flag Management System Call Information Specification Format

```
flgsvc   svc_nam
```

The items constituting the event flag management system call information are as follows.

svc_nam Specifies an event flag management system call name.
The following keywords can be specified for *svc_nam*.

cre_flg	del_flg	set_flg	clr_flg
wai_flg	pol_flg	twai_flg	ref_flg
vget_fid			

8.5.4 Synchronous Communication (Mailbox) Management System Call Information

As the mailbox management system call information define, for each system call, data that indicates the mailbox management system calls that are used by a user processing program.

Figure 8-17 shows the specification format for the synchronous communication (mailbox) management system call information.

Figure 8-17. Mailbox Management System Call Information Specification Format

```
mbxsvc   svc_nam
```

The items constituting the mailbox management system call information are as follows.

svc_nam Specifies a mailbox management system call name.
The following keywords can be specified for *svc_nam*.

cre_mbx	del_mbx	snd_msg	rcv_msg
prcv_msg	trcv_msg	ref_mbx	vget_mid

8.5.5 Interrupt Processing Management System Call Information

As the interrupt processing management system call information define, for each system call, data that indicates the interrupt processing management system calls that are used by a user processing program.

Figure 8-18 shows the specification format for the interrupt processing management system call information.

Figure 8-18. Interrupt Processing Management System Call Information Specification Format

```
intsvc   svc_nam
```

The items constituting the interrupt processing management system call information are as follows.

svc_nam Specifies an interrupt processing management system call name.
The following keywords can be specified for *svc_nam*.

def_int	ret_int	ret_wup	vret_clk
loc_cpu	unl_cpu	chg_ilv	ref_ilv

8.5.6 Memory Pool Management System Call Information

As the memory pool management system call information define, for each system call, data that indicates the memory pool management system calls that are used by a user processing program.

Figure 8-19 shows the specification format for the memory pool management system call information.

Figure 8-19. Memory Pool Management System Call Information Specification Format

```
mplsvc   svc_nam
```

The items constituting the memory pool management system call information are as follows.

svc_nam Specifies a memory pool management system call name.
The following keywords can be specified for *svc_nam*.

cre_mpl	del_mpl	get_blk	pget_blk
tget_blk	rel_blk	ref_mpl	vget_pid

8.5.7 Time Management System Call Information

As the time management system call information define, for each system call, data that indicates the time management system calls that are used by a user processing program.

Figure 8-20 shows the specification format for the time management system call information.

Figure 8-20. Time Management System Call Information Specification Format

```
timsvc   svc_nam
```

The items constituting the time management system call information are as follows.

svc_nam Specifies a time management system call name.
The following keywords can be specified for *svc_nam*.

set_tim	get_tim	dly_tsk	def_cyc
act_cyc	ref_cyc		

8.5.8 System Management System Call Information

As the system management system call information define, for each system call, data that indicates the system management system calls used by a user processing program.

Figure 8-21 shows the specification format for the system management system call information.

Figure 8-21. System Management System Call Information Specification Format

```
syssvc   svc_nam
```

The items constituting the system management system call information are as follows.

svc_nam Specifies a system management system call name.
The following keywords can be specified for *svc_nam*.

```
get_ver      ref_sys      def_svc      viss_svc
```

8.6 CAUTIONS

In a CF definition file, describe the configuration information (real-time OS information, SIT information, SCT 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
(5) Declaration of the start of the SCT information description
(6) SCT information description

Figure 8-22 illustrates how a CF definition file is described.

Figure 8-22. Describing a CF Definition 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
.....
.....
.....

