

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

CAN SOFTWARE DRIVERS

Document No. U16844EJ3V0UM00 (3rd edition)

Date Published August 2007 NS

© NEC Electronics Corporation 2003

Printed in Japan

[MEMO]

Windows is either registered trademark or trademark of Microsoft Corporation in the United States and/or other countries.

Multi is a trademark of Green Hills Software, Inc.

PC/AT is a trademark of International Business Machines Corporation.

The names of other companies and products are the registered trademarks or trademarks of each company.

• **The information in this document is current as of July, 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**

- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.
- NEC 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 NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or 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 a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.
- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.
- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"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.

"Special": 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, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)

- (1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
- (2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

[MEMO]

INTRODUCTION

Readers	This manual is intended for user engineers who wish to understand the functions of the CAN software drivers and design and develop application systems and programs for these devices.																
Purpose	This manual is intended to give users an understanding of the functions described in the Organization below.																
Organization	<p>This manual consists of the following items.</p> <ul style="list-style-type: none">• Product overview• Installation• System build• Configuration• Driver functions• Sample program																
How to Read This Manual	<p>It is assumed that the readers of this manual have general knowledge of electrical engineering, logic circuits, and microcontrollers.</p> <p>To gain a general understanding of CAN software driver functions: → Read this manual in the order of the CONTENTS. The mark <R> shows major revised points. The revised points can be easily searched by copying an "<R>" in the PDF file and specifying it in the "Fine what:" field.</p>																
Conventions	<table><tr><td>Data significance:</td><td>Higher digits on the left and lower digits on the right</td></tr><tr><td>Active low representation:</td><td>$\overline{\text{xxx}}$ (overscore over pin and signal name)</td></tr><tr><td>Note:</td><td>Footnote for item marked with Note in the text</td></tr><tr><td>Caution:</td><td>Information requiring particular attention</td></tr><tr><td>Remark:</td><td>Supplementary information</td></tr><tr><td>Numeric representation:</td><td>Binary ... xxxx or xxxxB Decimal ... xxxx Hexadecimal ... xxxxH</td></tr><tr><td>Units for representing powers of 2 (address space or memory space):</td><td>K (kilo): $2^{10} = 1,024$ M (mega): $2^{20} = 1,024^2$ G (giga): $2^{30} = 1,024^3$</td></tr><tr><td>Data type:</td><td>Word ... 32 bits Halfword ... 16 bits Byte ... 8 bits</td></tr></table>	Data significance:	Higher digits on the left and lower digits on the right	Active low representation:	$\overline{\text{xxx}}$ (overscore over pin and signal name)	Note:	Footnote for item marked with Note in the text	Caution:	Information requiring particular attention	Remark:	Supplementary information	Numeric representation:	Binary ... xxxx or xxxxB Decimal ... xxxx Hexadecimal ... xxxxH	Units for representing powers of 2 (address space or memory space):	K (kilo): $2^{10} = 1,024$ M (mega): $2^{20} = 1,024^2$ G (giga): $2^{30} = 1,024^3$	Data type:	Word ... 32 bits Halfword ... 16 bits Byte ... 8 bits
Data significance:	Higher digits on the left and lower digits on the right																
Active low representation:	$\overline{\text{xxx}}$ (overscore over pin and signal name)																
Note:	Footnote for item marked with Note in the text																
Caution:	Information requiring particular attention																
Remark:	Supplementary information																
Numeric representation:	Binary ... xxxx or xxxxB Decimal ... xxxx Hexadecimal ... xxxxH																
Units for representing powers of 2 (address space or memory space):	K (kilo): $2^{10} = 1,024$ M (mega): $2^{20} = 1,024^2$ G (giga): $2^{30} = 1,024^3$																
Data type:	Word ... 32 bits Halfword ... 16 bits Byte ... 8 bits																

Related Documents

The related documents indicated in this publication may include preliminary versions. However, preliminary versions are not marked as such.

Documents Related to Devices

- Refer to “**User’s Manual Hardware**” of each product about the target device.

Documents Related to Development Tools

Document Name		Document No.
CA850 Ver. 3.00 C Compiler Package	Operation	U17293E
	C Language	U17291E
	Assembly Language	U17292E
	Link Directive	U17294E
PM+ Ver. 6.30 Project Manager		U18416E
ID850 Ver. 3.00 Integrated Debugger	Operation	U17358E
ID850QB Ver. 3.40 Integrated Debugger	Operation	U18604E
RX850 Ver. 3.20 or Later Real-Time OS	Basics	U13430E
	Installation	U17419E
	Technical	U13431E
	Task Debugger	U17420E
RX850 Pro Ver. 3.21 or Later Real-Time OS	Basics	U18165E
	Installation	U17421E
	Technical	U13772E
	Task Debugger	U17422E
RD850 Ver. 3.01 Task Debugger		U13737E
AZ850 Ver. 3.30 System Performance Analyzer		U17423E
SM+ Ver 1.0 System simulator	Operation	U18601E
	User Open Interface	U18212E
RA78K0 Ver.3.80 Assembler Package	Operation	U17199E
	Language	U17198E
	Structured Assembly Language	U17197E
CC78K0 Ver.3.70 C Compiler	Operation	U17201E
	Language	U17200E
ID78K0-QB Ver. 3.00 Integrated Debugger	Operation	U18492E
PM+ Ver. 6.30		U18416E
PG-FP4 Flash Memory Programmer		U15260E

Caution The related documents listed above are subject to change without notice. Be sure to use the latest version of each document when designing.

CONTENTS

CHAPTER 1 PRODUCT OVERVIEW.....	11
1.1 General	11
1.2 Features.....	11
1.2.1 High portability.....	11
1.2.2 Configuration tool	11
1.3 Types of CAN Software Drivers	11
1.4 Execution Environment	12
1.5 Development Environment	14
CHAPTER 2 INSTALLATION	15
2.1 General	15
2.2 Installation Steps	15
2.2.1 Windows startup.....	15
2.2.2 Media setting.....	15
2.3 Directory Structure.....	16
2.3.1 CAN software drivers	16
2.3.2 Documentation	17
2.3.3 Sample programs.....	17
CHAPTER 3 SYSTEM BUILD	18
3.1 Position of CAN Software Drivers	18
3.2 System Building Steps.....	19
3.2.1 File generation by configurator	20
3.2.2 User applications.....	21
3.2.3 Creation of object files.....	23
3.2.4 Creation of load module files	23
CHAPTER 4 CONFIGURATION	24
4.1 General	24
4.2 Management of Input Information by Project File.....	24
4.3 File Creation Steps	25
4.4 Starting the Configurator	25
4.4.1 Device selection	26
4.4.2 Baud rate setting	28
4.4.3 Mask settings	34
4.4.4 Message buffer settings	37
4.4.5 Other settings.....	54
4.4.6 Code generation.....	60
4.4.7 Saving and Opening Project Files	62
4.5 Error/Warning Message List.....	63

CHAPTER 5 DRIVER FUNCTIONS	68
5.1 List of Driver Functions	68
5.1.1 Initialization and setting (6 types)	68
5.1.2 Operation modes (3 types)	68
5.1.3 Buffer data acquisition (4 types)	68
5.1.4 Buffer data setting (4 types).....	68
5.1.5 Transmit/receive confirmation (4 types).....	68
5.1.6 CAN channel status acquisition (3 types)	69
5.2 Data Types	70
5.3 Return Values (Error Codes)	72
5.4 CAN-ID Conversion Macros	73
5.5 Single-Channel Specification CAN Software Driver Functions	74
5.6 CAN Software Driver Functions with Improved Performance	75
5.7 Description of Driver Functions	76
5.8 Driver Functions	78
5.8.1 Initialization and setting	78
5.8.2 Operation modes	87
5.8.3 Buffer data acquisition	92
5.8.4 Buffer data setting	101
5.8.5 Transmit/receive confirmation.....	110
5.8.6 CAN channel status access.....	117
 CHAPTER 6 SAMPLE PROGRAM	 124
6.1 V850ES/FJ2.....	124
6.1.1 Operation environment	124
6.1.2 Overview of operation.....	124
6.1.3 Items preset by configurator	125
6.1.4 Sample program (for NEC Electronics tool)	126
 APPENDIX REVISION HISTORY	 134
APP.1 Main Revisions in this Edition	134
APP.2 Revision History of Preceding Editions.....	136

LIST OF FIGURES

Figure No.	Title	Page
2-1	Directory Structure for CAN Software Drivers.....	16
2-2	Directory Structure of Sample Programs	17
3-1	System Overview.....	18
3-2	System Building Steps.....	19
3-3	Correlations Between Application Program and CAN Software Driver/Configurator.....	22
4-1	Main Screen.....	25
4-2	Device Selection Menu Screen.....	27
4-3	Baud Rate Setting Screen (V850-aFCAN, V850-DCAN, 78K0-aFCAN).....	29
4-4	Baud Rate Setting Screen (V850-FCAN).....	31
4-5	Baud Rate Setting Screen (78K0-DCAN).....	33
4-6	Mask Setting Screen (V850-aFCAN, 78K0-aFCAN).....	34
4-7	Mask Setting Screen (V850-FCAN).....	35
4-8	Mask Setting Screen (V850-DCAN, 78K0-DCAN).....	36
4-9	Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN).....	38
4-10	Message Buffer Setting Screen (V850-FCAN).....	40
4-11	Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN).....	41
4-12	Transmit Message Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN).....	42
4-13	Receive Message Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN).....	44
4-14	Transmit Message Buffer Setting Screen (V850-FCAN).....	46
4-15	Receive Message Buffer Setting Screen (V850-FCAN).....	48
4-16	Transmit Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN).....	50
4-17	Receive Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN).....	52
4-18	Other Settings Screen (V850-aFCAN, 78K0-aFCAN).....	55
4-19	Other Settings Screen (V850-FCAN).....	57
4-20	Other Settings Screen (V850-DCAN, 78K0-DCAN).....	59
4-21	Output Options Setting Screen	60
4-22	Code Generation Startup Screen.....	61
4-23	Screen for Saving and Opening Project Files	62
5-1	Code Format of Driver Functions.....	76

LIST OF TABLES

Table No.	Title	Page
2-1	Provision of CAN Software Drivers	15
4-1	Error Code List.....	63
4-2	Warning Code List.....	66
5-1	Data Type List.....	70
5-2	Parameter Range.....	70
5-3	Macros for Parameters.....	71
5-4	Macros for Error Codes.....	72
5-5	List of CAN-ID Conversion Macros	73
5-6	Single-Channel Specification CAN Software Driver Functions.....	74
5-7	CAN Software Driver Functions with Improved Performance.....	75
5-8	Initialization and Setting	78
5-9	Operation Modes.....	87
5-10	Buffer Data Acquisition.....	92
5-11	Buffer Data Setting.....	101
5-12	Transmit/Receive Confirmation.....	110
5-13	CAN Channel Status Access.....	117

CHAPTER 1 PRODUCT OVERVIEW

1.1 General

CAN software drivers provide application program interface functions (API functions) to implement communications via NEC Electronics' V850 microcontroller of CAN communication function-equipped 32-bit microcontrollers and 78K0 microcontroller of 8-bit microcontrollers.

1.2 Features

1.2.1 High portability

Users can write CAN communication programs without having to know about the CAN's hardware dependencies or the CPU core. This facilitates portability to and customization of the execution environment.

1.2.2 Configuration tool

GUI-driven commands make it easy to set environment-based initial settings for CAN hardware and other devices to be used, as well as static generation of messages.

1.3 Types of CAN Software Drivers

<R>

The CAN software driver is supplied in a source file. The source file can be output with the type of the CAN controller and provision of a parameter check function selected.

Type	Description
Hardware dependency	Different drivers are provided for each CPU core and CAN controller type.
Parameter checking function	<p>A function that checks and recognizes a user-specified parameter as an error in CAN software driver functions, or a function that does not can be selected.</p> <p>The parameter check function checks errors in CAN software driver functions. Therefore, illegal parameter specification can be prevented but the code capacity increases. Using this function for debugging and evaluation is thus recommended.</p> <p>Since this parameter checking function performs an error check within the CAN software driver functions, there is a trade-off between parameter checking to prevent errors and increased code size. Accordingly, use of the parameter checking function is recommended only for debugging or evaluations.</p>

1.4 Execution Environment

CAN software drivers operate on target systems that are equipped with the following hardware.

<R> **(1) Target CPU**

V850 Microcontrollers	V850ES/FE2, V850ES/FF2, V850ES/FG2, V850ES/FJ2, V850ES/FE3, V850ES/FF3, V850ES/FG3, V850ES/FJ3, V850ES/FK3, V850ES/SG2, V850ES/SJ2, V850ES/SG3, V850ES/SJ3, V850E/RS1, V850E/RS2, V850E/PG2, V850E/DJ3, V850E/DL3, V850E/IA1
78K0 Microcontrollers	78K0/FC2, 78K0/FE2, 78K0/FF2, μ PD780822B

(2) Target CAN controller

aFCAN
DCAN
FCAN

(3) Memory capacity

The memory capacity varies depending on the number of functions (APIs) used by the user, the number of message buffers and channels, the CAN controller, and the register mode used. It particularly depends on the number of functions (APIs).

<R> The memory capacities of the total functions (16 functions) and the functions used in CHAPTER 6 SAMPLE PROGRAM (8 functions) are shown below.

(a) Memory capacity of total functions

- Compiler : CA850 Ver. 3.00 (made by NEC Electronics)
- Optimization : Size optimization
- CAN controller : aFCAN
- Register mode : 32 register mode

<R> • Number of functions used (APIs) : 16

- Table used : Number of channels : 2

Number of transmit/receive message buffers : 10 for each channel
(20 in total)

<R>

	ROM Capacity	RAM Capacity
16 functions (APIs)	Approx. 1.46 KB	8 bytes max.
Table	272 bytes	-
Total	Approx. 1.73 KB	

Remark It depends for the abovementioned contents on the Version of a compiler.

The RAM capacity is the capacity for the stack area.

(b) Memory capacity of functions used in sample program

- Compiler : CA850 Ver. 3.00 (made by NEC Electronics)
- Optimization : Size optimization
- CAN controller : aFCAN
- Register mode : 32 register mode
- Number of functions used (APIs) : 8
- Table used : Number of channels : 1
Number of transmit/receive message buffers: 2

<R>

<R>

	ROM Capacity	RAM Capacity
8 functions (APIs)	Approx. 1.01 KB	8 bytes max.
Table	116 bytes	-
Total	Approx. 1.13 KB	

Remark It depends for the abovementioned contents on the Version of a compiler.

The RAM capacity is the capacity for the stack area.

<R> 1.5 Development Environment

The following environment is required for use of CAN software drivers for development of application systems.

(1) Hardware

Host machine

- IBM PC/AT™ Series:

Supported Windows™ versions Windows 98, Windows Me, Windows 2000, Windows XP

(2) Software

Compiler package

- CA850: Made by NEC Electronics
- CCV850: Made by Green Hills Software, Inc.
- CC78K0: Made by NEC Electronics

(3) Debuggers

- ID850: Made by NEC Electronics
- Multi™: Made by Green Hills Software, Inc.
- PARTNER: Made by Kyoto Microcomputer
- ID78K0: Made by NEC Electronics
- ID78K0-NS: Made by NEC Electronics

(4) Simulators

- SM850: Made by NEC Electronics
- SM78K0: Made by NEC Electronics

CHAPTER 2 INSTALLATION

2.1 General

CAN software drivers are provided on Windows-based media. CAN software driver functions are provided in source files.

<R>

Table 2-1. Provision of CAN Software Drivers

Format	Description	Media
Full installation	<ul style="list-style-type: none">• Sample programs• CAN configurator/CAN software drivers (CAN configurator, driver source file, header files, device database)• Configurator• Documentation (this document)	CD-ROM
Database	<ul style="list-style-type: none">• Sample programs	Web (to be provided), floppy disk

<R>

2.2 Installation Steps

The following are the steps for installing to the host machine the set of files included in the media on which the CAN software drivers are provided.

2.2.1 Windows startup

Turn on the host machine and peripheral devices, then start Windows.

2.2.2 Media setting

Set the CD containing the CAN software drivers into the target device (CD-ROM drive). The setup program will open automatically.

The installer generally outputs the following three files.

<R>

- CAN configurator/CAN software driver
- Documentation
- Sample programs

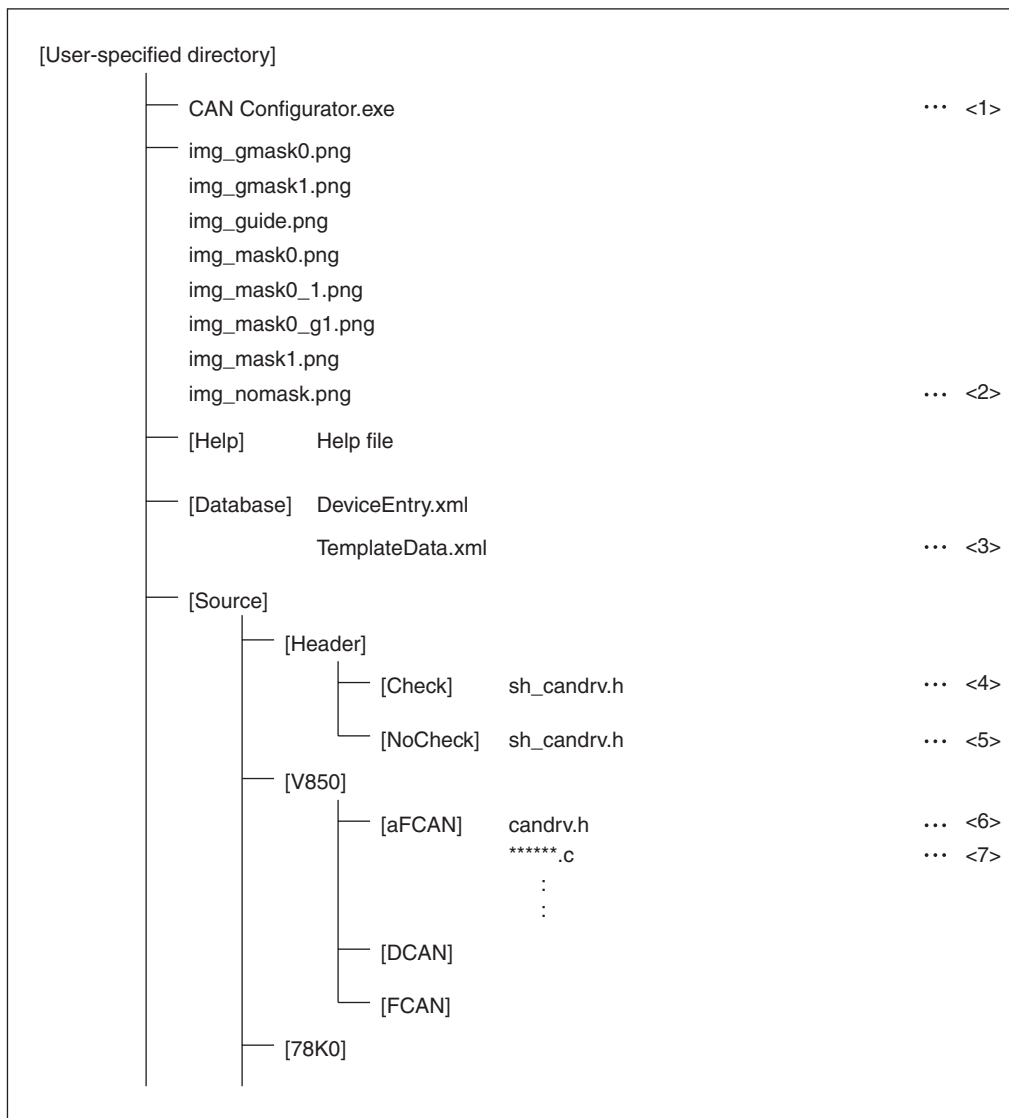
Install the above files in accordance with the message displayed on the screen.

2.3 Directory Structure

2.3.1 CAN software drivers

<R>

Figure 2-1. Directory Structure for CAN Software Drivers



<1> : Configurator main unit

<2> : Configurator image file

<3> : Version information file

<4> : Header file with parameter checking

<5> : Header file without parameter checking

<6> : CAN software driver header file

<7> : CAN software driver main unit source file

2.3.2 Documentation

Copy this document to any directory.

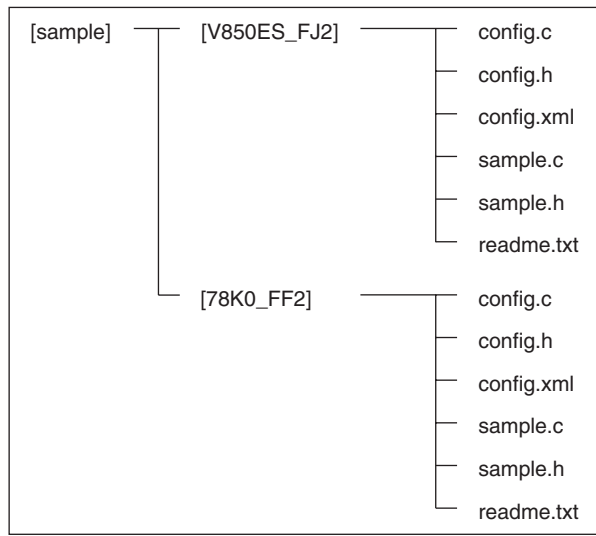
2.3.3 Sample programs

<R> Sample programs are provided for V850ES/FJ2 and 78K0/FF2. These programs are installed to the directory structure shown in **Figure 2-2**, under the user-specified directory.

See the **readme.txt** file under each directory for description of each sample program's operations.

<R>

Figure 2-2. Directory Structure of Sample Programs



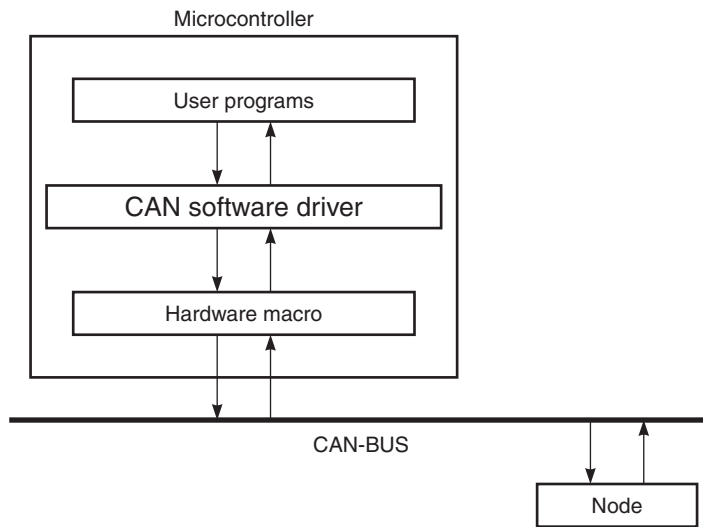
CHAPTER 3 SYSTEM BUILD

3.1 Position of CAN Software Drivers

In the system, CAN software drivers are positioned between the user application and hardware (see **Figure 3-1**). A user interface is provided for controlling the hardware.

The user can simply describe CAN software driver functions in the application without having to know about controlling the hardware registers.

Figure 3-1. System Overview



3.2 System Building Steps

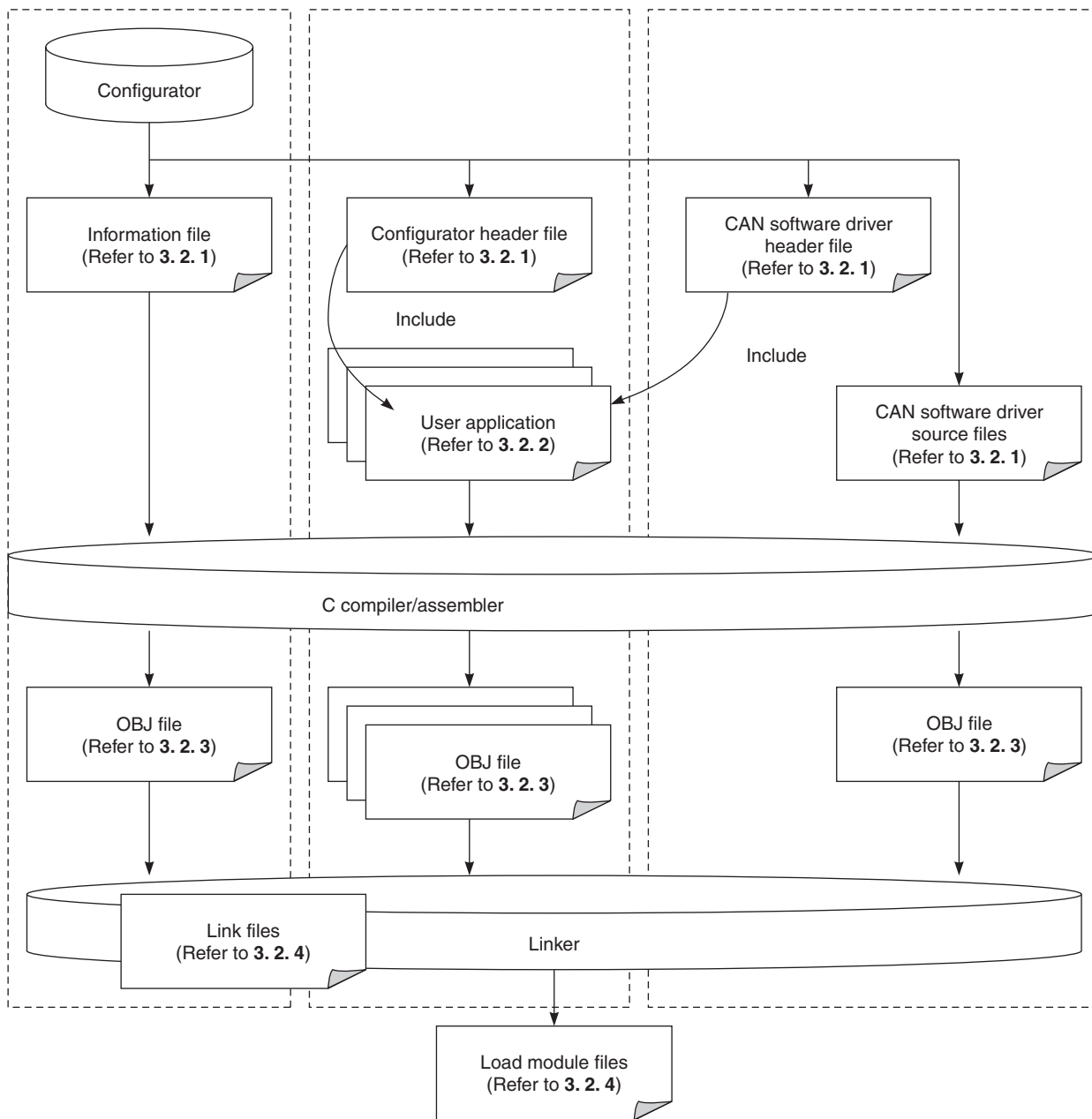
When building a system, the set of files that have been installed from the CAN software driver media to the user-specified environment (host machine) are used to generate load module files.

The following terms are used in these descriptions.

- OBJ file: Object file
- LINK file: Link directive file

The system building steps are illustrated in the following figure.

Figure 3-2. System Building Steps



3.2.1 File generation by configurator

The configurator sets initial values such as the CAN baud rate, message buffer allocation for transmission and reception, interrupts, and mask, and information files and header files.

For details of the configurator's setting steps, see **CHAPTER 4 CONFIGURATION**.

(1) Configurator's input data

The following must be determined as the minimum requirement before using the configurator to generate files.

- Device to be used (microcontroller name and device name)
Example: V850ES/FJ2 microcontroller, μ PD70F3239
- Device's system clock
- Channel to be used
- CAN system clock for each module
- Baud rate and data bit time configuration for each module
- Setting of message buffer per module

Allocation of transmit and receive buffers, CAN-ID, standard/extended frame, interrupt reporting ON/OFF, mask settings, etc.

- Interrupt enable/disable setting per module
Error reporting and transmit/receive reporting

(2) Configurator's output data

The following four types of files are output by the configurator.

- Information file
This file contains a table of information that was entered into the CAN device by the user. This file is referenced by the CAN software driver. Therefore, this file must be compiled/assembled and linked together with the user application program.
- Configurator header file
Message names set by the user are defined in macros. The user can employ these macro names as arguments, etc., for CAN software driver functions. Consequently, this file must be included in the user application.
- CAN software driver source files
A group of CAN software driver source files applicable to the device to be used is output. These files must be compiled, assembled, and linked, along with the user application program.
- CAN software driver header file (candrv.h)
A header file that will be used in combination with the target CAN software driver's functions is output.

3.2.2 User applications

CAN's API functions are used to create CAN communication applications. The header file that is created by the configurator must be included in any file that uses a driver function.

The user should make the following settings in order to use the CAN hardware and the CAN software driver functions.

(1) to (4) below are initialization routines, while (5) and (6) are code related to CAN control applications.

(1) System settings

The items to be set vary according to the target device. For example, the setting registers include the following.

For V850 microcontrollers

- System wait control register (VSWC)
- PLL control register (PLLCTL)

For 78K0 microcontrollers

- System wait control register (VSWC)
- Internal memory size switching register (IMS)
- Internal expansion RAM size switching register (IXS)

(2) Port settings

CAN transmit/receive modes are specified for pins assigned to the transmit/receive operations of the CAN controller to be used.

(3) BPC settings (not required for some devices)

These set the programmable peripheral I/O register area, which is the area allocated for the peripheral I/O registers used by the CAN controller.

(4) Interrupt settings

When the interrupt handler is used by the CAN controller, the destination address for the interrupt handler routine must be described.

(5) Inclusion of header files

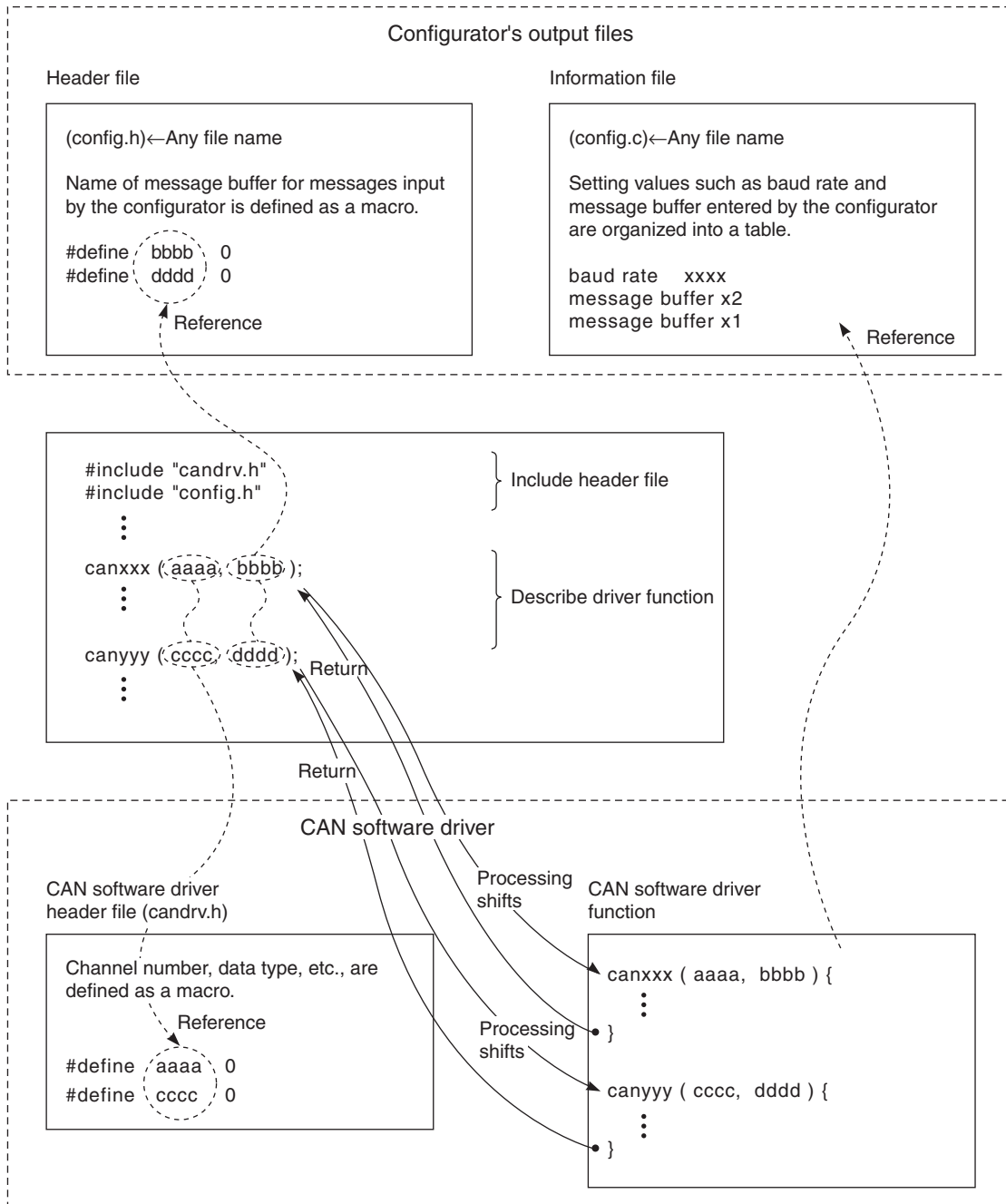
The following two header files are included in the source file that uses a CAN software driver.

- candrv.h: Header file provided with the CAN software driver
- *****.h: Header file generated by the configurator (***** is determined by the user)

(6) Coding of CAN software driver function (API)

This includes the control code for the CAN that uses the driver's API functions in an application program. For description of the CAN software driver's API functions, see **CHAPTER 5 DRIVER FUNCTIONS**.

Figure 3-3. Correlations Between Application Program and CAN Software Driver/Configurator



3.2.3 Creation of object files

Information files and CAN software driver source files are compiled and assembled by the user application and CAN configurator to create relocatable object files.

Remark See user's manual of each tool for details of the C compiler/assembler startup options and execution method.

3.2.4 Creation of load module files

The following files are linked to create load module files.

- Object file with compiled/assembled user application
- Object file with compiled information files (generated by the CAN configurator)
- Object files resulting from compiling CAN software driver source files created by the CAN configurator
- Link directive file
- Library files recommended for C compiler package

Remark See user's manual of each tool for details of the link editor startup options and execution method. Upon procedure is the example of the compiler made from the NEC electronics, when use other tools, refer to the user's manual of each tool for it.

CHAPTER 4 CONFIGURATION

4.1 General

The configurator is a development tool that the user employs to set the CAN's initial values when building a system that includes CAN functions. Functions are provided to enable the user to enter initial values for registers in accordance with the device to be used, and to perform static generation of messages used in the system.

The initial values for registers corresponding to the target device can be set while selecting the device, entering the system clock value, and setting the baud rate per channel. In addition to these clock and baud rate settings, interrupts can be set as enabled or disabled and message buffer settings can be entered.

In the separate window used to set the baud rate, numerical values that can be set are calculated automatically based on the values input by the user and are displayed in a table. When the user selects any setting from the table, the bar graph display changes to indicate the selected setting. This enables the user to graphically select and set baud rate-related values.

As for static generation of messages, assignment to transmit and receive messages, message names, CAN-ID, and interrupt enable/disable are all settings that can be made.

Remark See the CAN controller chapter in the document of each device for details of the CAN controller functions.

4.2 Management of Input Information by Project File

The configurator uses project files to store and manage various types of information entered by the user. This use of project files for storing and retrieving information enables generation of the header files and information files that can be used by driver functions any number of times.

4.3 File Creation Steps

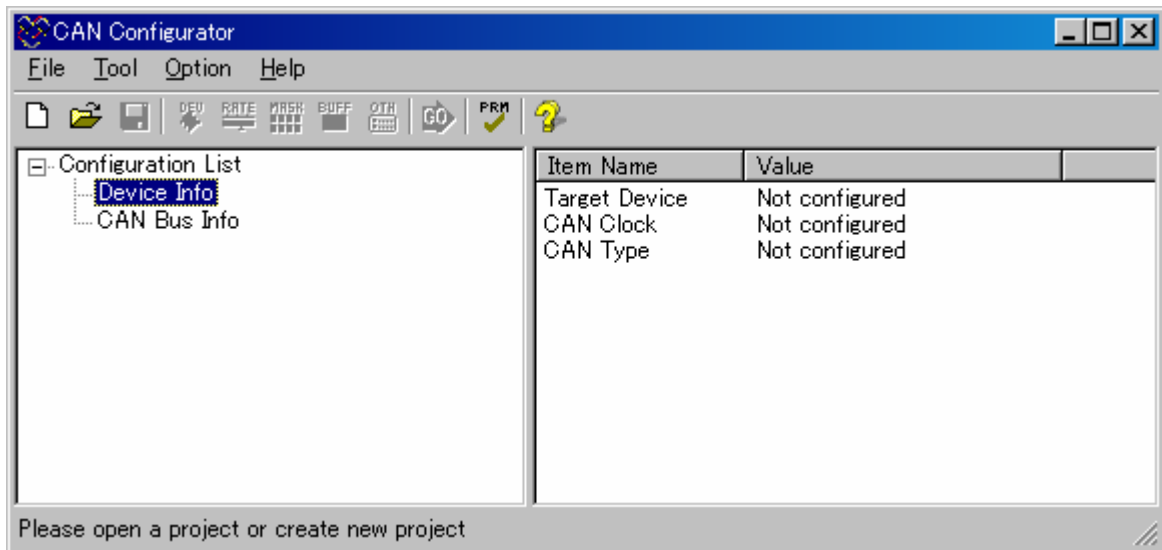
Once the device to be used has been selected, the other steps can be performed in any order. However, it is generally recommended to follow the steps described below in **4.4 Starting the Configurator**.

4.4 Starting the Configurator

When CANConfigurator.exe is started, the following main screen is displayed. Main items are listed on the left side of this screen, and brief descriptions of settings are shown on the right side. "Not configured" is displayed in the Value field for items that have not been set.

<R>

Figure 4-1. Main Screen



4.4.1 Device selection

Select device information from the device selection menu.

<Startup method>

- Start by selecting [File] in [New] menu
- Start by double-clicking "Device Information" in the main window.
- To change settings later on, start by selecting [Device Setup] on [Tool] menu.

<Settings>

- Select device microcontroller name and target device

Once the user has selected a device, the device's CAN-related information is displayed in the "Device Information" field. At that time, the number of CAN channels is changed to a number corresponding to the specified device.

<R>

- Input of CAN clock

Input the frequency of the clock to be supplied to the CAN modules.



The button appears if a device with which the CAN clock can be specified for each channel is selected. Clicking this button displays another dialog box for inputting multiple clock frequencies.

- Setting of programmable I/O area

Select the I/O area from the pull-down menu. Any I/O area that is not listed in the pull-down menu can be entered directly.

Selection or input may not be possible (the area is fixed) in some devices.

- Select CAN channel to be used

Select the check box next to the CAN channel to be used.

<R>

Figure 4-2. Device Selection Menu Screen

Device Selection

Series Name: V850ES/FJ2

Device Name: uPD70F3237
uPD70F3238
uPD70F3239

Device Information

CAN Macro Type : aFCAN
Channel : 4 ch
Buffer :
Channel1: 32 (buffers)
Channel2: 32 (buffers)
Channel3: 32 (buffers)
Channel4: 32 (buffers)
Programmable I/O Area : (higher 6 bits are don't-care-bit)
Start address (CAN register area):
03FEC000, 07FEC000, 0BFEC000, 0FFEC000, FFFEC000

CAN Clock: 16 MHz

CAN register area: 03FEC000

Channel Select

Channel1 Channel2 Channel3 Channel4
 Channel5 Channel6

OK Cancel

4.4.2 Baud rate setting

Select the baud rate for each channel.

<Startup method>


- Start by selecting [Baud Rate Setup] on [Tool] menu

<Settings> (V850-aFCAN, V850-DCAN, 78K0-aFCAN)

- Select CAN module system clock

Select a system clock value from the pull-down menu. The baud rate values that can be selected are determined according to the module system clock setting.


- Select baud rate

From the pull-down menu, select a baud rate value that corresponds to the CAN bus conditions to be used. The pull-down menu lists the baud rates that are used often. If the desired baud rate value is not listed, enter it directly. After selecting or entering a baud rate value, click the  button to automatically recalculate.

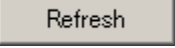
- Data bit time setting

The bit time is calculated automatically by the configurator. Select the corresponding bit time combination from the list. For the standard of a set up, samplepoint is just over or below 75%, and SJW is as large as possible(maximum is 4)

Clicking on the field for an item (such as Prescaler, DBT, SPT, Sample Point, or SJW) activates the sort function. The bit time that is centered on the sample point (red triangle) is displayed at the bottom of this list.

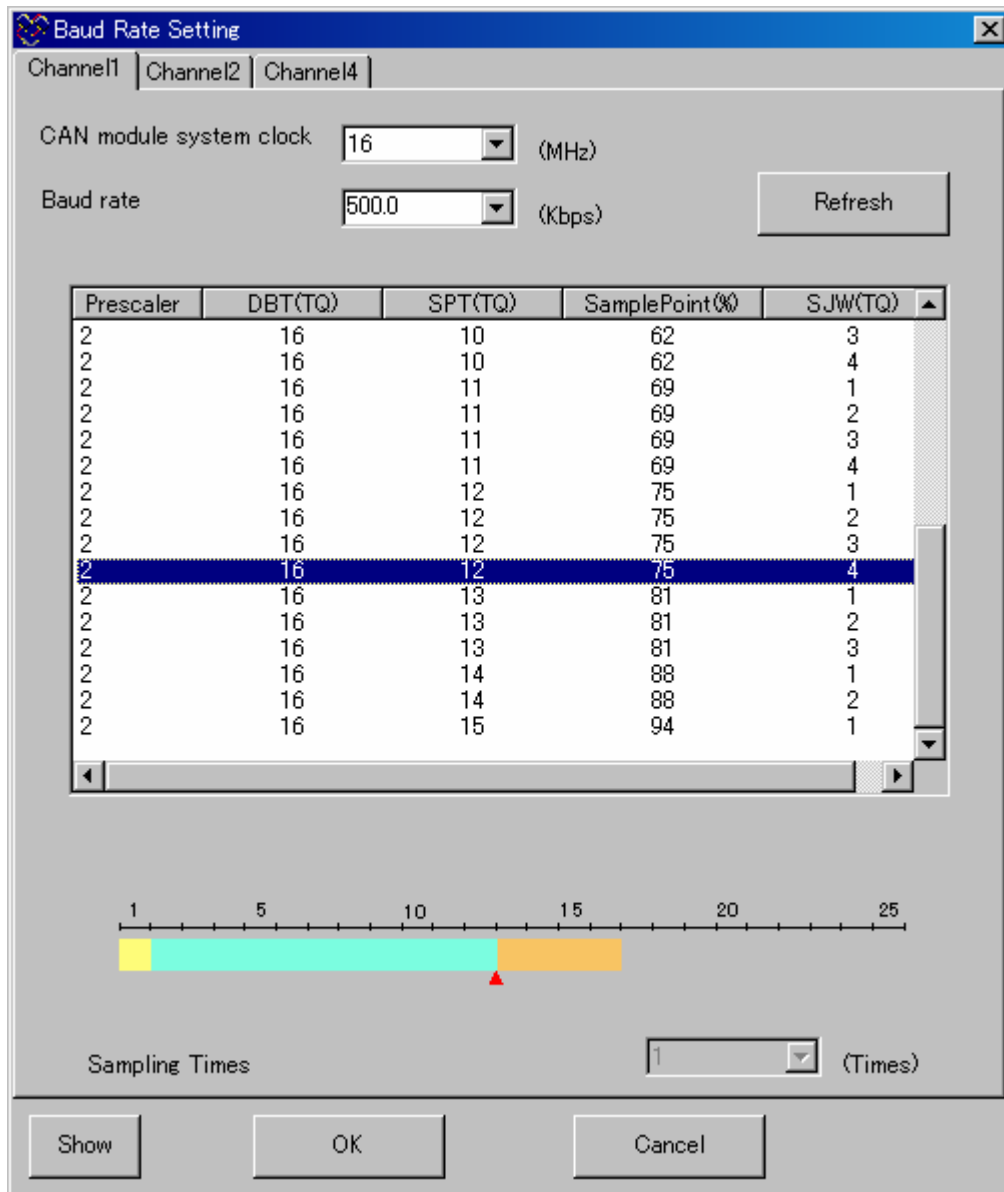
Click the  button to display values set to registers.

- Initialization of settings

Click the  button to deselect any previously selected data bit time and to reset the sort function to its default setting.

<R>

Figure 4-3. Baud Rate Setting Screen (V850-aFCAN, V850-DCAN, 78K0-aFCAN)



<Settings> (V850-FCAN)

- Select CAN global timer clock


Select a CAN global timer clock from the pull-down menu. The baud rate values that can be selected are determined according to the global timer clock setting. This setting is common to all the channels.

- Set time stamp clock

Input a time stamp clock. Select the unit by using  button. Pressing the

 button after inputting a clock automatically executes calculation.

- Select baud rate


From the pull-down menu, select a baud rate value that corresponds to the CAN bus conditions to be used. The pull-down menu lists the baud rates that are used often. If the desired baud rate value is not listed, enter it directly. After selecting or entering a baud rate value, click the  to automatically recalculate.

- Set data bit time

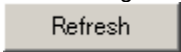
The bit time is calculated automatically by the configurator. Select the corresponding bit time combination from the list.

Clicking on the field for an item (such as Prescaler, DBT, SPT, Sample Point, or SJW) activates the sort function. The bit time that is centered on the sample point (red triangle) is displayed at the bottom of this list.

By clicking the  button, more detailed setting can be made by using the TL mode.

Click the  button to display values set to registers.

- Initialization of settings

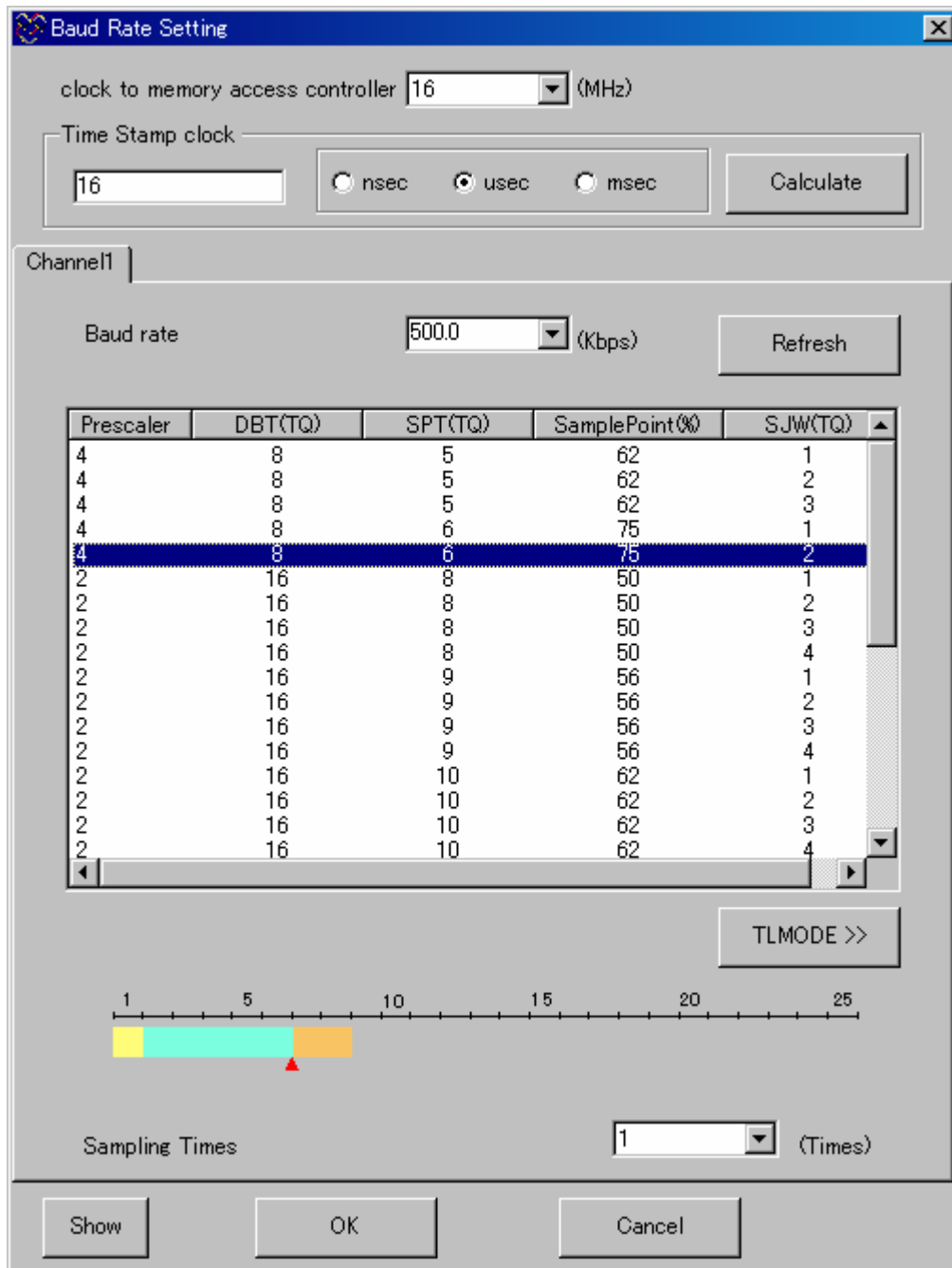
Click the  button to deselect any previously selected data bit time and to reset the sort function to its default setting.