-- Declaration of start of SCT information description
sct_def

-- SCT information description
.....
.....
.....
```

8.7 EXAMPLE DESCRIPTION

Figure 8-23 shows an example CF definition file description.
The example shown in Figure 8-23 describes the following data.

- Real-time OS information
  - RX series information

    Real-time OS: `RX830`

    Version: `V300`

- SIT information
  - System information

    Basic clock cycle: `0x1` ms

    Default stack size: `0x100` bytes

    Interrupt handler stack information: Assures the memory area between System Memory Pool 0 and `0x100` bytes

    Range of protected task ID number: `0x1`

    Range of protected semaphore ID number: `0x1`

    Range of protected event flag ID number: `0x1`

    Range of protected mailbox ID number: `0x1`

    Range of protected memory pool ID number: `0x1`

  - System maximum value information

    Task priority range: `0xf`

    Max. number of tasks that can be created: `0x2`

    Max. number of semaphores that can be created: `0x1`

    Max. number of event flags that can be created: `0x2`

    Max. number of mailboxes that can be created: `0x3`

    Max. number of memory pools that can be created: `0x2`

    Max. number of cyclic handlers that can be registered: `0x1`

    Max. number of extended SVC handlers that can be registered: `0x1`

  - Memory information

    System Memory Pool 0: Assures the memory area between address `0x0` and `0x2000` bytes

    System Memory Pool 1: Assures the memory area between address `0x2000` and `0x1000` bytes

    User Memory Pool 0: Assures the area between address `0x3000` and `0x7000` bytes

    User Memory Pool 0: Assures the memory area between `0x20000` and `0x2500` bytes

    User Memory Pool 1: Assures the memory area between `0x30000` and `0x1500` bytes

- Task information

ID:	0x1
Initial status:	ready
Activation code:	0x0
Extended information:	0x0
Description language:	Assembly language
Activation address:	_task01
Initial priority:	0x8
Interrupt status:	All interrupts enabled
Stack information:	Assures memory area between System Memory Pool 0 and 0x100 bytes
GP register-specific value:	Not set
TP register-specific value:	Not set
Key ID number:	0x1

ID:	0x2
Initial status:	dormant
Activation code:	0x0
Extended information:	0x0
Description language:	C
Activation address:	_task02
Initial priority:	0xf
Interrupt status:	All interrupts disabled
Stack information:	Assures memory area between System Memory Pool 0 and 0x100 bytes
GP register-specific value:	Not set
TP register-specific value:	Not set
Key ID number:	0x2

- Semaphore information

ID:	0x1
Extended information:	0x0
Task queuing:	Same order as that in which resource requests are issued (FIFO)
Initial resource count:	0xff
Maximum resource count:	0xff
Key ID number:	0x1

– Event flag information

ID:	0x1
Extended information:	0x0
Multiple task waiting:	Disabled
Initial bit pattern:	0x0
Key ID number:	0x1

– Mailbox information

ID:	0x1
Extended information:	0x0
Task queuing method:	Same order as that in which message receive requests are issued (FIFO)
Message queuing method:	Same order as that in which messages are issued (FIFO)
Key ID number:	0x1

– Interrupt handler information

Interrupt level:	0x0
Description language:	Assembly language
Activation address:	_inthdr01
GP register-specific value:	Not set
TP register-specific value:	Not set

– Memory pool information

ID:	0x1
Extended information:	0x0
Task queuing method:	According to task priority
Memory pool information:	Assures memory area between User Memory Pool 0 and 0x2000 bytes
Key ID number:	0x1

– Cyclic handler information

Specification number:	0x1
Extended information:	0x0
Description language:	C
Activation address:	_cychdr01
Initial activation status:	OFF
Activation interval:	0x100 basic block cycles
GP register-specific value:	Not set
TP register-specific value:	Not set

– Extended SVC handler information

Function number:	`0x1`
Description language:	C
Activation address:	`_svchdr01`
GP register-specific value:	Not set
TP register-specific value:	Not set

– Initialization handler information

Description language:	Assembly language
Activation address:	`_varfunc`
GP register-specific value:	Not set
TP register-specific value:	Not set

- SCT information
  – Task management/Task-associated synchronization management system call information

    As the task management/task-associated synchronization management system calls used by a user processing program, define the following:

```
sta_tsk          exd_tsk
```

  – Synchronous communication (semaphore) management system call information

    As the synchronous communication (semaphore) management system calls used by a user processing program, define the following:

```
sig_sem          wai_sem
```

  – Synchronous communication (event flag) management system call information

    As the synchronous communication (event flag) management system calls used by a user processing program, define the following:

```
cre_flg          del_flg            set_flg            wai_flg
```

  – Synchronous communication (mailbox) management system call information

    As the synchronous communication (mailbox) management system calls used by a user processing program, define the following:

```
cre_mbx          del_mbx            snd_msg            rcv_msg
```

  – Interrupt processing management system call information

    As the interrupt processing management system call used by a user processing program, define the following:

```
vret_clk
```

– Memory pool management system call information

  As the memory pool management system calls used by a user processing program, define the following:

```
cre_mpl          del_mpl            get_blk            rel_blk
```

– Time management system call information

  As the time management system calls used by a user processing program, define the following:

```
act_cyc          ref_cyc
```

– System management system call information

  As the system management system call used by a user processing program, define the following:

```
viss_svc
```

Figure 8-23. Example CF Definition File Description (1/4)

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

------------------------------------------------------------
-- Real-time OS information description
------------------------------------------------------------
-- RX series information
rxsers  RX830  V300

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

------------------------------------------------------------
-- SIT information description
------------------------------------------------------------
-- System information
clktim  0x1
defstk  0x100
intstk  0x100:SPOL0
prttsk  0x1
prtsem  0x1
prtflg  0x1
prtmbx  0x1
prtmpl  0x1

-- System maximum value information
maxpri  0xf
maxtsk  0x2
maxsem  0x1
maxflg  0x2
maxmbx  0x3
```

Figure 8-23. Example CF Definition File Description (2/4)

```
maxmpl  0x2
maxcyc  0x1
maxsvc  0x1

-- Memory information
mem     SPOL0   0x0      0x2000
mem     SPOL1   0x2000   0x1000
mem     UPOL0   0x3000   0x7000
mem     UPOL0   0x20000  0x2500
mem     UPOL1   0x30000  0x1500

-- Task information
tsk     0x1 TTS_RDY      0x0           0x0      TA_ASM    _task01  \
        0x8 TA_ENAINT    0x100:SPOL0   no_use   no_use    0x1
tsk     0x2 TTS_DMT      0x0           0x0      TA_HLNG   _task02  \
        0xf TA_DISINT    0x100:SPOL0   no_use   no_use    0x2

-- Semaphore information
sem     0x1  0x0  TA_TFIFO  0xff  0xff  0x1

-- Event flag information
flg     0x1  0x0  TA_WSGL  0x0  0x1

-- Mailbox information
mbx     0x1  0x0  TA_TFIFO  TA_MFIFO  0x1

-- Interrupt handler information
inthdr  0x0  TA_ASM  _inthdr01  no_use  no_use

-- Memory pool information
mpl     0x1  0x0  TA_TPRI  0x2000:UPOL0  0x1

-- Cyclic handler information
cyc     0x1  0x0  TA_HLNG  _cychdr01  TCY_OFF  0x100  no_use  no_use
```

Figure 8-23. Example CF Definition File Description (3/4)

```
-- Extended SVC handler information
svc      0x1  TA_HLNG  _svchdr01  no_use  no_use

-- Initialization handler information
ini      TA_ASM  _varfunc  no_use  no_use
------------------------------------------------------------
-- Declaration of start of SCT information description
------------------------------------------------------------
sct_def

------------------------------------------------------------
-- SCT information description
------------------------------------------------------------
-- Task management/task-associated synchronization management system call information
tsksvc   sta_tsk
tsksvc   exd_tsk

-- Synchronous communication (semaphore) management system call information
semsvc   sig_sem
semsvc   wai_sem

-- Synchronous communication (event flag) management system call information
flgsvc   cre_flg
flgsvc   del_flg
flgsvc   set_flg
flgsvc   wai_flg

-- Synchronous communication (mailbox) management system call information
mbxsvc   cre_mbx
mbxsvc   del_mbx
mbxsvc   snd_msg
mbxsvc   rcv_msg

-- Interrupt processing management system call information
intsvc   vret_clk
```

Figure 8-23. Example CF Definition File Description (4/4)

```
-- Memory pool management system call information
mplsvc   cre_mpl
mplsvc   del_mpl
mplsvc   get_blk
mplsvc   rel_blk

-- Time management system call information
timsvc   act_cyc
timsvc   ref_cyc

-- System management system call information
syssvc   viss_svc
```

[MEMO]

CHAPTER 9 INFORMATION FILE GENERATION

This chapter explains how an information file (system information table, branch table, system information header file) is created from a CF definition file.

9.1 OUTLINE

An information file (system information table, branch table, system information header file) is created from a CF definition file by activating the configurator from the command line.

The following explains how to activate the configurator from the command line.

Note that, in the input example, "A>" indicates the shell prompt, while "<cr>" indicates that the user must press the return key.

Those activation options that appear in square brackets ([ ]) can be omitted.