- Select sampling count

Select a sampling count from the pull-down menu. Select whether sampling is executed once at a sampling point or received data is sampled three times and is determined by majority.

<R>

Figure 4-4. Baud Rate Setting Screen (V850-FCAN)



<Settings> (78K0-DCAN)

• Select CAN module system clock

Select a system clock value from the pull-down menu. The baud rate values that can be selected are determined according to the module system clock setting.

To use an external clock, check Use the External CAN clock and directly input the external clock frequency as the module system clock.

• Select baud rate

From the pull-down menu, select a baud rate value that corresponds to the CAN bus conditions to be used. The pull-down menu lists the baud rates that are used often. If the desired baud rate value is not listed, enter it directly. After selecting or entering a baud rate value, click the Refresh button to automatically recalculate.

• Set data bit time

The bit time is calculated automatically by the configurator. Select the corresponding bit time combination from the list.

Clicking on the field for an item (such as Prescaler, DBT, SPT, Sample Point, or SJW) activates the sort function. The bit time that is centered on the sample point (red triangle) is displayed at the bottom of this list.

By clicking the TLMODE >> button, more detailed setting can be made by using the TL mode.

Click the Show button to display values set to registers.

• Initialization of settings

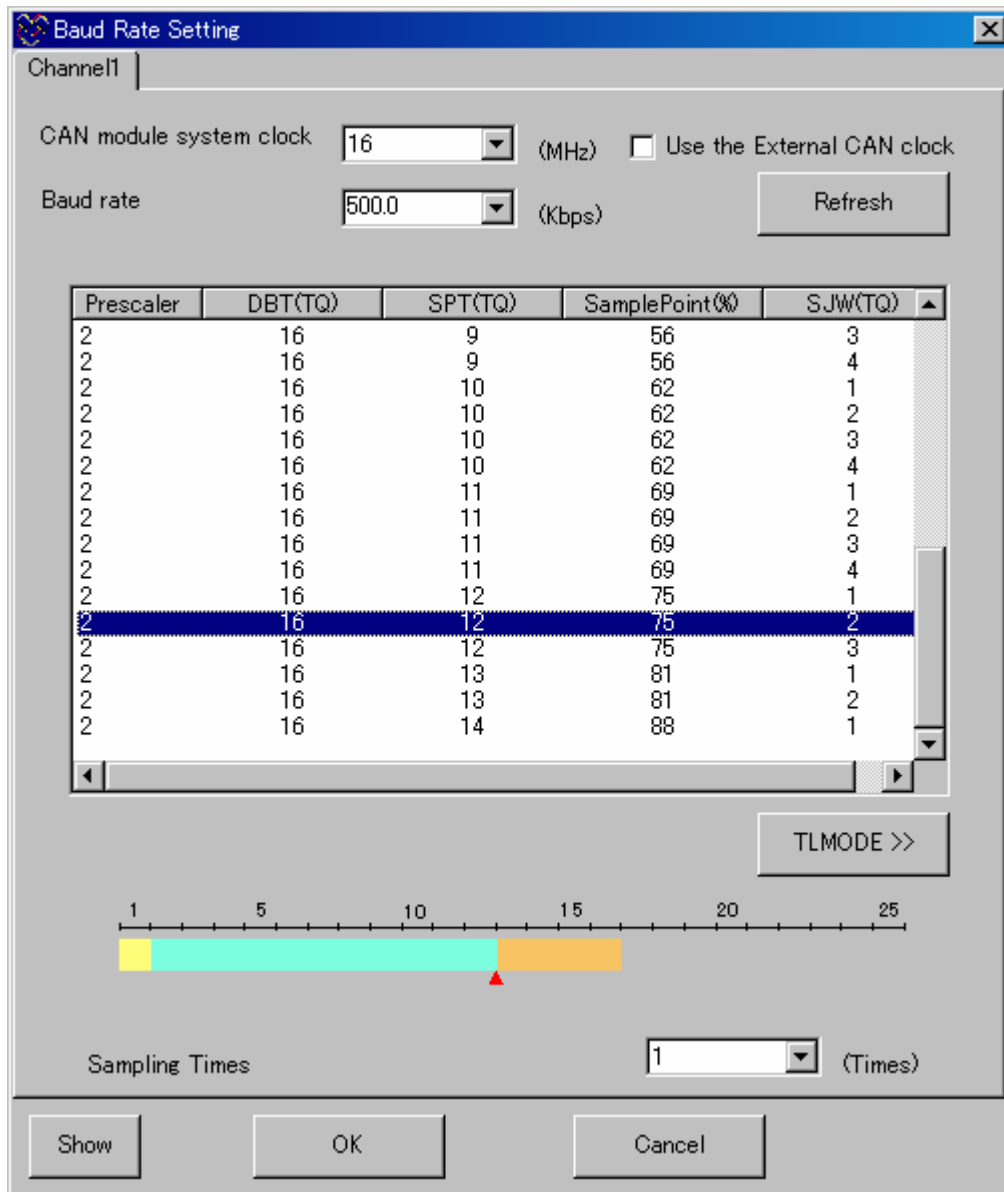
Click the Refresh button to deselect any previously selected data bit time and to reset the sort function to its default setting.

• Select sampling count

Select a sampling count from the pull-down menu. Select whether sampling is executed once at a sampling point or received data is sampled three times and is determined by majority.

<R>

Figure 4-5. Baud Rate Setting Screen (78K0-DCAN)



4.4.3 Mask settings

Mask settings can be entered for each channel.

Links between receive message buffers and masks are shown in the message buffer settings.

<Startup method>

- Start by selecting [Mask Setup] on [Tool] menu

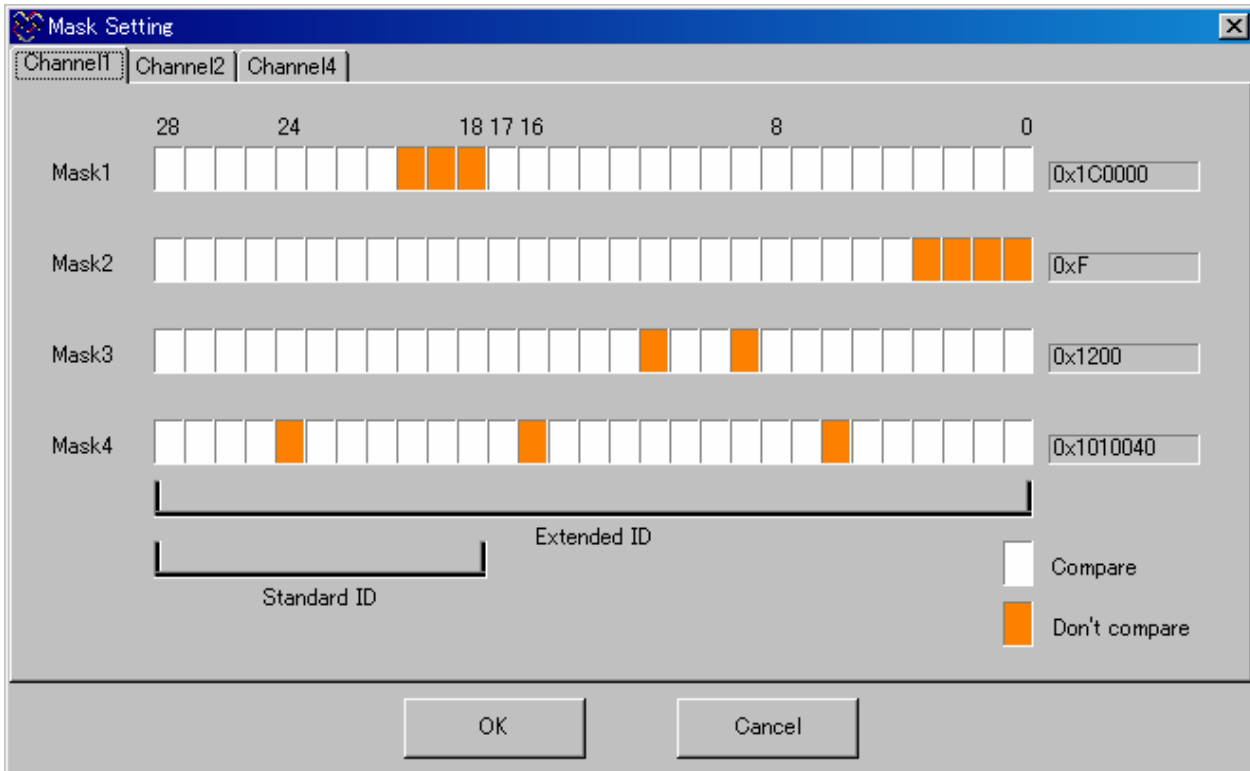
<Settings> (V850-aFCAN, 78K0-aFCAN)

- Mask settings

A bit image of the mask register is displayed. Set bits in order to avoid comparison between (i.e., to mask) the message buffer ID of a received message and the message buffer's own ID.

Remark The mask function is useful for managing CAN-IDs in group units because multiple CAN-IDs can be stored in one message buffer.

Figure 4-6. Mask Setting Screen (V850-aFCAN, 78K0-aFCAN)



<Settings> (V850-FCAN)

- Set mask

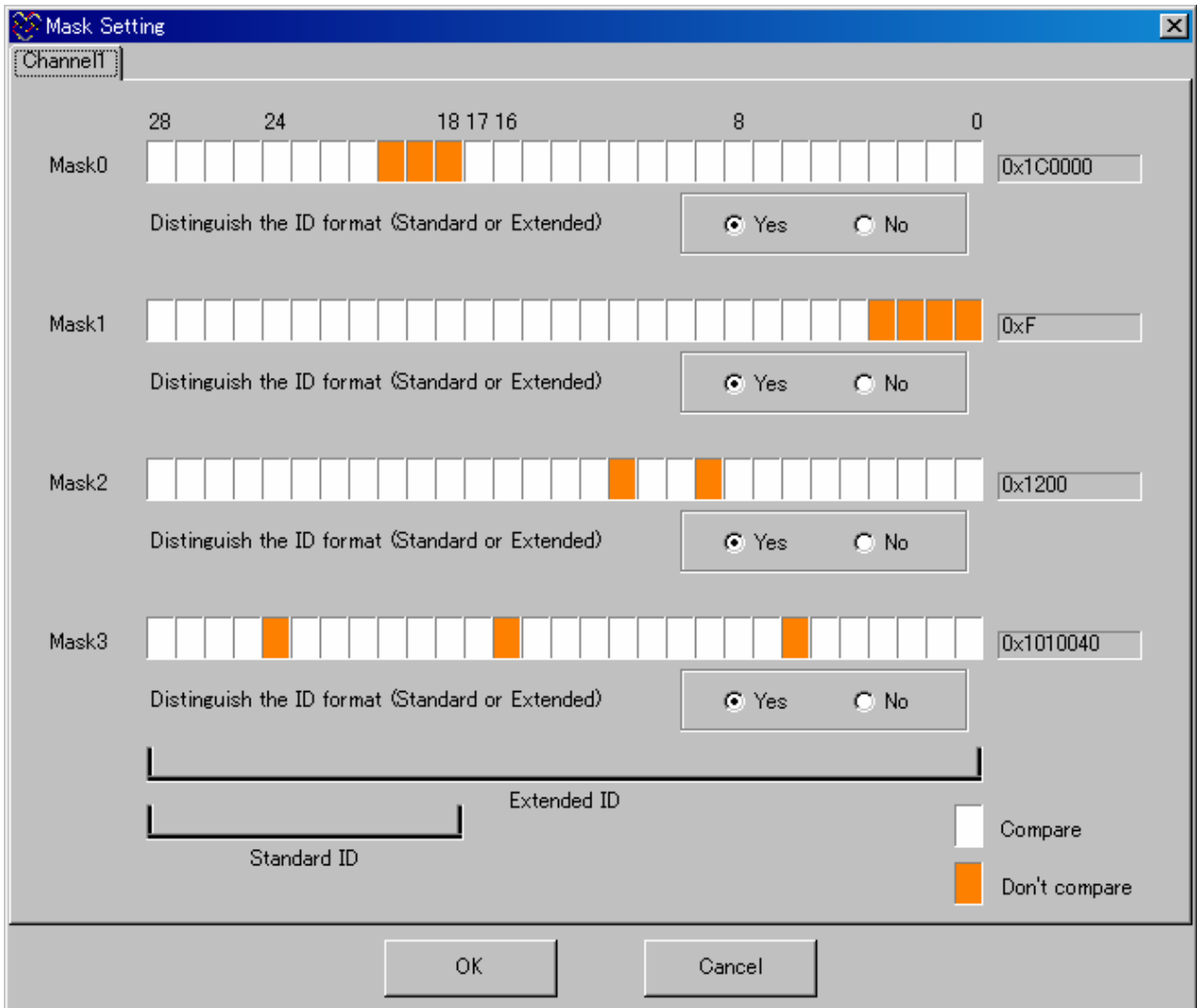
The mask register is displayed in bit image.

If the ID of a received message buffer and the ID of the message buffer are not compared (if comparison is to be masked), set the corresponding bit.

- Mask identifier (ID) format

Use the Yse No button to select whether ID format (standard or expansive) is checked or not for each mask register.

Figure 4-7. Mask Setting Screen (V850-FCAN)



<Settings> (V850-DCAN, 78K0-DCAN)

- Set mask

The mask register is displayed in bit image.

If the ID of a received message buffer and the ID of the message buffer are not compared (if comparison is to be masked), set the corresponding bit.

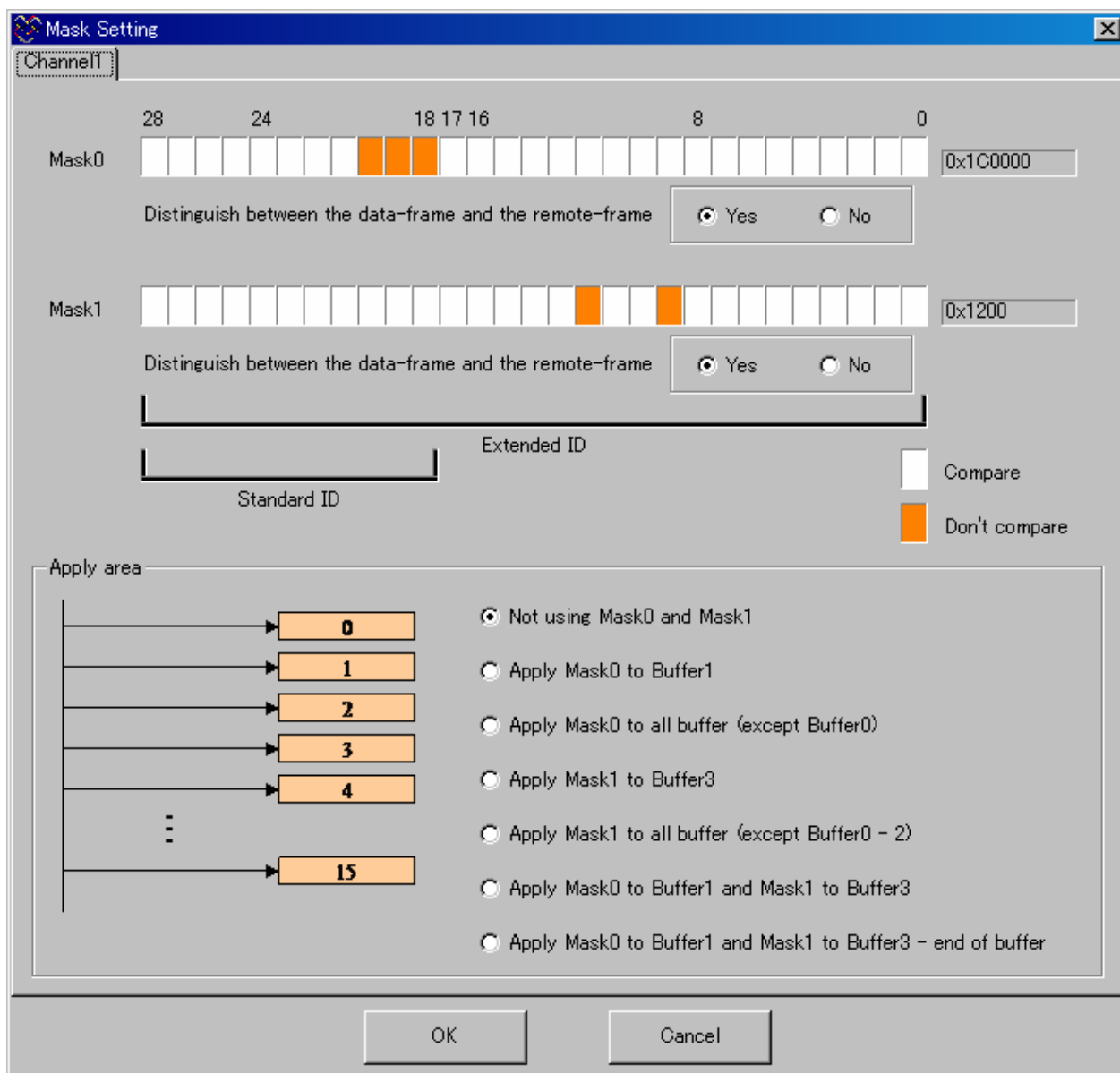
- Mask frame format

Use the Yes No button to select whether the frame format (data frame or remote frame) is checked for each mask register.

- Select global mask function

Select whether the mask register is to be used, and select a global mask. When the button is clicked, the image of the mask is displayed at the lower left part.

Figure 4-8. Mask Setting Screen (V850-DCAN, 78K0-DCAN)



4.4.4 Message buffer settings

The message buffers for each channel are assigned as transmit or receive buffer, and the ID, frame type, and other settings are entered for each message.


(1) Assignment of message buffers

<Startup method>

- Start by selecting [Message Buffer Setup] on [Tool] menu

<Settings> (V850-aFCAN, 78K0-aFCAN)

- Assignment of unused message buffers to transmit or receive function

Click the  button to move message buffers from the No Used Buffers list field to either the Tx Message or Rx Message list.

- Set the automatic block transmission (ABT) function ^{Note} as either enabled or disabled.

To use the ABT function, select the radio button next to "Use". This enables a guard to be set so that message buffers 0 to 7 cannot be used as transmit message buffers.

Note The automatic block transmission (ABT) function is used to transmit multiple data frames continuously without using the CPU. The number of transmit message buffers assigned to ABT is fixed to 8, from message buffer 0 to 7.

Note, however, that the ABT mode is not included in the CAN software driver function, so users must code it by themselves.


- Set mask function

The mask setting function described above in **4. 4. 3 Mask settings** can also be set by pressing the

 button.

<Other functions>

- How to move an assigned message buffer back to the No Used Buffers list field

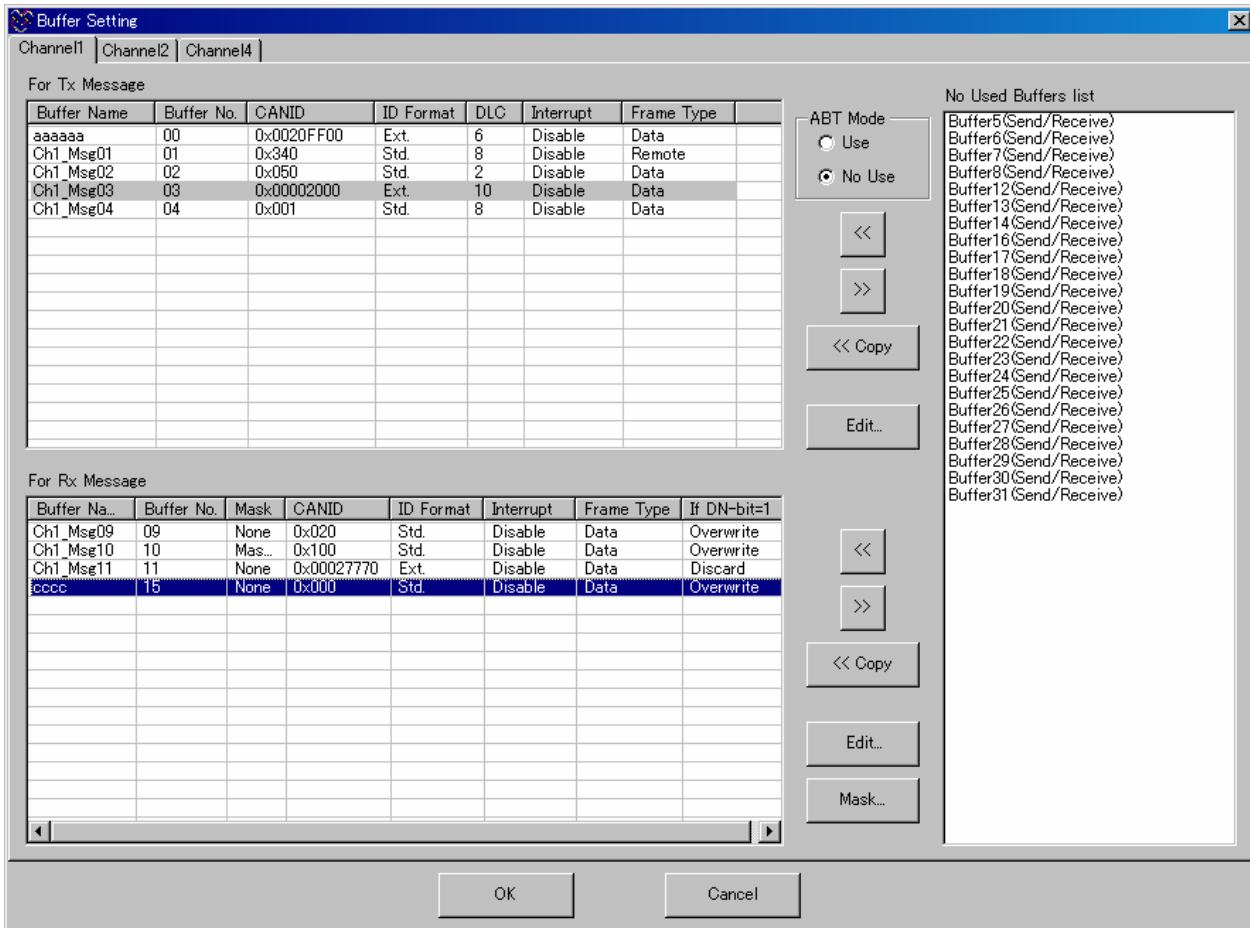
Select a message from the Tx (or Rx) list and click the  button to move the message to the No Used Buffers list field.

- Message copy function

To use message buffer settings that have already been set for a message buffer in the No Used Buffers list field, select the message in the No Used Buffers list field and the message to be copied, then click the


 button.

Figure 4-9. Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN)



<Settings> (V850-FCAN)

- Specify use of unused message buffers as transmit/receive message buffers

Use the  button to move unused message buffers from the list (List of Unused Buffers) to the list of transmit message (Tx Message) buffers or to the list of receive message (Rx Message) buffers.

- Select priority control for transmission

Select priority control by identifiers (ID) or by message numbers, by using the radio button.


- Select overwrite mode

To receive a new message to a message buffer that has already received a message, whether the old message is overwritten or the new message is discarded without the old message overwritten must be selected. When Overwrite (default) is selected, the old message is overwritten.

If Discard is selected, the new message is not overwritten to the message buffer that has already received a message^{Note} but is discarded.


Note Message buffer whose DN bit is set to “1”

- Setting mask function

The mask setting function in **4. 4. 3 Mask settings** can also be used by pressing the  button.

<Other functions>

- Return message buffers to unused status

If a message is selected from the Tx (Rx) list and the  button is pressed, the selected message returns to the unused message buffer list.

- Copy message

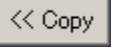
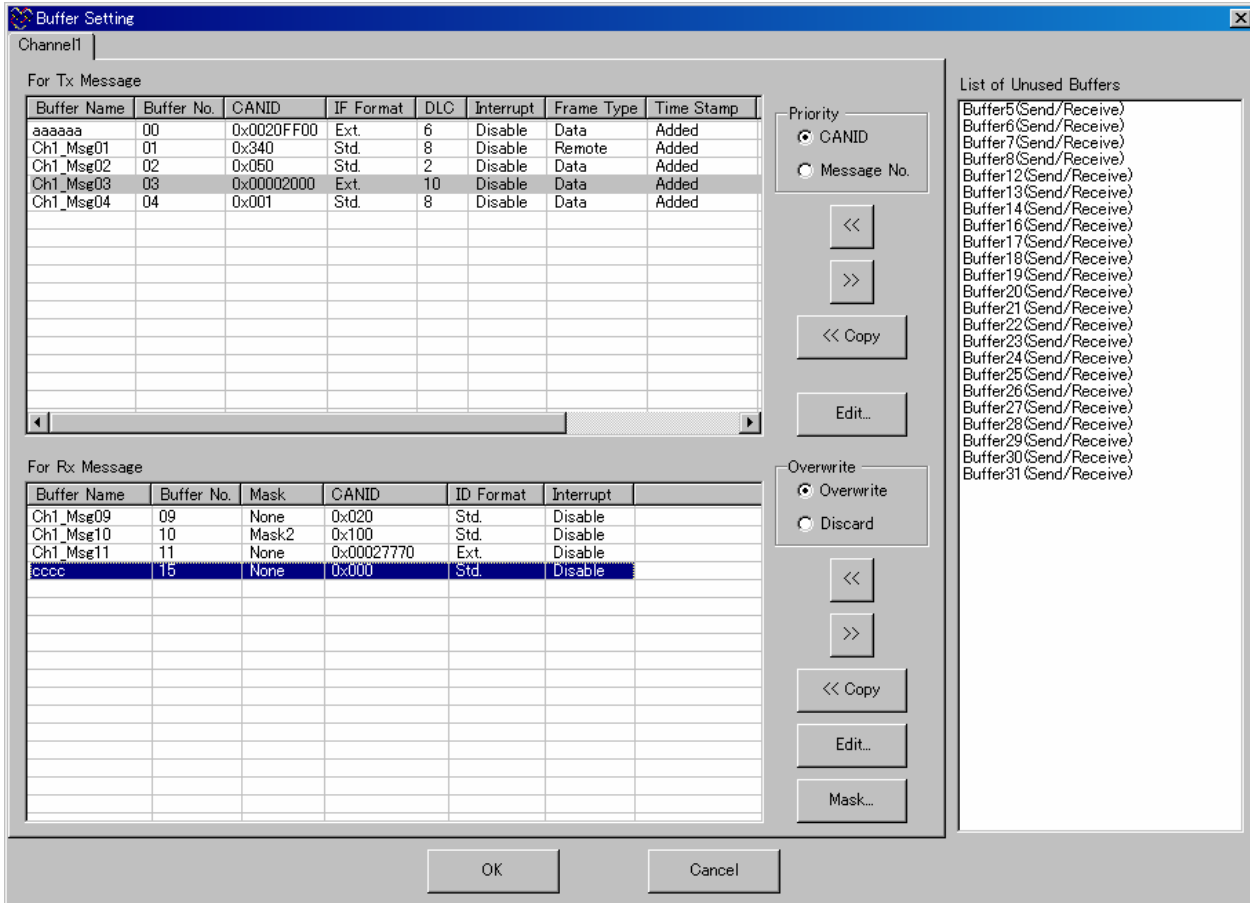
To use a message buffer in the unused message buffer list with the same contents of a message buffer that has already been set, select a message from the unused message buffer list and a message to be copied, and press the  button.

Figure 4-10. Message Buffer Setting Screen (V850-FCAN)



<Settings> (V850-DCAN, 78K0-DCAN)

- Select priority control for transmission

Use the radio button to select transmit buffer 0 or 1 as the priority transmission message buffer. The number of transmit message buffers is fixed to two.

- Register the number of receive message buffers used

Directly input the number of message buffers to be used (using buffer number) or select it from the pull-down menu and press the **Refresh** button to register the number of message buffers to the list of receive message (Rx Message) buffers.

- Set mask function

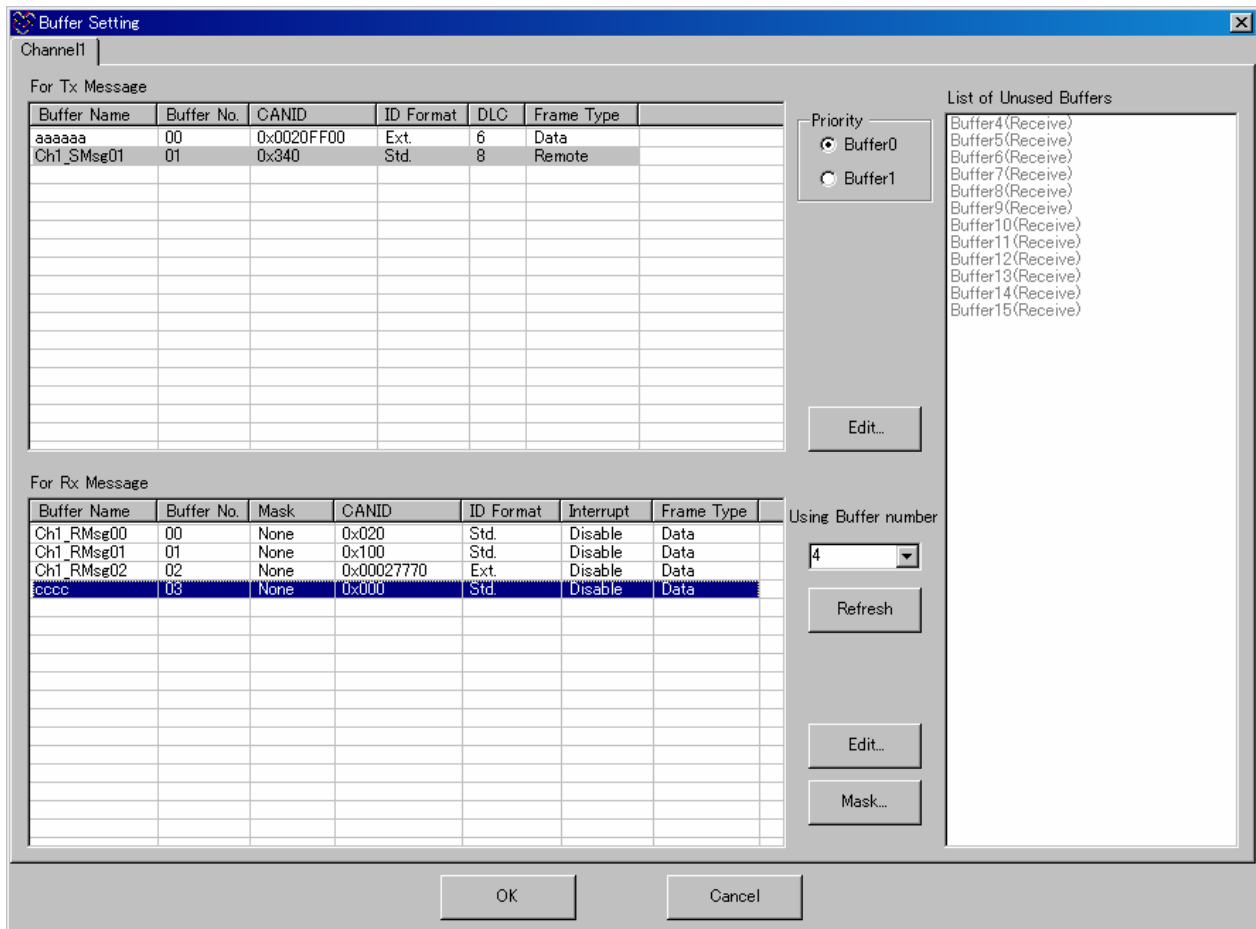
The mask setting function in **4. 4. 3 Mask setting** can also be used by pressing the **Mask** button.

<Other functions>

- Return the registered message buffer to the unused status

If the number of receive message buffers to be used (using buffer number) is decreased and the **Refresh** button is pressed, the message returns to the list of unused message buffers again.

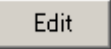
Figure 4-11. Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN)



(2) Message buffer settings

<Startup method>

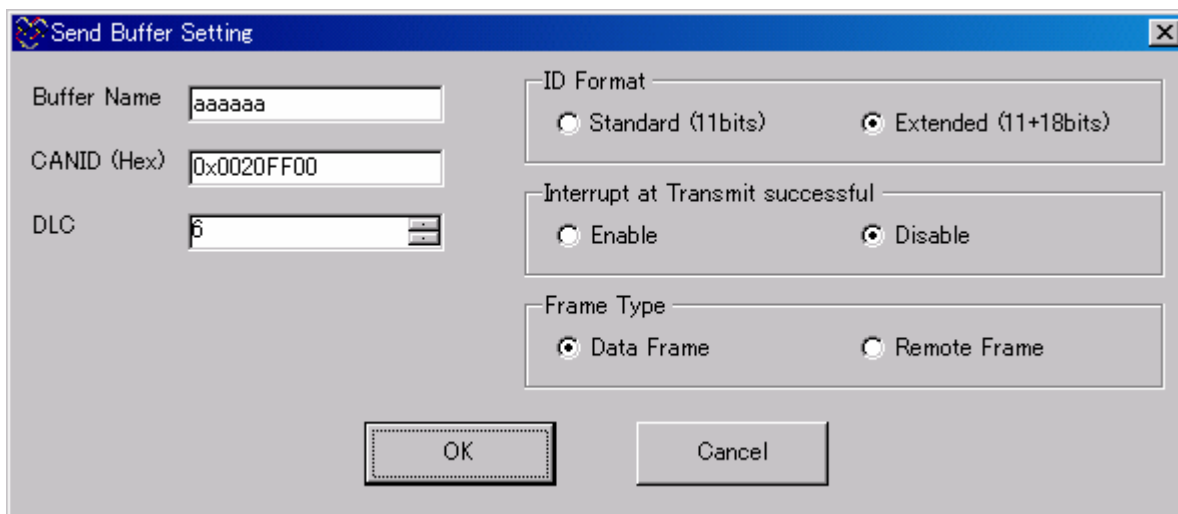
Start from the screen for assigning message buffers, as described in (1) above.

- After selecting a message buffer, click the  button.
- Double-click a message buffer.

<Settings> (V850-aFCAN, 78K0-aFCAN)

(a) Transmit message buffer setting items

Figure 4-12. Transmit Message Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN)



The screenshot shows a dialog box titled "Send Buffer Setting". It contains the following fields and options:

- Buffer Name:** Text input field containing "aaaaaa".
- CANID (Hex):** Text input field containing "0x0020FF00".
- DLC:** Spin box containing "6".
- ID Format:** Radio button group with "Standard (11bits)" and "Extended (11+18bits)". "Extended (11+18bits)" is selected.
- Interrupt at Transmit successful:** Radio button group with "Enable" and "Disable". "Disable" is selected.
- Frame Type:** Radio button group with "Data Frame" and "Remote Frame". "Data Frame" is selected.
- Buttons:** "OK" and "Cancel" buttons at the bottom.

- Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Do not use any of these names when entering message names.

ChX_MsgYY (X: 1 to 6 and YY: 00 to 63)

- Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set data length

Specify the CAN message's data length in the DLC field. Either enter the value directly or use the up/down arrows to change the entered value.

Enter a setting as a decimal value in a range from 0 to 15 bytes.

If the entered data length setting is more than 8 bytes, a warning message will appear.

Remark If the entered data length setting is more than 8 bytes, it is used as the length of data actually transmitted to the CAN bus.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for transmission end interrupts

Select either the enabled or disabled setting for transmission end interrupts. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.

- Select frame type

Select either data frame or remote frame as the frame type. The default setting (data frame) is already selected.

(b) Receive message buffer setting items

Figure 4-13. Receive Message Buffer Setting Screen (V850-aFCAN, 78K0-aFCAN)

The screenshot shows a dialog box titled "Receive Buffer Setting". It contains the following elements:

- Buffer Name:** A text input field containing "Ch1_Msg10".
- CANID (Hex):** A text input field containing "0x100".
- Mask Selection:** A pull-down menu currently showing "Mask2".
- ID Format:** A group box containing two radio buttons: "Standard (11bits)" (selected) and "Extended (11+18bits)".
- Interrupt at Valid message reception:** A group box containing two radio buttons: "Enable" and "Disable" (selected).
- Buffer's operate(If DN bit On):** A group box containing two radio buttons: "Overwrite" (selected) and "Discard".
- Frame Type:** A group box containing one radio button: "Data Frame" (selected).
- Buttons:** "OK" and "Cancel" buttons at the bottom.

- Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Be sure to avoid using any of these names when entering message names.

ChX_MsgYY (X: 1 to 6 and YY: 00 to 63)

- Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set mask link

When using the mask function, the setting for mask 1 to 4 set in **4. 4. 3 Mask settings** can be linked. Select the mask from the Mask Selection pull-down menu.

The default setting (None) is already selected.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for reception end interrupts

Select either the enabled or disabled setting for reception end interrupts. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.

- Select overwriting or discarding of data frame

Select Overwrite or Discard for messages newly received in a message buffer in which a message has already been received. Previously received messages will be overwritten by newly received messages when Overwrite is selected.

When Discard is selected, data frames newly received in a message bufferNote in which a message has already been received are discarded.

Note Receive message buffer for which DN bit is set to 1.

<Settings> (V850-FCAN)

(a) Transmit message buffer setting items

Figure 4-14. Transmit Message Buffer Setting Screen (V850-FCAN)

The screenshot shows a dialog box titled "Send Buffer Setting" with the following fields and options:

- Buffer Name:** Text field containing "aaaaaa".
- CANID (Hex):** Text field containing "0x0020FF00".
- DLC:** Spin box containing "6".
- ID Format:** Radio buttons for "Standard (11bits)" (unselected) and "Extended (11+18bits)" (selected).
- Interrupt at Transmit successful:** Radio buttons for "Enable" (unselected) and "Disable" (selected).
- Time Stamp Function:** Radio buttons for "Added" (selected) and "Not Added" (unselected).
- Frame Type:** Radio buttons for "Data Frame" (selected) and "Remote Frame" (unselected).
- Auto reply for Remote frame:** Radio buttons for "Set" (selected) and "Cancel" (unselected).
- If Remote frame is received:** Radio buttons for "Set the DN flag" (selected) and "Not set the DN flag" (unselected).

Buttons at the bottom: "OK" and "Cancel".

•Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Do not use any of these names when entering message names.

ChX_MsgYY (X : 1 to 6 and Y : 00 to 63)

•Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set data length

Specify the CAN message's data length in the DLC field. Either enter the value directly or use the up/down arrows to change the entered value.

Enter a setting as a decimal value in a range from 0 to 15 bytes.

If the entered data length setting is more than 8 bytes, a warning message will appear.

Remark If the entered data length setting is more than 8 bytes, it is used as the length of data actually transmitted to the CAN bus (initial setting). However, the data length that is transmitted is up to 8 bytes.

- Specify either whether the time stamp is added, when transmitting

Select either that the time stamp is added or is not, in the Time Stamp Function field. The default setting (added) is already entered.

- Specify either setting or releasing of remote frame automatic response function

Select either the setting or releasing of the remote frame automatic response function, in the Auto reply for Remote frame field. The default setting (set) is already entered.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for transmission end interrupts

Select either the enabled or disabled setting for transmission end interrupts. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.

- Select frame type

Select either data frame or remote frame as the frame type. The default setting (data frame) is already selected.

- Specify DN flag operation when receiving the remote frame in the transmit message buffer

Select either setting or not setting of the DN flag when receiving the remote frame.

The default setting (set) is already entered.

(b) Receive message buffer setting items

Figure 4-15. Receive Message Buffer Setting Screen (V850-FCAN)

- Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Be sure to avoid using any of these names when entering message names.

ChX_MsgYY (X : 1 to 6 and Y : 00 to 63)

- Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set mask link

When using the mask function, the setting for mask 1 to 4 set in **4.4.3 Mask settings** can be linked. When using in diagnostic processing mode, select Diagnostic. Select the mask from the Mask Selection pull-down menu.

The default setting (None) is already selected.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for reception end interrupts

Select either the enabled or disabled setting for message reception end interrupts in the Interrupt at Valid message reception field. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.

<Settings> (V850-DCAN, 78K0-DCAN)

(a) Transmit message buffer setting items

Figure 4-16. Transmit Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN)

- Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Do not use any of these names when entering message names.

ChX_MsgYY (X : 1 to 6 and Y : 00 to 63)

- Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set data length

Specify the CAN message's data length in the DLC field. Either enter the value directly or use the up/down arrows to change the entered value.

Enter a setting as a decimal value in a range from 0 to 15 bytes.

If the entered data length setting is more than 8 bytes, a warning message will appear.

Remark If the entered data length setting is more than 8 bytes, it is used as the length of data actually transmitted to the CAN bus (initial setting). However, the data length that is transmitted is up to 8 bytes.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for transmission end interrupts

Select either the enabled or disabled setting for transmission end interrupts. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.

- Select frame type

Select either data frame or remote frame as the frame type. The default setting (data frame) is already selected.

(b) Receive message buffer setting items

Figure 4-17. Receive Message Buffer Setting Screen (V850-DCAN, 78K0-DCAN)

- Buffer name

Enter the message buffer name in the Buffer Name field. The default message buffer name is already displayed in this field.

The user can use the message buffer name set in this area as the argument of the CAN software driver function (API) (message buffer specification).

Caution The following message names are reserved by the configurator. Be sure to avoid using any of these names when entering message names.

ChX_MsgYY (X : 1 to 6 and Y : 00 to 63)

- Set identifier

Enter the CANID (identifier) as a hexadecimal value. The configurator automatically adds "0x" to the start of the value.

The default value "0x000" is already entered.

The range of possible settings varies according to the frame format.

- Set mask link

Either masks 1 or 2, or None, which is set in **4. 4. 3 Mask settings**, is displayed in the pull-down menu. The setting cannot be changed from the pull-down menu.

- Select ID format

In the ID Format field, select either Standard ID or Extended ID. The default setting (Standard ID) is already entered.

- Select enabled or disabled setting for reception end interrupts
Select either the enabled or disabled setting for message reception end interrupts in the Interrupt at Valid message reception field. When these interrupts are enabled, interrupt requests will occur. The default setting (disabled) is already selected.
- Select frame type
Select either data frame or remote frame as the frame type. The default setting (data frame) is already selected.

4.4.5 Other settings

Enter other settings for CAN module functions and module interrupts.

<Startup method>

- Start by selecting [Other Setup] on [Tool] menu.

<Settings> (V850-aFCAN, 78K0-aFCAN)

- Set the transmit delay time for automatic block transfer.

If the automatic block transfer (ABT) function will be used, set the transmit delay time here. The unit for this setting is data bit time.

- Set operation in response to lost arbitration.

Select whether or not to retry transmission when arbitration is lost during single-shot mode.

- Enable/disable module interrupts

Set either enable or disable for the following interrupt-related settings.

- Interrupt at wakeup from sleep mode
- Interrupt when arbitration is lost
- Interrupt when a CAN protocol error occurs
- Interrupt when a CAN error status occurs
- Interrupt when valid message frame has been received from the message buffer
- Interrupt when normal message frame has been transmitted from the message buffer

Figure 4-18. Other Settings Screen (V850-aFCAN, 78K0-aFCAN)



<Settings> (V850-FCAN)

- Enable or disable global interrupt

Select either the enabled or disabled setting for interrupts regarding each of the following items.

- Memory access interrupt to unusable address
- Illegal write access interrupt (to temporary buffer, etc.)

- Select either use or non-use of time stamp function
Select the use or unuse of the time stamp function in the Time Stamp Function field. The default setting (Non-use) is already entered.

- Select dominant level of transmit pin
Select either the low level or high level is transmitted as dominant from the transmit pin. The default setting (Low level) is already entered.

- Select dominant level of receive pin
Select either the low level or high level to the receive pin is recognized as dominant. The default setting (Low level) is already entered.

- Enable/disable module interrupts
Set either enable or disable for the following interrupt-related settings.
 - CAN module error interrupt
 - CAN bus error interrupt
 - Interrupt at wakeup from sleep mode
 - Receive error passive interrupt
 - Transmit error passive or bus off interrupt
 - Reception end interrupt
 - Transmission end interrupt

Figure 4-19. Other Settings Screen (V850-FCAN)

Others Setting

Global Setting Information

Unavailable memory address access interrupt

Enable Disable

Invalid write access interrupt

Enable Disable

Time Stamp Function

Used Not used

Channel1

Module Function

Dominant of Sending

Low level High level

Dominant of Receiving

Low level High level

Time Capture timing when Receive the Message

Detect SOF Detect EOF

Module Interrupt Enable/Disable

CAN module error interrupt

Enable Disable

CAN bus error interrupt

Enable Disable

Wake up from CAN sleep mode interrupt

Enable Disable

Receive error passive interrupt

Enable Disable

Transmit error passive or bus off interrupt

Enable Disable

Receive completion interrupt

Enable Disable

Transmit completion interrupt

Enable Disable

OK Cancel

<Settings> (V850-DCAN, 78K0-DCAN)

- Set operation in response to lost arbitration.

Select whether or not to retry transmission when arbitration is lost during single-shot mode.

- Select either whether time stamp function is used or is not used

Select whether or not to use the time stamp function in the Use or not Function field. The default setting (non-use) is already entered.

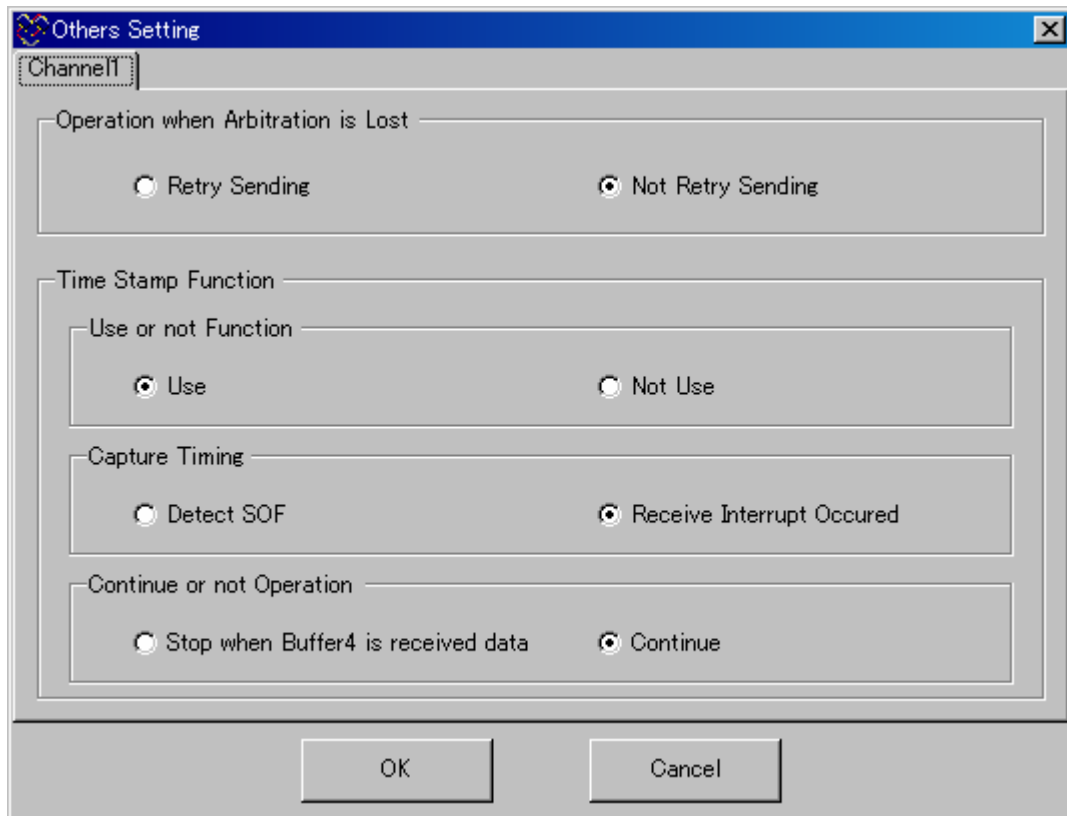
- Select SOFOUT output format

In the Capture Timing field, select whether SOFOUT is toggled by the interrupt that occurs when the message is received (time stamp mode) or SOFOUT is toggled for each start-of-frame reception (global time mode). The default setting (global time mode) is already entered.

- Select SOFTOUT function (SOFEn flag operation)

In the Continue or not Operation field, select either for the SOFEn bit not to depend on the CAN bus operation (toggle operation continued) or for the SOFEn bit to be cleared when the message begins to be stored in the receive message buffer 4 (toggle operation stopped). The default setting (toggle operation continued) is already entered.