Note The configurator of the version specific to CA830 is activated from VSH. The configurator of the version specific to CCV830 is activated from the MS-DOS prompt.

```
A> cf830 [-i sit_file][-c sct_file][-d h_file][-ni][-nc][-nd][-V][-help] cf_file
   <cr>
```

The configurator activation options are as follows.

`-i` *`sit_file`*

Specifies the file (system information table name) output by the configurator.

If omitted: The system appends an "`i`" to the end of the CF definition 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.

`-c` *`sct_file`*

Specifies the file (branch table name) output by the configurator.

If omitted: The system appends a "`c`" to end of the CF definition file name, specified with *`cf_file`*, changes the extension to `.tbl`, and outputs the file as the branch table.

Note When both this activation option and the `-nc` option are specified at the same time, only that which was input last is effective.

`-d` *`h_file`*

Specifies the file (system information header file name) output by the configurator.

If omitted: The system changes the extension of the CF definition 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.

`-ni`

Disables output of the system information table.

If omitted: The system appends an "`i`" to end of the CF definition 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.

`-nc`

Disables output of the branch table.

If omitted: The system appends a "c" to end of the CF definition file name, specified with *`cf_file`*, changes the extension to `.tbl`, and outputs the file as the branch table.

Note When both this activation option and the `-c` *`sct_file`* option are specified at the same time, only that which was input last is effective.

`-nd`

Disables output of the system information header file.

If omitted: The system changes the extension of the CF definition 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 the version of the configurator to the standard output (or standard error output).

If omitted: The version of the configurator is not output.

Note When this activation option is specified, all other activation options are disabled.

`-help` Outputs, to the standard output (or standard error output), an explanation of how to use the activation options of the configurator.

If omitted: An explanation of how to use the activation options of the configurator is not output.

Note When both this activation option is specified, all other activation options are disabled.

`cf_file` Specifies the file (CF definition file name) input to the configurator.

If omitted: This activation option cannot be omitted.

9.2 COMMAND INPUT EXAMPLES

The following are examples of configurator command input.

- `cf830 -i sitfile.tbl -c sctfile.tbl -d hfile.h cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `sitfile.tbl`, branch table `sctfile.tbl`, and system information header file `hfile.h`.

- `cf830 -i sitfile.tbl cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `sitfile.tbl`, branch table `cffilec.tbl`, and system information header file `cffile.h`.

- `cf830 -c sctfile.tbl cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `cffilei.tbl`, branch table `sctfile.tbl`, and system information header file `cffile.h`.

- `cf830 -d hfile.h cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `cffilei.tbl`, branch table `cffilec.tbl`, and system information header file `hfile.h`.

- `cf830 cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `cffilei.tbl`, branch table `cffilec.tbl`, and system information header file `cffile.h`.

- `cf830 -nc -nd cffile.cf`
  Reads CF definition file `cffile.cf`, and outputs system information table `cffilei.tbl`.

- `cf830 -V`
  Outputs, to the standard output (or standard error output), the version of the configurator.

- `cf830 -help`
  Outputs, to the standard output (or standard error output), an explanation of how to use the activation options of the configurator.

9.3 MESSAGES

Messages are output, to the standard output (or standard error output) if, during processing by the configurator, an error is found in the definition of the CF definition file, or in the description of the coding.

The messages output by the configurator are divided into three levels (fatal errors, non-fatal errors, and warnings). Upon the output of a message, its level is indicated by a letter, as follows.

F... Fatal error

Upon the occurrence of a fatal error, a message is output, after which processing by the configurator is terminated.

Example: Memory area shortfall.

E... Non-fatal error

Upon the occurrence of a non-fatal error, a message is output, after which processing by the configurator is terminated.

Example: When a definition is duplicated.

W... Warning

Upon the occurrence of a warning, a message is output, and processing by the configurator continues.

Example: When the description of a parameter has been omitted.

[MEMO]

INDEX

D

E

F

G

H

I

K

L

M

N

O

P

R

S

V

W

[MEMO]

Phase-out/Discontinued

NEC

Facsimile Message

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.

From:

Name

Company

Tel. FAX

Address

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: +55-11-6465-6829

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

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

Taiwan
NEC Electronics Taiwan Ltd.
Fax: 02-2719-5951

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

Japan
NEC Semiconductor Technical Hotline
Fax: 044-548-7900

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	❑	❑	❑	❑