Figure 4-20. Other Settings Screen (V850-DCAN, 78K0-DCAN)



4.4.6 Code generation

This includes source code for information files and configurator header files, as well as code generated for the CAN software driver source files and CAN software driver header file.

(1) Output option setting for CAN software driver source files

<Startup method>

- Start by selecting each command in the [Option] menu.

<Settings>

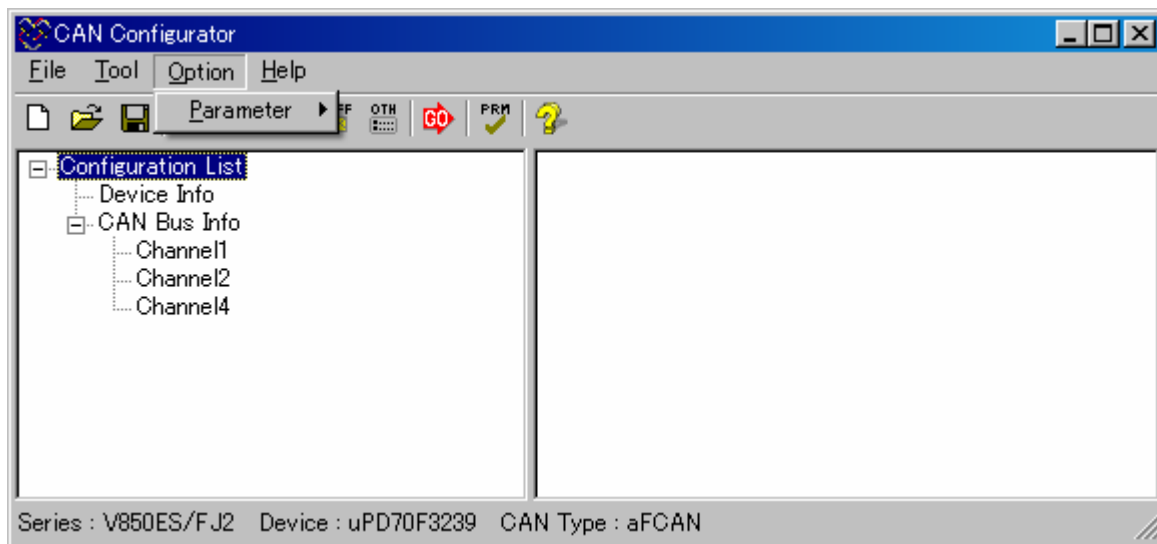
- [Option] – [Parameter]

Select whether or not to use the parameter check function for libraries to be output during library output.

Check	Outputs CAN software driver source files with parameter check.
No Check	Outputs CAN software driver source files without parameter check.

<R>

Figure 4-21. Output Options Setting Screen



(2) File output

Types of files to be output include information files, configurator header files, CAN software driver source files, and CAN software driver header files.

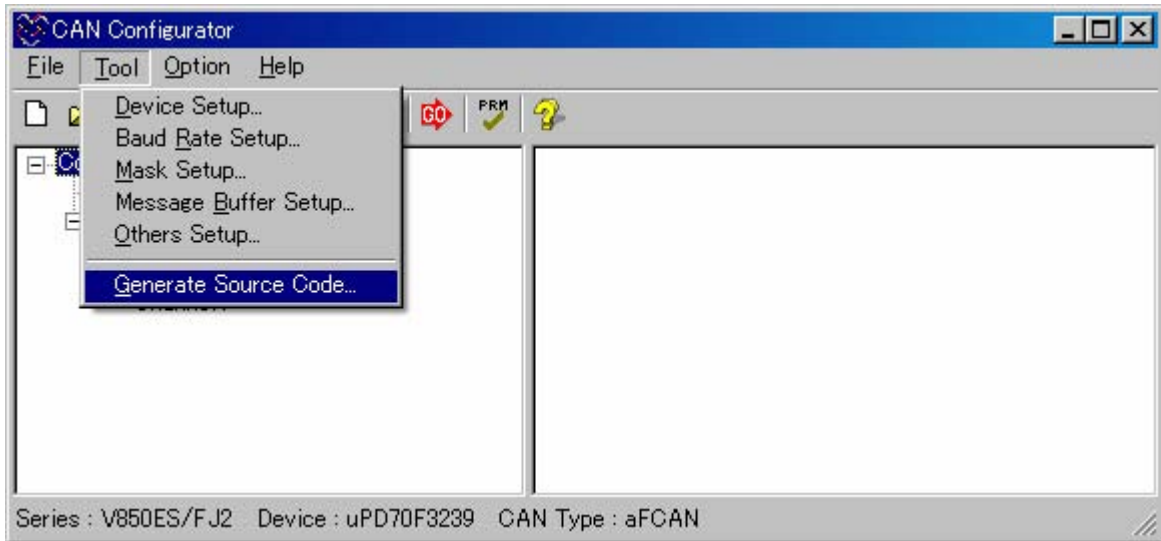
The user can specify any folder as the storage destination for these files.

<Startup method>

- Start by selecting [Generate source code] on [Tool] menu.

<R>

Figure 4-22. Code Generation Startup Screen



4.4.7 Saving and Opening Project Files

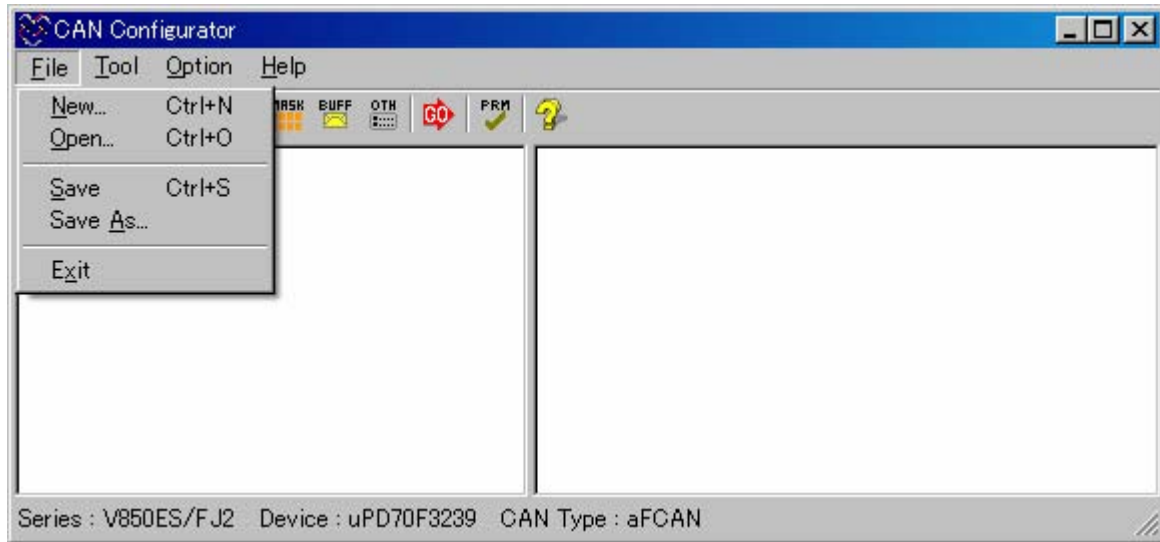
Project files that are used to manage information files and header files can be stored and opened. These project files are in XML format.

<Startup method>

- Select [Save As] in the [File] menu to save a new project file.
- Select [Save] in the [File] menu to save a project file (the previous project file will be overwritten).
- Select [Open] in the [File] menu to open a project file.

<R>

Figure 4-23. Screen for Saving and Opening Project Files



4.5 Error/Warning Message List

Messages such as those listed below are output when a setting is out of range or when a required setting has not been entered.

Table 4-1. Error Code List

(1/3)

Code Name	Message	Description	Action
(E)0101	Device was not selected. Select a device.	Displayed when [Tool]-[Generate Source Code...] (generating source code) is executed while no device is selected.	Select a device by [File]-[New] or [File]-[Open].
(E)0102	Channel was not selected. Select a channel.	Displayed when [Tool]-[Generate Source Code...] (generating source code) is executed while a channel to be used is not selected in the device selection dialog box.	Select a channel to be used by using [Tool]-[Device Setup...].
(E)0103	Baud rate was not selected. Select a baud rate.	Displayed when [Tool]-[Generate Source Code...] (generating source code) is executed while no baud rate is selected in the baud rate setting dialog box.	Select a baud rate value by using [Tool]-[Baud Rate Setup...].
(E)0104	The CAN clock is wrong value.	Displayed if a CAN clock value for which a baud rate value cannot be set is set in the device selection dialog box.	Set a CAN clock value in a range of $0 < \text{CAN clock} < 1310$.
(E)0105	The baud rate value is incorrect.	Displayed if a value other than a decimal number is input as a baud rate value in the baud rate setting dialog box.	Input a decimal value.
(E)0106	Enter a different baud rate value (from 5 to 1000 Kbps).	Displayed if a baud rate outside the range of 5 to 1000 kbps is selected in the baud rate setting dialog box.	Set a baud rate in the range of 5 to 1000 kbps.
(E)0107	Channel X not set to baud rate list. Select from list.	Displayed if there is a channel for which a list of combination of settings is not selected in the baud rate setting dialog box.	Select a combination of set values from the list.
(E)0108	Buffer name 'XXX' already exists.	Displayed if an attempt is made to set a buffer name that has already been used in the transmit (or receive) message buffer setting dialog box.	Set a buffer name that is not the same as other buffer name.

<R>

Code Name	Message	Description	Action
(E)0109	CANID 'XXX' is incorrect.	Displayed if a character string other than a hexadecimal number is input as CANID in the transmit (or receive) message buffer setting dialog box (however, excluding the first two characters 0x).	Input a value consisting of a combination of 0 to 9 and A to F.
(E)010A	CANID 'XXX' is outside setting range.	Displayed if a value outside the settable range is input as CANID in the transmit (or receive) message buffer setting dialog box.	Standard ID: 0x000 to 0x7FF Extended ID: 0x00000000 to 0x1FFFFFFF
(E)010B	Buffer name of 'XXX' is reserved name. Set another name.	Displayed if the default name of other buffer is set as a buffer name in the transmit (or receive) message buffer setting dialog box.	Set a buffer name that is not the default name of other buffer.
(E)010E	DLC 'XXX' is incorrect.	Displayed if characters other than numerals are set as the value of DLC in the transmit message buffer setting dialog box.	Set the value of DLC in a range of 0 to 15.
(E)010F	DLC 'XXX' is out of range. Enter a value in the range from 0 to 15.	Displayed if a value other than 0 to 15 is set as the value of DLC in the transmit message buffer setting dialog box.	Set the value of DLC in a range of 0 to 15.
(E)0111	CAN register area address is incorrect.	Displayed if values other than settable addresses are set as a CAN register area in the device selection dialog box.	Set settable addresses.
<R> (E)0112	Enter the CAN clock value.	Displayed if a CAN clock value is not set (blank) in the device selection dialog box.	Set a CAN clock value in a range of 0 < CAN clock < 1310.
<R> (E)0113	Enter the CAN register area address.	Displayed if no value of a CAN register area is set (blank) in the device selection dialog box.	Set settable addresses.
(E)0114	Enter a buffer name.	Displayed if no buffer name is set (blank) in the transmit (or receive) message buffer setting dialog box.	Set a buffer name.
(E)0115	Enter a CANID value.	Displayed if a value of CANID is not set (blank) in the transmit (or receive) message buffer setting dialog box.	Set a CANID.
(E)0116	Enter a DLC value.	Displayed if the value of DLC is not set (blank) in the transmit message buffer setting dialog box.	Set DLC.

Code Name	Message	Description	Action
(E)0117	Install database file DeviceEntry.xml.	Displayed if a database file for device-dependent information (DeviceEntry.xml) cannot be found.	Install a database file for device-dependent information (DeviceEntry.xml).
(E)0118	Install database file TemplateData.xml.	Displayed if a database file for code generation (TemplateData.xml) cannot be found.	Install a database file for code generation (TemplateData.xml).
(E)0119	The file name [candrv.h] is library header name. Set another name.	Displayed if candrv.c/h is set as the output file name of the configuration result.	Set an output file name other than candrv.c/h.
(E)011A	XXX is incorrect. (XXX is the set number of receive message buffers.)	Displayed if a character other than a numeral is set in the field of the number of receive message buffers used in the buffer registration dialog box for DCAN.	Set a numeral in the field of the number of receive message buffers used.
(E)011B	This channel doesn't have XXX receive buffers. Input the number less than N. (XXX is the set number of receive message buffers and N is the maximum number of buffers that can be registered as receive message buffers.)	Displayed if a value greater than the value that can be registered in the field of the number of receive message buffers used in the buffer registration dialog box for DCAN.	Set a number that can be registered in the field of the number of receive message buffers.
(E)011C	This device must use Rx buffer. Input the number of 1 or over.	Displayed if a value lower than 0 is set in the field of the number of receive message buffers in the buffer registration dialog box for DCAN (78K0).	Set a number that can be registered in the field of the number of receive message buffers.
(E)011D	The external CAN clock value is incorrect. Enter a different external CAN clock value.	Displayed if a character other than a numeral is set as the CAN module system clock in the baud rate setting dialog box when an external clock is used as the CAN module system clock (78K0).	Input a clock value in number as the CAN module system clock.
(E)011E	XXX doesn't exist, select other file! (XXX is a file name)	Displayed if a project file that is to be opened does not exist.	Select a project file that exists.
(E)011F	Target CAN message buffer has not been selected. Select a message buffer.	Displayed when [Tool]-[Generate Source Code...] (generating source code) is executed while no message buffer is set.	Set a message buffer by using [Tool]-[Baud Rate Setup...].
(E)0120	The microcontroller name XXX or the device name YYY is not in the database file DeviceEntry.xml. (XXX is a microcontroller name, YYY is a device name)	Displayed if the microcontroller name or device name described in the opened project file does not exist in a database file for device-dependent information (DeviceEntry.xml).	Select the project file that includes the microcontroller name and device name described in the database file.

<R>

Table 4-2. Warning Code List

(1/2)

Code Name	Message	Output Location	Remarks
(W)0001	Do you want to create a new project without saving the current project?	Displayed if [File]-[New] (creating new project) is executed when there is setting that is not saved to a file.	To not save the current setting: Press Yes button. To save the current setting: No → [File]-[Save]/[Save as]
<R> (W)0002	The CAN clock has been changed, the Baud rate setting should be update!	Displayed if the value of the CAN clock is changed by [Tool]-[Device Setup...] (displaying device selection dialog box).	Set the baud rate again in the baud rate setting dialog box.
(W)0003	The channel being used has been changed: Setting must be update!	Displayed if the number of channels is added by [Tool]-[Device Setup...] (displaying device selection dialog box).	Set the added channel in the respective dialog boxes (baud rate setting, mask setting, buffer registration, and other settings dialog boxes).
(W)0007	Buffer X can be used as a Send Message! (X is a buffer number.)	Displayed if an attempt is made to register any of buffers 0 to 7 to "For Rx Message" list while the ABT mode is selected in the buffer registration dialog box.	To use Buffer X as Rx Message: Select not to use the ABT mode.
(W)0008	XXX is set at Rx Message. Delete it from the Rx Message list. (XXX is a buffer name.)	Displayed if an attempt is made to use the ABT mode while any of buffers 0 to 7 is registered to the "For Rx Message" list in the buffer registration dialog box.	To use ABT mode: Delete buffers 0 to 7 registered to the "For Rx Message" list from the list.
<R> (W)0009	The DLC value in the message frame has to be transferred as programmed but only 8 data bytes a transferred in the data field. Do you want to set the DLC value (XX)? (XX is DLC value)	Displayed if an attempt is made to set a value 9 to 15 as DLC in the transmit message buffer setting dialog box.	To not use 9 to 15 as the DLC value, press the No button and set the DLC value again (if 9 to 15 is set, the set value is used as the DLC that is actually transmitted to the CAN bus, but 8-byte data is transmitted regardless of the set DLC value.)

Code Name	Message	Output Location	Remarks
(W)000A	Do you want to save the current project?	Displayed if [File]-[Exit] or the x button at the upper right of the window is pressed while some settings have not been saved to a file.	When current setting is saved, and application is ended: Press Yes button. When application is ended without save a current setting: Press No button. When application is not ended: Press Cancel button.
(W)000B	Select the using device or read the project file.	Displayed if an attempt is made to open the help while a device to be used is not selected.	Select a device to be used, by using [File]-[New], or open the existing project file by using [File]-[Open].
(W)000C	Check and update the message buffer setting!	Displayed if the mask operation setting in "apply area" is changed by mask setting dialog box for DCAN.	Check a message buffer with a [Tool]-[Message Buffer Setup...], and when required, perform a re-setup
(W)000D	Do you want to load the project without saving the current project?	Displayed if [File]-[Open] (reading existing project) is executed when there is setting that is not saved to a file.	To not save the current setting: Press Yes button. To save the current setting: No → [File]-[Save]/[Save as]

CHAPTER 5 DRIVER FUNCTIONS

5.1 List of Driver Functions

A list of driver functions is shown below.

5.1.1 Initialization and setting (6 types)

Function	Description
CanChEnable	Enables CAN (specifies channels)
CanAllEnable	Enable CAN (specifies all channels)
CanChInit	Initializes CAN channel (re-initializes channel specification)
CanAllInit	Initializes CAN channel (re-initializes all channels)
<R> CanChShutdown	Forced shutdown (specifies channels)
<R> CanAllShutdown	Forced shutdown (specifies all channels)

5.1.2 Operation modes (3 types)

Function	Description
CanChSetNrmMode	Set normal operation mode
CanChGetMode	Acquires operation mode and power-saving mode status
<R> CanChSetInitMode	Set Initialization mode

5.1.3 Buffer data acquisition (4 types)

Function	Description
CanMsgGetDatDlc	Acquires data and data length
CanMsgGetIdDatDlc	Acquires CAN-ID, data, and data length
CanMsgGetDatDlc_DSx	Acquires data and data length ^{Note}
CanMsgGetIdDatDlc_DSx	Acquires CAN-ID, data, and data length ^{Note}

5.1.4 Buffer data setting (4 types)

Function	Description
CanMsgSetDat	Sets data
CanMsgSetIdDatDlc	Sets CAN-ID, data, and data length
CanMsgSetDat_DSx	Sets data ^{Note} (x = 1 to 8)
CanMsgSetIdDatDlc_DSx	Sets CAN-ID, data, and data length ^{Note} (x = 1 to 8)

5.1.5 Transmit/receive confirmation (4 types)

Function	Description
CanMsgTxReq	Transmit request
CanMsgGetTxInfo	Acquires transmit information
CanChSrcRxInfo	Searches receive information (search DN)
CanChSrcRxInfo_MSxx	Searches receive information (search DN) ^{Note} (xx = 01 to 16)

Note Performance improving function dedicated to 78K0-DCAN.

5.1.6 CAN channel status acquisition (3 types)

Function	Description
CanChGetStatus	Acquires CAN channel status
CanChClrStatus	Clears CAN channel status
CanChGetBusStatus	Acquires CAN bus status

5.2 Data Types

All data types that are used by applications which use the CAN software driver are declared in candrv.h as a special data type (using typedef).

The data types used in the CAN software driver are listed in Table 5-1.

Table 5-1. Data Type List

Data Type in CAN Software Driver	Actual Data Type	Description
CD_ER	unsigned int ^{Note}	Error codes, return values
CD_ID	unsigned long	CAN-ID
CD_DLC	signed char	Data length
CD_DAT	unsigned char	CAN data
CD_CHNO	signed char	CAN channel number
CD_BUFNO	unsigned char	CAN message buffer number

Note The size of the int type is 4 bytes for the CA850 (V850). For the CC78K0 (78K0), it is 2 bytes.

Table 5-2 lists the range of values that can be specified for parameters. Note that operations may become undefined when an out of range value has been specified.

Table 5-2. Parameter Range

Data Type	Specifiable Range
CD_ID	0x0 to 0x1FFFFFFF
CD_DLC	0 to 15
CD_DAT	0x00 to 0xFF
CD_CHNO	0 to X (X: depends on number of channels implemented in device) ^{Note}
CD_BUFNO	0 to X (X: depends on number of buffers implemented in device) ^{Note}

Note For details, see the specific device's user's manual.

Table 5-3 lists the macros that are provided to specify channel numbers and other parameters.

Table 5-3. Macros for Parameters

Macro	Value	Description
CD_CAN1	0	Macro for CAN1 specification
CD_CAN2	1	Macro for CAN2 specification
CD_CAN3	2	Macro for CAN3 specification
CD_CAN4	3	Macro for CAN4 specification
CD_CAN5	4	Macro for CAN5 specification
CD_CAN6	5	Macro for CAN6 specification
CD_ERR_CLR_STS	0x0004	Macro that specifies clearing of CAN error status
CD_ERR_CLR_PRT	0x0008	Macro that specifies clearing of CAN protocol error status
CD_ERR_CLR_ABL	0x0010	Macro that specifies clearing of arbitration lost status
CD_ERR_CLR_WAK	0x0020	Macro that specifies clearing of wakeup from CAN sleep mode
CD_ERR_CLR_OVR	0x0040	Macro that specifies clearing of CAN overrun error status
CD_ERR_CLR_TXP	0x0080	Macro that specifies clearing of CAN transmission error passive status or bus-off status
CD_ERR_CLR_RXP	0x0100	Macro that specifies clearing of CAN reception error passive status

5.3 Return Values (Error Codes)

The CAN software driver functions return CD_ER type error codes (return values). The symbols used in these error codes are declared in the header file candrv.h. Table 5-4 lists return values that are returned by driver functions.

Table 5-4. Macros for Error Codes

Symbol	Value	Meaning
CD_TRUE	1	–
CD_FALSE	0	–
CD_E_OK	0x0	Normal end
CD_E_FLG	MSB = 1	MSB = 1. This indicates that the value is an error code.
CD_E_PRM	CD_E_FLG + 0x1	Parameter error
CD_E_STS	CD_E_FLG + 0x2	CAN module status error
CD_E_ALRDY	CD_E_FLG + 0x3	Already set
CD_E_NOMSG	CD_E_FLG + 0x4	Message not received

5.4 CAN-ID Conversion Macros

The CAN software driver functions handle the CAN-ID in the CAN software driver format, so be sure to use the conversion macros that are declared in candrv.h, which are listed in Table 5-5.

Remark See the [Use example] of the CanMsgGetIdDatDlc and CanMsgSetIdDatDlc functions for how to use the CAN-ID macro.

Table 5-5. List of CAN-ID Conversion Macros

Macro Name	Value	Description
CD_SET_STD_ID(id)	(id << 18)	Standard CAN-ID format → CAN software driver format
CD_SET_EXT_ID(id)	(id 0x80000000)	Extended CAN-ID format → CAN software driver format
CD_GET_STD_ID(id)	(id = (id >> 18) & 0x000007ff)	CAN software driver format → Standard CAN-ID format
CD_GET_EXT_ID(id)	(id = id & 0x1fffffff)	CAN software driver format → Extended CAN-ID format

5.5 Single-Channel Specification CAN Software Driver Functions

Because the CAN software driver functions for the 78K0 microcontroller are of single-channel specification (fixed channel specification), a channel cannot be specified. The functions of single-channel specification are different in API from the basic functions, but can be called by basic function names because the function name is replaced as follows depending on the definition in candrv.h.

Table 5-6. Single-Channel Specification CAN Software Driver Functions

Basic Function Name (Before Replacement)	78K0 Microcontroller-Dedicated Function Name (After Replacement)
CanChEnable(chno)	CanChEnable_CH1()
CanChInit(chno)	CanChInit_CH1()
CanChSetNrmMode(chno)	CanChSetNrmMode_CH1()
CanChGetMode(chno)	CanChGetMode_CH1()
<R> CanChSetInitMode(chno)	CanChSetInitMode_CH1()
CanMsgGetIdDatDlc(chno, bufno, p_canid, p_data, p_dlc)	CanMsgGetIdDatDlc_CH1(bufno, p_canid, p_data, p_dlc)
CanMsgGetDatDlc(chno, bufno, p_data, p_dlc)	CanMsgGetDatDlc_CH1(bufno, p_data, p_dlc)
CanMsgSetIdDatDlc(chno, bufno, canid, p_data, dlc)	CanMsgSetIdDatDlc_CH1(bufno, canid, p_data, dlc)
CanMsgSetDat(chno, bufno, p_data)	CanMsgSetDat_CH1(bufno, p_data)
CanMsgTxReq(chno, bufno)	CanMsgTxReq_CH1(bufno)
CanMsgGetTxInfo(chno, bufno)	CanMsgGetTxInfo_CH1(bufno)
CanChSrcRxInfo(chno, bufno)	CanChSrcRxInfo_CH1(bufno)
CanChGetStatus(chno)	CanChGetStatus_CH1()
CanChClrStatus(chno, clrdat)	CanChClrStatus_CH1(clrdat)
CanChGetBusStatus(chno)	CanChGetBusStatus_CH1()

[Caution]

With the 78K0 microcontroller, both the above function names can be called. To call a basic function name, a dummy channel number must be specified.

5.6 CAN Software Driver Functions with Improved Performance

Some functions with improved performance (processing speed) are available for the 78K0-DCAN.

Table 5-7. CAN Software Driver Functions with Improved Performance

Basic Function Name (Common to V850 and 78K0)	Name of Function with Improved Performance (78K0-DCAN only)
CanMsgGetIdDatDlc(chno, bufno, p_canid, p_data, p_dlc)	CanMsgGetIdDatDlc_DSx()
CanMsgGetDatDlc(chno, bufno, p_data, p_dlc)	CanMsgGetDatDlc_DSx()
CanMsgSetIdDatDlc(chno, bufno, canid, p_data, dlc)	CanMsgSetIdDatDlc_DSx()
CanMsgSetDat(chno, bufno, p_data)	CanMsgSetDat_DSx()
CanChSrcRxInfo(chno, bufno)	CanChSrcRxInfo_MSxx()

x: 1 to 8

xx: 01 to 16

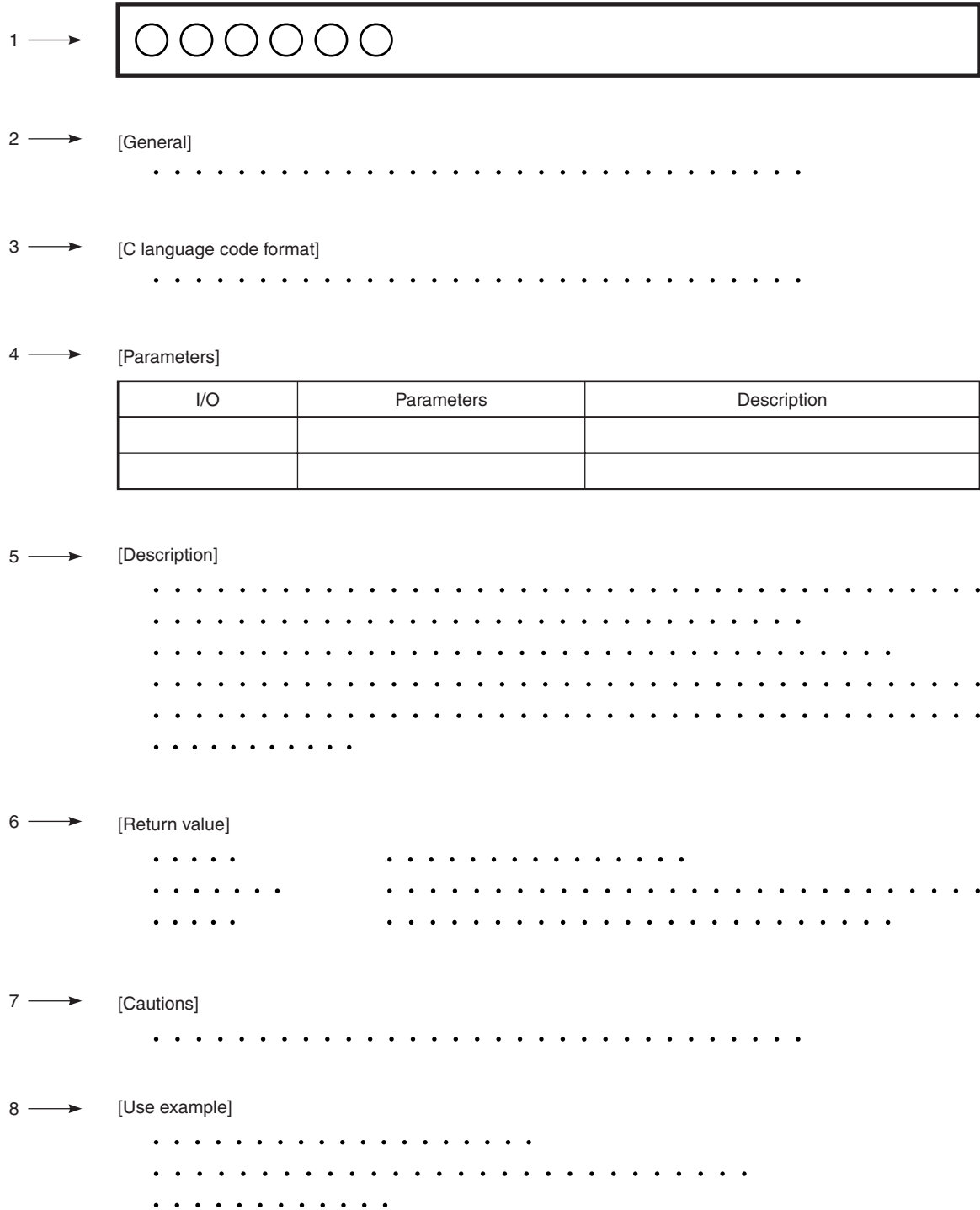
[Caution]

The functions with improved performance are different in API from the basic functions and do not have an argument. Data is transferred by using a global variable. For details, check the specification of each function.

5.7 Description of Driver Functions

The format shown in Figure 5-1 is used to describe the driver functions.

Figure 5-1. Code Format of Driver Functions



1. Name
This indicates the name of the driver function.
2. [General]
This indicates each driver function's general functions.
3. [C language code format]
This indicates the code format used to issue driver functions in C language.

4. [Parameters]
Driver function parameters are indicated in the following format.

I/O	Parameter	Description
A	B	C

A : Parameter I/O classification

I ... Input parameter

O ... Output parameter

B : Parameter type and name

C : Description of parameter

5. [Description]
This describes the functions of each driver function.
6. [Return value]
This uses macros and numerical value to indicate the values returned by driver functions.
7. [Cautions]
This indicates cautions concerning driver functions. In particular, device-dependent cautions are explained.
8. [Use example]
This provides use examples for specific driver functions.

5.8 Driver Functions

5.8.1 Initialization and setting

The driver functions listed in Table 5-8 are described below.

Table 5-8. Initialization and Setting

Function	Description
CanChEnable	Enables CAN (specifies channel)
CanAllEnable	Enables CAN (specifies all channels)
CanChInit	Initializes CAN channel (re-initializes)
CanAllInit	Initializes CAN channel (re-initializes all channels)
CanChShutdown	Forced shutdown (specifies channels)
CanAllShutdown	Forced shutdown (specifies all channels)

<R>

<R>

CanChEnable

<R> [General]

This function is used to set the specified channel's CAN clock and start the CAN controller.

[C language code format]

```
CD_ER CanChEnable (CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

<R> [Description]

This function sets the specified channel's CAN clock and starts the CAN controller^{Note}. When this operation ends normally, the CAN module is set to initialization mode.

If this function is issued after the CAN controller has been started, the CAN clock is not set and the CAN module is not switched to initialization mode.

This function must be issued before using any other driver function.

<R> **Note** The DCAN controller simply starts the CAN controller but does not perform the CAN clock setting.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
CD_E_ALRDY	CD_E_FLG + 0x3	CAN clock cannot be set because CAN controller has already been started

[Cautions]

This function cannot be used with the FCAN controller.

[Use example]

```
CD_ER ret;

ret = CanChEnable(CD_CAN1);
if (ret == CD_E_OK){
    "Describe normal end processing";
}
else if (ret == CD_E_PRM){
    "Describe error processing"; /* Specified channel number is invalid */
                                /* Parameter error occurred when using driver with parameter check */
}
else{ /* (ret == CD_E_ALRDY) */
    "Describe error processing"; /* CAN controller has been started */
}
```

CanAllEnable

[General]

This function is used to set the CAN clock of all channels and starts the CAN controller.

[C language code format]

```
CD_ER CanAllEnable();
```

[Parameters] None

[Description]

This driver function sets the CAN clock of all channels, starts the CAN controller, and sets the operation of the time stamp counter. When this function is terminated normally, all the CAN modules are in the initialization mode.

If this function is issued when even one of the CAN controllers has been started, settings such as that of the CAN clock are not performed nor is the operation mode of the CAN module changed.

This function must be issued before the other driver functions are used.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_ALRDY	CD_E_FLG + 0x3	CAN controller has already been started and the CAN clock cannot be set.

[Cautions]

This function can be used only with the FCAN controller.

[Use example]

```
CD_ER ret;

ret = CanAllEnable();
if (ret == CD_E_OK){
    "Describe Normal end processing";
}
else{
    /* (ret == CD_E_ALRDY) */
    "Describe error processing"; /* CAN controller has been started */
}
```

CanChInit

[General]

This function is used to initialize (reset) the specified channel.

[C language code format]

```
CD_ER CanChInit (CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This function performs the following settings for the specified channel.

- Baud rate setting
- Sample point setting
- DBT setting
- SJW setting
- Channel interrupt enable/disable setting^{Note}
- Mask register setting
- Message buffer settings (attribute setting, data clearing, etc.)
- Operation setting when arbitration is lost
- ABT delay setting^{Note}

The above settings are specified via a separate configuration file.

This function must be issued while the CAN module is in initialization mode.

If this function is issued when in any mode other than initialization mode, the above settings cannot be made.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
CD_E_STS	CD_E_FLG + 0x2	The function is issued when a CAN module is not in the initialization mode.

Note Except the DCAN controller

[Cautions]

This function cannot be used with the FCAN controller.

[Use example]

```
CD_ER ret;

ret = CanChInit(CD_CAN1);
if (ret == CD_E_OK){
    "Describe normal end processing";
}
else if (ret == CD_E_PRM){
    "Describe error processing"; /* Specified channel number is invalid */
                                /* Parameter error occurred when using driver with parameter check */
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* Issued in mode other than initialization mode */
}
```

CanAllInit

[General]

This function is used to initialize (re-initializes) all channels.

[C language code format]

```
CD_ER CanAllInit();
```

[Parameters] None

[Description]

This driver functions makes the following setting on a specified channel.

- Baud rate
- Sample point
- DBT
- SJW
- Enabling/disabling channel interrupt
- Mask register
- Message buffer (setting of attribute and clearing data)

The above settings are separately specified by the configuration file.

This function must be issued when all the CAN modules are in the initialization mode.

The above settings are not made if even one of the CAN modules is not in the initialization mode.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	The function is issued when a CAN module is not in the initialization mode.

[Cautions]

This function can be used only with the FCAN controller.

[Use example]

```
CD_ER ret;

ret = CanAllInit();
if (ret == CD_E_OK){
    "Describe Normal end processing";
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* Issued in mode other than initialization mode */
}
```

<R>

CanChShutdown

[General]

This function is used to shut down the specified channel forcibly.

[C language code format]

```
CD_ER CanChShutdown(CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This driver function is used to shut down the specified channel forcibly.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
CD_E_STS	CD_E_FLG + 0x2	Forced shutdown failed

[Cautions]

This function is available only when the aFCAN controller is used.

[Use example]

```
CD_ER ret;

ret = CanChShutdown(CD_CAN1);
if (ret == CD_E_OK){
    "Describe Normal end processing";
}
else if (ret == CD_E_PRM){
    "Describe error processing"; /* Specified channel number is invalid */
    /* Parameter error occurred when using driver with parameter check */
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* Forced shutdown failed */
}
```

<R> CanAllShutdown**[General]**

This function is used to shut down all the channels forcibly.

[C language code format]

```
CD_ER CanAllShutdown();
```

[Parameters] None**[Description]**

This driver function is used to shut down all the channels forcibly.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	Forced shutdown failed

[Cautions]

This function can be used only with the FCAN controller.

[Use example]

```
CD_ER ret;

ret = CanAllShutdown();
if (ret == CD_E_OK){
    "Describe Normal end processing";
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* Forced shutdown failed */
}
```


5.8.2 Operation modes

The driver function operation modes listed in Table 5-9 are described below.

Table 5-9. Operation Modes

Function	Description
CanChSetNrmMode	Set normal operation mode
CanChGetMode	Acquire operation mode and power-saving mode status
CanChSetInitMode	Set initialization mode

<R>

CanChSetNrmMode

[General]

This function is used to set the specified channel to normal operation mode.

[C language code format]

```
CD_ER CanChSetNrmMode (CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This function switches the CAN module for the specified channel number from initialization mode to normal operation mode. This function must be issued before transmitting or receiving any messages after CanChEnable and CanChInit have been issued.

This function must be issued while the CAN module is in initialization mode.

If this function is issued when in any mode other than initialization mode, the operation mode cannot be changed.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
CD_E_STS	CD_E_FLG + 0x2	The function is issued when a CAN module is not in the initialization mode.

[Use example]

```
CD_ER ret;

ret = CanChSetNrmMode (CD_CAN1);
if (ret == CD_E_OK){
    "Describe normal end processing";
}
else if (ret == CD_E_PRM){
    "Describe error processing"; /* Specified channel number is invalid */
    /* Parameter error occurred when using driver with parameter check */
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* Issued in mode other than initialization mode */
}
```

CanChGetMode

[General]

This function is used to acquire the specified channel's operation mode and power-saving mode statuses.

[C language code format]

```
CD_ER CanChGetMode (CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This function is used to acquire the specified channel's operation mode and power-saving mode status and return it as the return value.

[Return value]

Error Code	Value	Meaning
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
-	When MSB = 0	See Description of bits when MSB = 0

• Description of bits when MSB = 0

Bit Position	Description
Bits 2 to 0	Operation mode status (OPMODE) 000: Initialization mode 001: Normal operation mode 010: Normal operation mode with auto block transfer function ^{Note} 011: Receive-only mode 100: Single-shot mode 101: Self-test mode ^{Note}
Bits 4, 3	Power-saving mode status (PSMODE) 00: Power-saving mode not selected 01: CAN sleep mode 11: CAN stop mode
MSB to bit 5	Fixed to zero

Note aFCAN controller only.

[Use example]

```

CD_ER ret;
CD_ER pmmode;
CD_ER psmode;
ret = CanChGetMode(CD_CAN1);
if (ret == CD_E_PRM){
    "Describe error processing" /* Specified channel number is invalid */
                                /* Parameter error occurred when using driver with parameter check */
}
else{
    pmmode = ret & 0x07;
    switch (pmmode){
        case 0:
            "Processing in initialization mode"; /* Currently set to initialization mode */
            break;
        case 1:
            "Processing in normal operation mode"; /* Currently set to normal operation mode */
            break;
        case 2:
            "Processing in ABT mode"; /* Currently set to ABT mode */
            break;
        case 3:
            "Processing in receive-only mode"; /* Currently set to receive-only mode */
            break;
        case 4:
            "Processing in single-shot mode"; /* Currently set to single-shot mode */
            break;
        case 5:
            "Processing in self-test mode"; /* Currently set to self-testmode */
            break;
    }
    psmode = (ret >> 3) & 0x02;
    switch (psmode){
        case 0:
            "Processing in mode other than power-saving mode";
            break;
        case 1:
            "Processing in CAN sleep mode"; /* Currently in CAN sleep mode */
            break;
        case 2:
            "Processing in CAN stop mode"; /* Currently in CAN stop mode */
            break;
    }
}
}

```

<R>

CanChSetInitMode

[General]

This function is used to set the specified channel to the initialization mode.

[C language code format]

```
CD_ER CanChSetInitMode(CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This driver function is used to set the specified channel in a CAN module, which is in an operating mode other than the initialization mode, to the initialization mode.

The operating mode does not change if this function is issued when the CAN module has already been in the initialization mode.

After calling this function, be sure to confirm that the initialization mode is entered, using CanChGetMode.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
CD_E_STS	CD_E_FLG + 0x2	The initialization mode has been entered

[Use example]

```
CD_ER ret;

ret = CanChSetInitMode(CD_CAN1);
if (ret == CD_E_OK){
    ret = CanChGetMode(CD_CAN1);
    if (ret == 0x00){
        "Describe normal end processing";
    }
    else{
        /* The initialization mode has not been entered */
    }
}
else if (ret == CD_E_PRM){
    "Describe error processing"; /* Specified channel number is invalid */
    /* Parameter error occurred when using driver with parameter check */
}
else{
    /* (ret == CD_E_STS) */
    "Describe error processing"; /* The initialization mode has been entered */
}
```

5.8.3 Buffer data acquisition

The driver functions listed in Table 5-10 are described below.

Table 5-10. Buffer Data Acquisition

Function	Description
CanMsgGetDatDlc	Acquires data and data length
CanMsgGetIdDatDlc	Acquires CAN-ID, data, and data length
CanMsgGetDatDlc_DSx	Acquires data and data length ^{Note} (x = 1 to 8)
CanMsgGetIdDatDlc_DSx	Acquires CAN-ID, data, and data length ^{Note} (x = 1 to 8)

Note Functions dedicated to 78K0-DCAN and with improved performance.

CanMsgGetDatDlc

[General]

This function is used to acquire the data and data length from the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgGetDatDlc( CD_CHNO chno, CD_BUFNO bufno,
                      CD_DAT* p_data, CD_DLC* p_dlc);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number
O	CD_DAT* p_data	Start address of area for storing message data
O	CD_DLC* p_dlc	Start address of area for storing message length

[Description]

This function is used to acquire the following data from the specified message buffer.

- Data (acquires DLC byte count only: maximum is 8 bytes)
- DLC value (acquired value)

This function first checks for new data. If there is no new data, there is no data acquisition operation. The data update bit (DN bit) is cleared when data is acquired.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_STS	CD_E_FLG + 0x2	Acquiring data has failed
CD_E_NOMSG	CD_E_FLG + 0x4	No new data

[Cautions]

When the DCAN controller is used, only the receive buffer can obtain data by using this function.

[Use example]

```

/* The following are set in advance by the configurator.
· One message buffer is assigned for reception.
  (Buffer name is defined as Ch1_Msg01, using channel 1's message buffer 1.)
· Standard ID format is set as the assigned message buffer's frame format.
· The CAN-ID to be received is set as any value.
· Mask setting (when required)*/

CD_DLC canRdlc;
CD_DAT canRdata[8];
CD_DAT tempdata[8];
CD_ER ret;
int i;

ret = CanMsgGetDatDlc(CD_CAN1,Ch1_Msg01,canRdata,&canRdlc);
if (ret == CD_E_OK){
  for (i=0; i<canRdlc ; i++ ){
    tempdata[i] = canRdata[i];          /* DLC-byte data is moved to a different buffer */
  }
}
else if (ret == CD_E_NOMSG){
  "Processing when message is not received"; /* No new messages */
}
else{
  "Processing when parameter error occurs"; /* Parameter error (ret == CD_E_PRM) occurred */
                                          /* while using driver with parameter check function */
}

```


CanMsgGetIdDatDlc

[General]

This function is used to acquire the CAN-ID, data, and data length from the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgGetIdDatDlc( CD_CHNO chno, CD_BUFNO bufno,
                        CD_ID* p_canid, CD_DAT* p_data, CD_DLC* p_dlc);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number
O	CD_ID* p_canid	Start address of area for storing CAN-ID
O	CD_DAT* p_data	Start address of area for storing message data
O	CD_DLC* p_dlc	Start address of area for storing message length

[Description]

This function is used to acquire the following data from the specified message buffer.

- CAN-ID (set using CAN software driver format)
- Data (acquire DLC byte count only: maximum is 8 bytes)
- DLC value (acquired value)

Since the CAN-ID is set using the CAN software driver format, use a CAN-ID conversion macro to reference the CAN-ID.

This function first checks for new data. If there is no new data, there is no data acquisition operation. The data update bit (DN bit) is cleared when data is acquired.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_STS	CD_E_FLG + 0x2	Acquiring data has failed
CD_E_NOMSG	CD_E_FLG + 0x4	No new data

[Cautions]

When the DCAN controller is used, only the receive buffer can obtain data by using this function.

[Use example]

```

/* The following are set in advance by the configurator.
· One message buffer is assigned for reception.
  (Buffer name is defined as Ch1_Msg01, using channel 1's message buffer 1.)
· Standard ID format is set as the assigned message buffer's frame format.
· The CAN-ID to be received is set as any value.
· Mask setting (when required)
*/

CD_ID canRid;
CD_DLC canRdlc;
CD_DAT canRdata[8];
CD_DAT tempdata[8];
CD_ER ret;
int i;

ret = CanMsgGetIdDatDlc(CD_CAN1, Ch1_Msg01, &canRid, canRdata, &canRdlc);
if (ret == CD_E_OK){
    CD_GET_STD_ID(canRid);          /* Acquired CAN-ID is converted to the standard IDformat. */
    for (i=0; i<canRdlc ; i++){
        tempdata[i] = canRdata[i]; /* DLC-byte data is moved to a different buffer */
    }
}
else if (ret == CD_E_NOMSG){
    "Processing when message is not received"; /* No new messages */
}
else{
    "Processing when error occurs";          /* Parameter error (ret == CD_E_PRM) occurred */
                                           /* after data acquisition failure (ret == CD_E_STS) or */
                                           /* while using driver with parameter check function */
}

```

CanMsgGetDatDlc_DSx (x = 1 to 8)

[General]

Function dedicated to the 78K0-DCAN and with improved performance.

This function is used to obtain a data length from the message buffer of a specified channel and store it in a global variable.

[C language code format]

```

CD_ER CanMsgGetDatDlc_DS1 ();      (Obtained data size = 1)
CD_ER CanMsgGetDatDlc_DS2 ();      (Obtained data size = 2)
CD_ER CanMsgGetDatDlc_DS3 ();      (Obtained data size = 3)
CD_ER CanMsgGetDatDlc_DS4 ();      (Obtained data size = 4)
CD_ER CanMsgGetDatDlc_DS5 ();      (Obtained data size = 5)
CD_ER CanMsgGetDatDlc_DS6 ();      (Obtained data size = 6)
CD_ER CanMsgGetDatDlc_DS7 ();      (Obtained data size = 7)
CD_ER CanMsgGetDatDlc_DS8 ();      (Obtained data size = 8)

```

[Parameters] None

[Global variables]

I/O	Global Variable Name	Description
I	unsigned char* u1gp_rxbuf_addr	Address of the DSTAT register of the target receive buffer. The result of search by CanChSrcRxInfo MSxx() can be used as is. See to [Use example].
O	CD_DAT u1g_rxdata	Global variable in which the message data that has been obtained is to be stored (The data size is set in accordance with the function to be used.)
O	CD_DLC u1g_rxdlc	Global function in which DLC that has been obtained is to be stored

[Description]

This driver function obtains the following data from a specified message buffer and stores the data in a global variable.

- Data (Data size to be obtained is fixed depending on the function.)
- DLC value (Value that has been obtained as is)

This function first checks whether there is new data. It obtains no data if there is no new data.

When the function has obtained data, it also clears the data updating (DN) bit.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	Acquiring data has failed
CD_E_NOMSG	CD_E_FLG + 0x4	No new data

[Cautions]

This function can obtain data only from the receive buffer.

<R> [Use example]

```

/* The following are set in advance by the configurator.
   Message buffers 1 to 16 are assigned for reception.
   (Buffer name is defined as Ch1_Msgxx, using channel 1's message buffer xx.)
   Standard ID format is set as the assigned message buffer's frame format.
   The CAN-ID to be received is set as any value.
   Mask setting (when required)
Define the following global variable.
   unsigned char* ulgp_rxbuf_addr;
   CD_DAT        ulg_rxdata[8];
   CD_DLC        ulg_rxdlc;
*/

CD_DAT tempdata[8];
CD_ER ret1;
CD_ER ret2;
int i;

ret1 = CanChSrcRxInfo_MS01(); /* Searches receiving buffer. */
/* Search result is stored in global variable ulgp_rxbuf_addr. */

if(!(ret1 & CD_E_FLG)) {
    ret2 = CanMsgGetDatDlc_DS8(); /* Acquires 8 bytes of data from buffer specified by ulgp_rxbuf_addr */
    if (ret2 == CD_E_OK) {
        for (i=0; i<ulg_rxdlc ; i++) {
            tempdata[i] = ulg_rxdata[i]; /* DLC-byte data is moved to a different buffer */
        }
    }
    else if (ret2 == CD_E_NOMSG) {
        "Processing when message is not received"; /* No new messages */
    }
    else {
        "Processing in case of parameter error"; /* Parameter error (ret == CD_E_PRM) occurred */
        /* while using driver with parameter check function. */
    }
}
}

```

CanMsgGetIdDatDlc_DSx (x = 1 to 8)

[General]

Function dedicated to the 78K0-DCAN and with improved performance.

This function is used to obtain CAN-ID, data, and data length from the message buffer of a specified channel and store them in a global variable.

[C language code format]

```

CD_ER CanMsgGetIdDatDlc_DS1 (); (Obtained data size = 1)
CD_ER CanMsgGetIdDatDlc_DS2 (); (Obtained data size = 2)
CD_ER CanMsgGetIdDatDlc_DS3 (); (Obtained data size = 3)
CD_ER CanMsgGetIdDatDlc_DS4 (); (Obtained data size = 4)
CD_ER CanMsgGetIdDatDlc_DS5 (); (Obtained data size = 5)
CD_ER CanMsgGetIdDatDlc_DS6 (); (Obtained data size = 6)
CD_ER CanMsgGetIdDatDlc_DS7 (); (Obtained data size = 7)
CD_ER CanMsgGetIdDatDlc_DS8 (); (Obtained data size = 8)
    
```

[Parameters] None

[Global variables]

I/O	Global Variable Name	Description
I	unsigned char* u1gp_rxbuf_addr	Address of the DSTAT register of the target receive buffer. The result of search by CanChSrcRxInfo_MSxx() can be used as is. See [Use example].
O	unsigned char u1g_rxid[5]	Global variable in which CAN-ID that has been obtained is to be stored
O	CD_DAT u1g_rxdata	Global variable in which the message data that has been obtained is to be stored (The data size is set in accordance with the function to be used)
O	CD_DLC u1g_rxdc	Global function in which DLC that has been obtained is to be stored

<R>

[Description]

This driver function obtains the following data from a specified message buffer and stores the data in a global variable.

- CAN-ID (set in register image)
- Data (Data size to be obtained is fixed depending on the function.)
- DLC value (Value that has been obtained as is) However, the most significant bit indicates the format of CAN-ID.
0: Standard CAN-ID, 1: Extended CAN-ID)

This function first checks whether there is new data. It obtains no data if there is no new data. When the function has obtained data, it also clears the data updating (DN) bit.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	Acquiring data has failed
CD_E_NOMSG	CD_E_FLG + 0x4	No new data

[Cautions]

This function can obtain data only from the receive buffer.

[Use example]

```

/* The following are set in advance by the configurator.
· Message buffers 0 to 15 are assigned for reception.
  (Buffer name is defined as Ch1_Msgxx, using channel 1's message buffer xx.)
· Standard ID format is set as the assigned message buffer's frame format.
· The CAN-ID to be received is set as any value.
· Mask setting (when required)
Define the following global variable.
  unsigned char *ulgp_rxbuf_addr;
  unsigned char  ulg_rxid[5]
  CD_DAT      ulg_rxdata[8];
  CD_DLC      ulg_rxdlc;
*/

unsigned char tempid[5];
CD_DAT tempdata[8];
CD_ER  ret1;
CD_ER  ret2;
int i;

ret1 = CanChSrcRxInfo_MS01();      /* Searches receiving buffer. */
/* Search result is stored in global variable ulgp_rxbuf_addr. */

if(!(ret1 & CD_E_FLG)) {
  ret2 = CanMsgGetIdDatDlc_DS8();  /* Acquires 8 bytes of data from buffer specified by ulgp_rxbuf_addr. */
  if (ret2 == CD_E_OK){
    for (i=0; i<ulg_rxdlc ; i++){
      tempdata[i] = ulg_rxdata[i]; /* DLC-byte data is moved to a different buffer */
    }
    for (i=0; i<5 ; i++){
      tempid[i] = ulg_rxid[i];     /* CAN-ID is moved to a different buffer */
    }
  }
}
else if (ret2 == CD_E_NOMSG){
  "Processing when message is not received"; /* No new messages */
}
else{
  "Processing in case of parameter error"; /* Parameter error (ret == CD_E_PRM) occurred */
/* while using driver with parameter check function. */
}
}

```

5.8.4 Buffer data setting

The functions listed in Table 5-11 are described below.

Table 5-11. Buffer Data Setting

Function	Description
CanMsgGetDatDlc	Sets data
CanMsgGetIdDatDlc	Sets CAN-ID, data, and data length
CanMsgSetDat_DSx	Sets data ^{Note} (x = 1 to 8)
CanMsgSetIdDatDlc_DSx	Sets CAN-ID, data, and data length ^{Note} (x = 1 to 8)

Note Functions dedicated to 78K0-DCAN and with improved performance

CanMsgSetDat

[General]

This function is used to set data to the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgSetDat (CD_CHNO chno, CD_BUFNO bufno, CD_DAT* p_data);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number
I	CD_DAT* p_data	Start address of area for storing message data

[Description]

This function is used to set the following data to the specified message buffer.

- Data (data length: current value set to the buffer (DLC))

Values set to the message buffer (initial values set upon configuration when initialization is complete) are used to set the transmit CAN-ID and message length.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_STS	CD_E_FLG + 0x2	Data setting failed

[Cautions]

If a value has been changed from the initial value set during CAN-ID and DLC configuration by the CanMsgSetIdDatDlc function, the modified value will be transmitted thereafter by the CanMsgSetDat function as well.

With the DCAN controller, this function can set data for only the transmit buffer.

[Use example]

```

/* The following are set in advance by the configurator.
   One message buffer is assigned for reception.
   (Buffer name is defined as Ch1_Msg00, using channel 1's message buffer 0.)
   The DLC is set to 8 bytes
   The CAN-ID to be transmitted is set.
   Standard ID format is set as the assigned message buffer's frame format.
   Data frame is set as the assigned message's frame type
*/

CD_DAT canTdata[8];
CD_ER ret;

canTdata[0] = 0x00; /* Tx data byte1 */
canTdata[1] = 0x11; /* Tx data byte2 */
canTdata[2] = 0x22; /* Tx data byte3 */
canTdata[3] = 0x33; /* Tx data byte4 */
canTdata[4] = 0x44; /* Tx data byte5 */
canTdata[5] = 0x55; /* Tx data byte6 */
canTdata[6] = 0x66; /* Tx data byte7 */
canTdata[7] = 0x77; /* Tx data byte8 */

ret = CanMsgSetDat(CD_CAN1, Ch1_Msg00, canTdata);

/* Transmit data is set to message buffer */
if(ret == CD_E_OK){
    ret = CanMsgTxReq(CD_CAN1, Ch1_Msg00); /* Message buffer's transmit request bit is set */
    if(ret == CD_E_OK){
        "Processing when transmit request is successful"; /* Transmit request bit is set */
    }
    else{
        "Processing when transmit request failed"; /* Setting of transmit request bit failed */
    }
}
else{
    "Processing when parameter error occurs"; /* Parameter error (ret == CD_E_PRM) occurred */
    /* while using driver with parameter check function */
}

```

CanMsgSetIdDatDlc

[General]

This function is used to set the CAN-ID, data, and data length to the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgSetIdDatDlc( CD_CHNO chno, CD_BUFNO bufno,
                        CD_ID canid, CD_DAT* p_data, CD_DLC dlc);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number
I	CD_ID canid	CAN-ID (CAN software driver format)
I	CD_DAT* p_data	Start address of area for storing message data
I	CD_DLC dlc	Message length

[Description]

This function is used to set the following data to the specified message buffer.

- CAN-ID (sets using the CAN software driver format)
- Data (sets DLC byte count only: maximum is 8 bytes)
- DLC value (only lower 4 bits are valid)

Since the CAN-ID must be set in the CAN software driver format, use a CAN-ID conversion macro to set it.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_STS	CD_E_FLG + 0x2	Data setting failed

[Cautions]

With the DCAN controller, this function can set data for only the transmit buffer.

[Use example]

```

/* The following are set in advance by the configurator.
· One message buffer is assigned for reception.
  (Buffer name is defined as Ch1_Msg00, using channel 1's message buffer 0.)
· Standard ID format is set as the assigned message buffer's frame format.
· Data frame is set as the assigned message's frame type
*/

CD_ID canTid;
CD_DLC canTdlc;
CD_DAT canTdata[8];
CD_ER ret;

canTdata[0] = 0x00; /* Tx data byte1 */
canTdata[1] = 0x11; /* Tx data byte2 */
canTdata[2] = 0x22; /* Tx data byte3 */
canTdata[3] = 0x33; /* Tx data byte4 */
canTdata[4] = 0x44; /* Tx data byte5 */
canTdata[5] = 0x55; /* Tx data byte6 */
canTdata[6] = 0x66; /* Tx data byte7 */
canTdata[7] = 0x77; /* Tx data byte8 */

canTid = CD_SET_STD_ID(0x100); /* ID with standard ID frame (0x100) is converted into driver format*/
canTdlc = 8; /* DLC is 8 bytes */

ret = CanMsgSetIdDatDlc(CD_CAN1, Ch1_Msg00, canTid, canTdata, canTdlc);
/* Transmit data is set to messagebuffer */

if(ret == CD_E_OK){
  ret = CanMsgTxReq(CD_CAN1, Ch1_Msg00); /* Message buffer's transmit requestbit is set */
  if(ret == CD_E_OK){
    "Processing when transmit request is successful"; /* Transmit request bit is set */
  }
  else{
    "Processing when transmit request failed"; /* Setting of transmit request bit failed */
  }
}
else{
  "Processing when data setting fails or parameter error occurs";
  /* Data setting failure (ret == CD_E_STS) or */
  /* parameter error (ret == CD_E_PRM) occurred */
  /* while using driver with parameter check function */
}

```

CanMsgSetDat_DSx (x = 1 to 8)

[General]

Function dedicated to the 78K0-DCAN and with improved performance.

This function is used to set data specified by a global variable to a specified message buffer.

[C language code format]

```

CD_ER CanMsgSetDat_DS1 ();          (Set data size = 1)
CD_ER CanMsgSetDat_DS2 ();          (Set data size = 2)
CD_ER CanMsgSetDat_DS3 ();          (Set data size = 3)
CD_ER CanMsgSetDat_DS4 ();          (Set data size = 4)
CD_ER CanMsgSetDat_DS5 ();          (Set data size = 5)
CD_ER CanMsgSetDat_DS6 ();          (Set data size = 6)
CD_ER CanMsgSetDat_DS7 ();          (Set data size = 7)
CD_ER CanMsgSetDat_DS8 ();          (Set data size = 8)

```

[Parameters] None

[Global variables]

I/O	Global Variable Name	Description
I	unsigned char* u1gp_txbuf_addr	Address of the TCON register of the target transmit buffer
I	CD_DAT u1g_txdata	Global variable in which message data to be set is stored (Data size is set depending on the function to be used.)

[Description]

This driver functions sets the following data to a specified message buffer.

- Data (Data size is set depending on the function to be used.)

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	Setting data has failed

[Cautions]

This function can set data only for the transmit buffer.

[Use example]

```

/* The following are set in advance by the configurator.
   One message buffer is assigned for transmission.
   (Buffer name is defined as Ch1_Msg00, using channel 1's message buffer 0.)
   DLC is set to 8 bytes.
   CAN-ID to be transmitted is set.
   Standard ID format is set as the assigned message buffer's frame format.
   Data frame is set as the assigned message buffer's frame type.
Define the following global variable.
   unsigned char * ulgp_txbuf_addr;
   CD_DAT      ulg_txdata[8];
The following macro is defined. (Address of message buffer to be set)
   #define      TXBUFADD_00      (0xf600)
*/

CD_ER  ret;

ulgp_txbuf_addr = (unsigned char *) (TXBUFADD_00);
ulg_txdata[0] = 0x00;          /* Tx data byte1 */
ulg_txdata[1] = 0x11;          /* Tx data byte2 */
ulg_txdata[2] = 0x22;          /* Tx data byte3 */
ulg_txdata[3] = 0x33;          /* Tx data byte4 */
ulg_txdata[4] = 0x44;          /* Tx data byte5 */
ulg_txdata[5] = 0x55;          /* Tx data byte6 */
ulg_txdata[6] = 0x66;          /* Tx data byte7 */
ulg_txdata[7] = 0x77;          /* Tx data byte8 */

ret = CanMsgSetDat_DS8();      /* Sets transmit data to the message buffer */
if(ret == CD_E_OK){
    ret = CanMsgTxReq(CD_CAN1,Ch1_Msg00); /* Sets the transmit request bit of the message buffer */
    if(ret == CD_E_OK){
        "Processing when transmit request is successful"; /* Sets the transmit request bit */
    }
    else{
        "Processing when transmit request has failed"; /* Setting the transmit request has failed. */
    }
}
else{
    "Processing when setting data has failed"; /* Setting data has failed (ret == CD_E_ETS). */
}

```

CanMsgSetIdDatDlc_DSx (x = 1 to 8)

[General]

Function dedicated to the 78K0-DCAN and with improved performance.

This function is used to set CAN-ID, data, and data length specified by a global variable to a specified message buffer.

[C language code format]

```

CD_ER CanMsgSetIdDatDlc_DS1 ();    (Set data size = 1)
CD_ER CanMsgSetIdDatDlc_DS2 ();    (Set data size = 2)
CD_ER CanMsgSetIdDatDlc_DS3 ();    (Set data size = 3)
CD_ER CanMsgSetIdDatDlc_DS4 ();    (Set data size = 4)
CD_ER CanMsgSetIdDatDlc_DS5 ();    (Set data size = 5)
CD_ER CanMsgSetIdDatDlc_DS6 ();    (Set data size = 6)
CD_ER CanMsgSetIdDatDlc_DS7 ();    (Set data size = 7)
CD_ER CanMsgSetIdDatDlc_DS8 ();    (Set data size = 8)

```

[Parameters] None

[Global variables]

I/O	Global Variable Name	Description
I	unsigned char* u1gp_txbuf_addr	Address of the TCON register of the target transmit buffer
I	unsigned char u1g_txid[5]	Global variable in which CAN-ID to be set is stored
I	CD_DAT u1g_txdata	Global variable in which message data to be set is stored (Data size is set depending on the function to be used.)
I	CD_DLC u1g_txdlc	Global variable in which DLC to be set is stored

[Description]

This driver function sets the following data to the specified message buffer.

- CAN-ID (set in register image)
- Data (Data size is set depending on the function to be used.)
- DLC value (Only the lower 4 bits are valid. The most significant bit indicates the format of CAN-ID.
0: Standard CAN-ID, 1: Extended CAN-ID)

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_STS	CD_E_FLG + 0x2	Setting data has failed
CD_E_NOMSG	CD_E_FLG + 0x4	No new data

[Cautions]

This function can set data only for the transmit buffer.

[Use example]

```

/* The following are set in advance by the configurator.
· One message buffer is assigned for transmission.
  (Buffer name is defined as Ch1_Msg00, using channel 1's message buffer 0.)
· DLC, CAN-ID, and frame format can be arbitrarily set.
· The frame type of the assigned message buffer is set as a data frame
Define the following global variable.
  unsigned char * ulgp_txbuf_addr;
  unsigned char ulg_txid[5];
  CD_DAT      ulg_txdata[8];
  CD_DLC      ulg_txdlc;
The following macro is defined. (Address of message buffer to be set)
#define TXBUFADD_00 (0xf600)
*/

CD_ER ret;

ulgp_txbuf_addr = (unsigned char *) (TXBUFADD_00); /* Message buffer address */
ulg_txdata[0] = 0x00; /* Tx data byte1 */
ulg_txdata[1] = 0x11; /* Tx data byte2 */
ulg_txdata[2] = 0x22; /* Tx data byte3 */
ulg_txdata[3] = 0x33; /* Tx data byte4 */
ulg_txdata[4] = 0x44; /* Tx data byte5 */
ulg_txdata[5] = 0x55; /* Tx data byte6 */
ulg_txdata[6] = 0x66; /* Tx data byte7 */
ulg_txdata[7] = 0x77; /* Tx data byte8 */
ulg_txid[0] = 0x00; /* ID byte0 (IDTX0) */
ulg_txid[1] = 0x00; /* ID byte1 (IDTX1) */
ulg_txid[2] = 0x00; /* ID byte2 (IDTX2) */
ulg_txid[3] = 0x00; /* ID byte3 (IDTX3) */
ulg_txid[4] = 0x00; /* ID byte4 (IDTX4) */
ulg_txdlc = 8; /* DLC */

ret = CanMsgSetIdDatDlc_DS8(); /* Sets transmit data to the message buffer */
if(ret == CD_E_OK){
  ret = CanMsgTxReq(CD_CAN1,Ch1_Msg00); /* Sets the transmit request bit of the message buffer */
  if(ret == CD_E_OK){
    "Processing when transmit request is successful"; /* Sets the transmit request bit */
  }
  else{
    "Processing when transmit request has failed"; /* Setting the transmit request has failed */
  }
}
else{
  "Processing when setting data has failed"; /* Setting data has failed (ret == CD_E_ETS) */
}

```

5.8.5 Transmit/receive confirmation

The driver functions listed in Table 5-12 are described below.

Table 5-12. Transmit/Receive Confirmation

Function	Description
CanMsgTxReq	Transmit request (set TRQ bit)
CanMsgGetTxInfo	Acquires transmit information (acquires TRQ)
CanChSrcRxInfo	Searches receive information (searches DN)
CanChSrcRxInfo_MSxx	Searches receive information (searches DN) (xx = 01 to 16)

Note Functions dedicated to 78K0-DCAN and with improved performance

CanMsgTxReq

[General]

This function is used to set the transmit request bit in the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgTxReq(CD_CHNO chno, CD_BUFNO bufno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number

[Description]

This function is used to set the transmit request bit in the specified channel's message buffer.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_STS	CD_E_FLG + 0x2	Transmit request bit setting failed

[Use example]

See the description of CanMsgSetIdDatDlc

CanMsgGetTxInfo

[General]

This function is used to acquire the transmit request bit in the specified channel's message buffer.

[C language code format]

```
CD_ER CanMsgGetTxInfo(CD_CHNO chno, CD_BUFNO bufno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number

[Description]

This function is used to acquire the transmit request bit in the specified channel's message buffer.

[Return value]

Error Code	Value	Meaning
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_TRUE	1	Transmit request bit has been set
CD_FALSE	0	Transmit request bit has not been set

[Use example]

```
/* The following are set in advance by the configurator.
· One message buffer is assigned for reception.
  (Buffer name is defined as Ch1_Msg00, using channel 1's message buffer 0.)
· Standard ID format is set as the assigned message buffer's frame format.
· Data frame is set as the assigned message's frame type
*/

CD_ER ret;

ret = CanMsgGetTxInfo(CD_CAN1, Ch1_Msg00);
if (ret == CD_TRUE){
    "Processing when transmit request bit has been set";    /* Unsent data remains */
}
else if (ret == CD_FALSE){
    "Processing when transmit request bit has not been set"; /* No unsent data */
}
else{
    "Processing when parameter error occurs";                /* Parameter error (ret == CD_E_PRM) occurred */
                                                            /* while using driver with parameter check function */
}
}
```

CanChSrcRxInfo

[General]

This function is used to search for the data update bit (DN bit) that has been set to the specified channel.

[C language code format]

```
CD_ER CanChSrcRxInfo(CD_CHNO chno, CD_BUFNO bufno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	CD_BUFNO bufno	Buffer number

[Description]

This function is used to search the transmit request bit (TRQ bit) that has been set to the specified channel.

The search proceeds in ascending order, starting from the specified buffer number in the specified channel, and continues until the highest buffer number is reached. The first buffer number that is found is returned as the return value.

[Return value]

Error Code	Value	Meaning
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
CD_E_NOMSG	CD_E_FLG + 0x4	Buffer in specified range with set data update bit (DN bit) was not found
-	When MSB = 0	Buffer number found first

[Use example]

```

/* The following are set in advance by the configurator.
· Message buffers 0 to 15 are assigned for reception.
  (Buffer name is defined as Ch1_Msgxx, using channel 1's message buffer xx.)
· Standard ID format is set as the assigned message buffer's frame format.
· The CAN-ID to be received is set as any value.
· Mask setting (when required)
*/

CD_ER  ret;

ret = CanChSrcRxInfo(CD_CAN1,Ch1_Msg00);          /* DN bit search starts from buffer number 0 */
if (ret == CD_E_NOMSG){
    "Processing when data update bit has not been set"; /* DN bit was not set */
}
else if (ret == CD_E_PRM){
    "Processing when parameter error occurs";          /* Parameter error occurred when using driver */
                                                    /* with parameter check */
}
else{
    switch (ret){
        case 0:
            "Processing when buffer number 0 has been set";
            break;                                /* DN bit is set for buffer number 0 */
        case 1:
            "Processing when buffer number 1 has been set";
            break;                                /* DN bit is set for buffer number 1 */
        case 2:
            "Processing when buffer number 2 has been set";
            break;                                /* DN bit is set for buffer number 2 */
        case 3:
            break;
                                                    /* ..... */
    }
}

```

CanChSrcRxInfo_MSxx (xx = 01 to 16)

[General]

Function dedicated to the 78K0-DCAN and with improved performance.

This function is used to search a data updating (DN) bit that is set.

[C language code format]

```

CD_ER CanChSrcRxInfo_MS01();      (Searched receive buffer size = 1)
CD_ER CanChSrcRxInfo_MS02();      (Searched receive buffer size = 2)
CD_ER CanChSrcRxInfo_MS03();      (Searched receive buffer size = 3)
CD_ER CanChSrcRxInfo_MS04();      (Searched receive buffer size = 4)
CD_ER CanChSrcRxInfo_MS05();      (Searched receive buffer size = 5)
CD_ER CanChSrcRxInfo_MS06();      (Searched receive buffer size = 6)
CD_ER CanChSrcRxInfo_MS07();      (Searched receive buffer size = 7)
CD_ER CanChSrcRxInfo_MS08();      (Searched receive buffer size = 8)
CD_ER CanChSrcRxInfo_MS09();      (Searched receive buffer size = 9)
CD_ER CanChSrcRxInfo_MS10();      (Searched receive buffer size = 10)
CD_ER CanChSrcRxInfo_MS11();      (Searched receive buffer size = 11)
CD_ER CanChSrcRxInfo_MS12();      (Searched receive buffer size = 12)
CD_ER CanChSrcRxInfo_MS13();      (Searched receive buffer size = 13)
CD_ER CanChSrcRxInfo_MS14();      (Searched receive buffer size = 14)
CD_ER CanChSrcRxInfo_MS15();      (Searched receive buffer size = 15)
CD_ER CanChSrcRxInfo_MS16();      (Searched receive buffer size = 16)

```

[Parameters] None

[Global variables]

<R>

I/O	Global Variable Name	Description
O	unsigned char* u1gp_rxbuf_addr	Global variable in which the address of the DSTAT register of the receive buffer that has been found first is stored The result of searching by CanMsgGetDatDlc_DS1() can be used as is. See [Usage example] of CanMsgGetDatDlc_DSx.

[Description]

This driver function searches a data updating (DN) bit that is set.

The DN bit is always searched starting from receive buffer 0 in the ascending order by the search receive buffer size that is determined for each function. The address of the DSTAT register of the receive buffer that has been found first is stored in a global variable.

[Return value]

Error Code	Value	Meaning
CD_E_OK	0x0	Buffer with the data updating (DN) bit set has been found in the searched range.
CD_E_NOMSG	CD_E_FLG + 0x4	No buffer with the data updating (DN) bit set has been found in the searched range.

[Use example]

See CanMsgGetDatDlc_DSx descriptions.

5.8.6 CAN channel status access

The driver functions listed in Table 5-13 are described below.

Table 5-13. CAN Channel Status Access

Function	Description
CanChGetStatus	Acquires CAN channel status
CanChClrStatus	Clears CAN channel status
CanChGetBusStatus	Acquires CAN bus status

CanChGetStatus

[General]

This function is used to acquire the specified channel's CAN status.

[C language code format]

```
CD_ER CanChGetStatus(CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This function is used to acquire the CAN status (wakeup from CAN sleep mode, arbitration lost, CAN protocol error, CAN error status, etc.) and return it as the return value.

[Return value]

Error Code	Value	Meaning
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid
-	When MSB = 0	See Description of bits when MSB = 0

• Description of bits when MSB = 0

Bit Position	Description	Support of Each CAN Controller		
		aFCAN	FCAN	DCAN
Bits 1, 0	Fixed to zero	-	-	-
Bit 2	CAN error status (0: No event pending, 1: Event pending)	Supported	Not supported	Not supported
Bit 3	CAN protocol error (0: No event pending, 1: Event pending)	Supported	Supported	Not supported
Bit 4	Arbitration lost (0: No event pending, 1: Event pending)	Supported	Not supported	Not supported
Bit 5	Wakeup from CAN sleep mode (0: No event pending, 1: Event pending)	Supported	Supported	Supported
Bit 6	CAN overrun error (0: No event pending, 1: Event pending)	Not supported	Supported	Supported
Bit 7	CAN transmission error passive status or bus-off status (0: No event pending, 1: Event pending)	Not supported	Supported	Not supported
Bit 8	CAN reception error passive status (0: No event pending, 1: Event pending)	Not supported	Supported	Not supported
MSB to bit 9	Fixed to zero	-	-	-

<R>

[Use example]

```
CD_ER ret;

ret = CanChGetStatus(CD_CAN1);
if (ret == CD_E_PRM){
    "Processing when parameter error occurs";          /* Parameter error occurred when using driver */
                                                    /* with parameter check */
}
else{
    if ((ret & 0x04) == 0x04){
        "Processing when CAN error status event has been held pending";
    }
    if ((ret & 0x08) == 0x08){
        "Processing when CAN protocol error event has been held pending ";
    }
    if ((ret & 0x10) == 0x10){
        "Processing when arbitration lost event has been held pending";
    }
    if ((ret & 0x20) == 0x20){
        "Processing when wakeup from CAN sleep mode event has been held pending";
    }
}
}
```

CanChClrStatus

[General]

This function is used to clear the specified channel's CAN status.

[C language code format]

```
CD_ER CanChClrStatus(CD_CHNO chno, unsigned char clrdat);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number
I	unsigned char clrdat	Data specified by clear bit setting

[Description]

This function is used to clear the specified channel's CAN status. The CAN status to be cleared is specified by the clear bit specification data.

Clear bit specification data macro definitions are used for the clear bit specification data. When multiple bits are cleared by one function call, a vertical line "|" (logical OR operator) is used to combine the targets' macro definitions.

- Macro definitions for clear bit specification data

Macro	Description	Support of Each CAN Controller		
		aFCAN	FCAN	DCAN
CD_ERR_CLR_STS	CAN error status clear specification macro	Supported	Not supported	Not supported
CD_ERR_CLR_PRT	CAN protocol error status clear specification macro	Supported	Supported	Not supported
CD_ERR_CLR_ABL	Arbitration lost status clear specification macro	Supported	Not supported	Not supported
CD_ERR_CLR_WAK	Wakeup from CAN sleep mode status clear specification macro	Supported	Supported	Supported
CD_ERR_CLR_OVR	CAN overrun error status clear specification macro	Not supported	Supported	Supported
CD_ERR_CLR_TXP	CAN transmission error passive status or bus-off status clear specification macro	Not supported	Supported	Not supported
CD_ERR_CLR_RXP	CAN reception error passive status clear specification macro	Not supported	Supported	Not supported

[Return value]

Error Code	Value	Meaning
CD_E_OK	CD_E_FLG + 0x0	Normal end
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number is invalid

[Use example]

```
CD_ER ret;
unsigned char clrdat;

clrdat = CD_ERR_CLR_STS | CD_ERR_CLR_PRT;
ret = CanChClrStatus(CD_CAN1,clrdat);          /* Error status and protocol status clear specification */
if (ret == CD_E_OK){
    "Processing upon normal end";
}
else{
    "Processing when parameter error occurs";  /* Parameter error (ret = CD_E_PRM) occurred */
                                              /* while using driver with parameter check function */
}
}
```

CanChGetBusStatus

[General]

This function is used to acquire the specified channel's CAN bus status.

[C language code format]

```
CD_ER CanChGetBusStatus(CD_CHNO chno);
```

[Parameters]

I/O	Parameter	Description
I	CD_CHNO chno	Channel number

[Description]

This function is used to acquire the CAN bus status such as the bus-off status and transmit/receive error counter status of the specified channel and return it as the return value.

[Return value]

Error Code	Value	Meaning
CD_E_PRM	CD_E_FLG + 0x1	Specified channel number or buffer number is invalid
-	When MSB = 0	See Description of bits when MSB = 0

• Description of bits when MSB = 0

Bit Position	Description
Bits 1, 0	Receive error counter status bit 00: Receive error counter is below warning level (up to 95) 01: Receive error counter is within warning level range (96 to 127) (Warning level range or error passive range in the case of DCAN controller) 10: Not defined 11: Receive error counter is in error passive range (128 or more) (Undefined in the case of DCAN controller)
Bits 3, 2	Transmit error counter status bit 00: Transmit error counter is below warning level (up to 95) 01: Transmit error counter is within warning level range (96 to 127) (Warning level range, error passive range, or bus-off range in the case of DCAN controller) 10: Not defined 11: Transmit error counter is in error passive or bus-off range (128 or more) (Undefined in the case of DCAN controller)
Bit4	Bus-off status bit 0: Not bus-off status (transmit error counter is less than 256) 1: Bus-off status (transmit error counter is at least 256)
MSB to bit5	Fixed to 0

<R>

<R>

<R>

<R>

[Use example]

```

CD_ER ret;
CD_ER rx_err;
CD_ER tx_err;

ret = CanChGetBusStatus(CD_CAN1);
if (ret == CD_E_PRM){
    "Processing when parameter error occurs"; /* Parameter error occurred */
                                           /* when using driver with parameter check */
}
else{
    rx_err = ret & 0x03;
    switch (rx_err){
        case 0:
            <R>    "Receive error counter is below warning level (up to 95)";
                break;
        case 1:
            <R>    "Receive error counter is within warning level range (96 to 127)";
                break;
        case 3:
            "Receive error counter is in error passive range (128 or more)";
                break;
    }
    tx_err = (ret >> 2) & 0x03;
    switch (rx_err){
        case 0:
            <R>    "Transmit error counter is below warning level (up to 95)";
                break;
        case 1:
            <R>    "Transmit error counter is within warning level range (96 to 127)";
                break;
        case 3:
            "Transmit error counter is in error passive range (128 or more)";
                break;
    }
    if ((ret & 0x10) == 0x10){
        "Bus-off status (transmit error counter is at least 256)";
    }
    else{
        "Not bus-off status";
    }
}

```

6.1 V850ES/FJ2

This chapter explains the V850ES/FJ2 sample program.

6.1.1 Operation environment

Target device: μ PD70F3239 (V850ES/FJ2)

Target board: CEB-V850ES/FJ2-SJ2 (made by COSMO)

6.1.2 Overview of operation

- When device initialization processing ends normally, the 7seg-LED display changes from "0" to "1".
- Receive data is detected upon an interrupt.
- Data is transmitted in the cycles generated by timer M.
- The first byte of the receive data is displayed in the 8bit-LED each time data is received.
- Transmit data varies per transmission.

First time : 8-byte data consisting of 0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, and 0x07

Second time : 8-byte data consisting of 0x08, 0x09, 0x0a, 0x0b, 0x0c, 0x0d, 0x0e, and 0x0f

Third time : 8-byte data consisting of 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, and 0x17

:
:
:

6.1.3 Items preset by configurator

The configurator uses the following setting information to generate setting files.

- Name of generated file
Information file: config.c
Header file: config.h

```

Device selection : Device Name          uPD70F3239 (V850/FJ2)
                  : CAN Clock           20MHz
                  : CAN register area   0x03FEC000
                  : Channel Select      Channel 1 only
Baud rate setting : Can module system clock 20MHz
                  : Baud rate           500Kbps
                  : Data bit setting
                    Prescaler           2
                    DBT                  20
                    SPT                  15
                    Sample point        75
                    SJW                  2
Mask Setting      : None
Buffer Setting(Tx) : Transmit message buffer
                   Buffer name          TxData_00
                   Buffer No            00
                   CAN ID              100
                   ID Type             Std
                   DLC                  8
                   Interrupt           Disable
                   Flame Type          Data
(Rx) : Receive message buffer
       Buffer name          RxData_00
       Buffer No            01
       Mask                None
       CAN ID              200
       ID Type             Std
       Interrupt           Enable
       Flame Type          Data
       If DN-bit=1        Overwrite
Other setting      : Default setting

```

6.1.4 Sample program (for NEC Electronics tool)

Header file: sample.h

```

/*
 * #include
 */
#include <candrv.h>                                /* Header file for CAN software driver */

/* Wait macro */
#define      WAIT( val )      {                                \
                                                    \
                unsigned int   ctr ;                        \
                                                    \
                for( ctr = ( unsigned int )( 0 ) ; ctr < val ; ctr++ ) ; \
                                                    \
            }

/* Display wait */
#define      WAIT_DISP      ( ( unsigned int )( 2000000 ) )

/* 8bit LED port info. */
#define      OUT_LED_7_2      ( P6H )
#define      OUT_LED_1_0      ( PCT )

/* LED mask info */
#define      MSK_LED_7_2      ( ( unsigned char )( 0x03 ) )
#define      MSK_LED_1_0      ( ( unsigned char )( 0xf3 ) )

/* 7seg LED port info. */
#define      SFR_LED0_L      ( PCD )
#define      SFR_LED0_H      ( PCS )

/* individual segments (0:on, 1:off) */
#define      SEG7_A          ( ( unsigned char )( ~0x01 ) )
#define      SEG7_B          ( ( unsigned char )( ~0x02 ) )
#define      SEG7_C          ( ( unsigned char )( ~0x04 ) )
#define      SEG7_D          ( ( unsigned char )( ~0x08 ) )
#define      SEG7_E          ( ( unsigned char )( ~0x10 ) )
#define      SEG7_F          ( ( unsigned char )( ~0x20 ) )
#define      SEG7_G          ( ( unsigned char )( ~0x40 ) )
#define      SEG7_DP         ( ( unsigned char )( ~0x80 ) )

/* all segments ON and OFF */
#define      LED_ON          ( ( unsigned char )( 0x00 ) )
#define      LED_OFF         ( ( unsigned char )( 0xff ) )

/* pattern */

```



```

#define LED_PAT_0 ( SEG7_A&SEG7_B&SEG7_C&SEG7_D&SEG7_E&SEG7_F )
#define LED_PAT_1 ( SEG7_B&SEG7_C )
#define LED_PAT_2 ( SEG7_A&SEG7_B& SEG7_D&SEG7_E& SEG7_G )
#define LED_PAT_3 ( SEG7_A&SEG7_B&SEG7_C&SEG7_D& SEG7_G )
#define LED_PAT_4 ( SEG7_B&SEG7_C& SEG7_F&SEG7_G )
#define LED_PAT_5 ( SEG7_A& SEG7_C&SEG7_D& SEG7_F&SEG7_G )
#define LED_PAT_6 ( SEG7_A& SEG7_C&SEG7_D&SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_7 ( SEG7_A&SEG7_B&SEG7_C& SEG7_F )
#define LED_PAT_8 ( SEG7_A&SEG7_B&SEG7_C&SEG7_D&SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_9 ( SEG7_A&SEG7_B&SEG7_C&SEG7_D& SEG7_F&SEG7_G )
#define LED_PAT_A ( SEG7_A&SEG7_B&SEG7_C& SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_B ( SEG7_C&SEG7_D&SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_C ( SEG7_A& SEG7_D&SEG7_E&SEG7_F )
#define LED_PAT_D ( SEG7_B&SEG7_C&SEG7_D&SEG7_E& SEG7_G )
#define LED_PAT_E ( SEG7_A& SEG7_D&SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_F ( SEG7_A& SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_H ( SEG7_B&SEG7_C& SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_G ( SEG7_A&SEG7_B& SEG7_D&SEG7_E&SEG7_F )
#define LED_PAT_J ( SEG7_B&SEG7_C&SEG7_D )
#define LED_PAT_L ( SEG7_D&SEG7_E&SEG7_F )
#define LED_PAT_N ( SEG7_C& SEG7_E& SEG7_G )
#define LED_PAT_O ( SEG7_C&SEG7_D&SEG7_E& SEG7_G )
#define LED_PAT_P ( SEG7_A&SEG7_B& SEG7_E&SEG7_F&SEG7_G )
#define LED_PAT_Q ( SEG7_A&SEG7_B&SEG7_C& SEG7_F&SEG7_G )
#define LED_PAT_R ( SEG7_E& SEG7_G )
#define LED_PAT_U ( SEG7_B&SEG7_C&SEG7_D&SEG7_E&SEG7_F )

```

Source file: sample.c

```

/*
 * #include
 */
#include "sample.h" /* Header file for can sample program */
#include "config.h" /* Configuration file */

/*
 * #pragma
 */
#pragma ioreg
#pragma interrupt INTTP0CC0v0i_candrv_tx /* Set interrupt vector address of Timer-P */
#pragma interrupt INTCOREC v0i_candrv_rx /* Set interrupt vector address of CAN0 receive */

/*
 * Prototype declaration
 */
void main( void ) ;
static void v0s_candrv_init( void ) ;
static void v0i_candrv_tx( void ) ;

```

```

static void      v0i_candrv_rx( void ) ;
static void      v0s_start( void ) ;
static void      v0s_ledout( unsigned char ) ;
static void      v0s_7segout( unsigned char ) ;
static void      v0s_error( void ) ;

/*
 * Main function
 */
void
main( void )
{

    v0s_start() ;                               /* Initialize sample program */
    v0s_7segout( LED_PAT_0 ) ;                  /* Output 7seg-LED */
    v0s_candrv_init() ;                         /* Initialize CAN */

    TP0CE = 1 ;                                /* Start TMP0 */

    while( 1 ){                                /* Main loop */

        v0s_7segout( LED_PAT_1 ) ;             /* Output 7seg-LED */

    }

}

/*
 * CAN initialize operation
 * (CAN software driver)
 */
static
void
v0s_candrv_init( void )
{

    CD_ER  u4t_ret ;

    u4t_ret = CanChEnable( CD_CAN1 ) ;         /* Enable CAN1 */
    if( u4t_ret != CD_E_OK ){

        v0s_error() ;

    }

    u4t_ret = CanChInit( CD_CAN1 ) ;          /* Initialize CAN1 */
    if( u4t_ret != CD_E_OK ){

```

```

        v0s_error() ;

    }

    u4t_ret = CanChSetNrmMode( CD_CAN1 ) ;           /* Set normal mode CAN1 */
    if( u4t_ret != CD_E_OK ){

        v0s_error() ;

    }

    return ;
}

/*
 * CAN transmit interrupt operation
 * (CAN software driver)
 */
__interrupt
void
v0i_candrv_tx( void )
{

    CD_ER          u4t_ret ;
    CD_DAT         ult_TxDatabuf[ 8 ] ;
    static unsigned char  ult_txdata = ( unsigned char )( 0x00 ) ;

    u4t_ret = CanMsgGetTxInfo( CD_CAN1, TxData_00 ) ;   /* Check TRQ bit */
    if( u4t_ret == CD_TRUE ){

        return ;

    }

    {

        /* Set Tx data */

        signed int  s4t_ctr;

        for( s4t_ctr = ( signed int )( 0 ) ; s4t_ctr < ( signed int )( 8 ) ; s4t_ctr++ ){

            ult_TxDatabuf[ s4t_ctr ] = ult_txdata++ ;

        }

    }

}

/* Set Tx message */

```

```

u4t_ret = CanMsgSetDat( CD_CAN1, TxData_00, &ult_TxDatabuf[ 0 ] ) ;
if( u4t_ret != CD_E_OK ){

    v0s_error() ;

}

u4t_ret = CanMsgTxReq( CD_CAN1, TxData_00 ) ;          /* Set Tx request */
if( u4t_ret != CD_E_OK ){

    v0s_error() ;

}

return ;
}

/*
 * CAN receive interrupt operation
 * (CAN software driver)
 */
__interrupt
void
v0i_candrv_rx( void )
{

    CD_ER    u4t_ret ;
    CD_DAT   ult_RxDatabuf[ 8 ] ;
    CD_DLC   s1t_RxDlc ;

    u4t_ret = CanChSrcRxInfo( CD_CAN1, RxData_00 ) ;    /* Search Rx message */
    if( ( u4t_ret & CD_E_FLG ) == ( unsigned int )( 0x00000000 ) ){

        CD_BUFNO    ult_bufno ;

        ult_bufno   = u4t_ret;

                                                /* Get Rx message */
        u4t_ret = CanMsgGetDatDlc( CD_CAN1, ult_bufno, &ult_RxDatabuf[ 0 ], &s1t_RxDlc ) ;
        if( u4t_ret != CD_E_OK ){

            v0s_error() ;

        }

        v0s_ledout( ult_RxDatabuf[ 0 ] ) ;            /* Display Rx message */

    }
}

```

```

    return ;
}

/*
 * Device initialization function
 */
static
void
v0s_start( void )
{
extern signed int    _rcopy( unsigned long *, signed long ) ;
extern unsigned long _S_romp;

    VSWC      = ( unsigned char )( 0x01 ) ;          /* System clock = 20MHz */
    RCM       = ( unsigned char )( 0x01 ) ;          /* Stop ring-OSC */
    WDTM2     = ( unsigned char )( 0x00 ) ;          /* Stop WDT2 */
    BPC       = ( unsigned short )( 0x8ffb ) ;       /* Set programmable peripheral I/O area */

    ( void )_rcopy( &_S_romp, ( signed long )( -1 ) ) ; /* Copy ".data" section in ROM to RAM */

    v0s_7segout( LED_OFF ) ;                          /* clear 7seg-LED */
    v0s_ledout( ( unsigned char )( 0x00 ) ) ;          /* clear 8bit-LED */

    PMCD      = ( unsigned char )( 0xf0 ) ;           /* Port CD (7seg-LED) */
    PMCCS     = ( unsigned char )( 0x00 ) ;           /* Port CS (7seg-LED) */
    PMCS      = ( unsigned char )( 0x00 ) ;
    PMCCT     = ( unsigned char )( 0x00 ) ;           /* Port CT (8bit-LED) */
    PMCT      = ( unsigned char )( 0x00 ) ;
    PMC6H     = ( unsigned char )( 0x00 ) ;           /* Port 6H (8,1bit-LED) */
    PM6H      = ( unsigned char )( 0x00 ) ;

                                                    /* Port setting for CAN1 */
    PFC3L     &= ( unsigned char )( 0xe7 ) ;          /* Clear PFC33,34 */
    PFCE3L    |= ( unsigned char )( 0x18 ) ;          /* Set PFCE33,34 */
    PMC3L     |= ( unsigned short )( 0x00d8 ) ;       /* Set PMC37,36,34,33 */

                                                    /* Setup Timer P */
    TP0CE     = 0 ;                                  /* Stop TMP0 */
    TP0CTL0   = ( unsigned char )( 0x07 ) ;
    TP0CTL1   = ( unsigned char )( 0x00 ) ;
    TP0IOC0   = ( unsigned char )( 0x00 ) ;
    TP0IOC1   = ( unsigned char )( 0x00 ) ;
    TP0IOC2   = ( unsigned char )( 0x00 ) ;
    TP0OPT0   = ( unsigned char )( 0x00 ) ;
    TP0CCR0   = ( unsigned short )( 0xffff ) ;        /* TMP0 : 419.424(msec) */
}

```

```

                                                                    /*      = ( 1 / ( 20MHz / 128 ) * 65535 */
SELCNT0   = ( unsigned char )( 0x00 ) ;
SELCNT1   = ( unsigned char )( 0x00 ) ;

__EI() ;                                                                    /* Enable global interrupt */
CORECMK   = 0 ;                                                                /* Enable receive complete interrupt */
TPOCCMK0  = 0 ;                                                                /* Enable Timer-P interrupt */

}

/*
 * 8bit-LED port output function
 */
static
void
v0s_ledout(                                                                    /* 8bit-LED output */
    unsigned char  ult_dat
)
{

    OUT_LED_7_2 = ( ( ~ult_dat ) | MSK_LED_7_2 ) ;    /* 8bit-LED : bit7-bit2 */
    OUT_LED_1_0 = ( ( ( ~ult_dat ) << 2 ) | MSK_LED_1_0 ) ;    /* 8bit-LED : bit1-bit0 */

    return ;

}

/*
 * 7seg-LED port output function
 */
static
void
v0s_7segout(                                                                    /* 7seg-LED output */
    unsigned char  ult_pat
)
{

                                                                    /* 7seg-LED : a - d */
    SFR_LED0_L = ( ( unsigned char )( 0xf0 ) | ult_pat & ( unsigned char )( 0x0f ) ) ;
                                                                    /* 7seg-LED : e - dot */
    SFR_LED0_H = ( ( unsigned char )( 0x0f ) | ult_pat & ( unsigned char )( 0xf0 ) ) ;

    return ;

}

/*
 * Error function

```

```
*/
static
void
v0s_error( void )                /* Error message output */
{
    while( 1 ){
        v0s_7segout( LED_PAT_E ) ;          /* Output 7seg-LED "E" */
        WAIT( WAIT_DISP ) ;
        v0s_7segout( LED_PAT_R ) ;          /* Output 7seg-LED "R" */
        WAIT( WAIT_DISP ) ;
        v0s_7segout( LED_PAT_R ) ;          /* Output 7seg-LED "R" */
        WAIT( WAIT_DISP ) ;
        v0s_7segout( LED_PAT_O ) ;          /* Output 7seg-LED "O" */
        WAIT( WAIT_DISP ) ;
        v0s_7segout( LED_PAT_R ) ;          /* Output 7seg-LED "R" */
        WAIT( WAIT_DISP ) ;
        v0s_7segout( LED_OFF ) ;           /* Output 7seg-LED " " */
        WAIT( WAIT_DISP ) ;
    }
}
```

APPENDIX REVISION HISTORY

The mark <R> shows major revised points.

The revised points can be easily searched by copying an "<R>" in the PDF file and specifying it in the "Fine what:" field.

APP.1 Main Revisions in this Edition

(1/2)

Page	Description
p.11	Modification of description to 1.3 Types of CAN Software Drivers
pp.12, 13	Modification of description to 1.4 Execution Environment
p.14	Modification of description to 1.5 Development Environment
p.15	Modification of Table 2-1. Provision of CAN Software Drivers Modification of description to 2.2.2 Media setting Deletion of 2.2.3 Installation of CAN software driver
p.16	Modification of Figure 2-1. Directory Structure for CAN Software Drivers , and deletion of Remark
p.17	Modification of description to 2.3.3 Sample programs Modification of Figure 2-2. Directory Structure of Sample Programs
p.25	Modification of Figure 4-1. Main Screen
p.26	Modification of description to 4.4.1 Device selection
p.27	Modification of Figure 4-2. Device Selection Menu Screen
p.29	Modification of Figure 4-3. Baud Rate Setting Screen (V850-aFCAN, V850-DCAN, 78K0-aFCAN)
p.31	Modification of Figure 4-4. Baud Rate Setting Screen (V850-FCAN)
p.33	Modification of Figure 4-5. Baud Rate Setting Screen (78K0-DCAN)
p.60	Modification of Figure 4-21. Output Options Setting Screen
p.61	Modification of Figure 4-22. Code Generation Startup Screen
p.62	Modification of Figure 4-23. Screen for Saving and Opening Project Files
pp.63 to 65	Modification of Table 4-1. Error Code List
p.66	Modification of Table 4-2. Warning Code List
p.68	Modification of 5.1.1 Initialization and setting (6 types) Modification of 5.1.2 Operation modes (3 types)
p.74	Modification of Table 5-6. Single-Channel Specification CAN Software Driver Functions
p.78	Modification of Table 5-8. Initialization and Setting
p.79	5.8.1 Initialization and setting CanChEnable Deletion of Note in [General] Modification of Note 1 and deletion of Note 2 in [Description]
p.85	5.8.1 Initialization and setting Addition of CanChShutdown
p.86	5.8.1 Initialization and setting Addition of CanAllShutdown
p.87	Modification of Table 5-9. Operation Modes
p.91	5.8.2 Operation modes Addition of Note in the CanChSetInitMode
p.98	5.8.3 Buffer data acquisition Modification of [Use example] in the CanMsgGetDatDlc_DSx

Page	Description
p.99	5.8.3 Buffer data acquisition Modification of [Global variables] in the CanMsgGetIdDatDlc_DSx
p.115	5.8.5 Transmit/receive confirmation Modification of [Global variables] in the CanChSrcRxInfo_MSxx
p.118	5.8.6 CAN channel status access Modification of • Description of bits when MSB = 0 in the CanChGetStatus
pp.122, 123	5.8.6 CAN channel status access Modification of • Description of bits when MSB = 0, and [Use example] in the CanGetBusStatus
p.124	Modification of CHAPTER 6 SAMPLE PROGRAM

APP.2 Revision History of Preceding Editions

Here is the revision history of the preceding editions. Chapter indicates the chapter of each edition.

(1/2)

Edition	Description	Chapter
2nd	Addition of V850/SF1, V850/DB1, μ PD780824B(A), 780826B(A), 780828B(A), 78F0828B, 78F0876 for target device. Modification of driver library for driver source file.	Throughout
	Modification of Documents Related to Devices in INTRODUCTION Modification of Documents Related to Development Tools in INTRODUCTION	
	Modification of 1. 3 Types of CAN Software Drivers	CHAPTER 1
	Modification of 1. 4 (3) (a) Memory capacity of total functions	PRODUCT OVERVIEW
	Modification of 1. 4 (3) (b) Memory capacity of functions used in sample program	
	Modification of Figure 2-1. Directory Structure for CAN Software Drivers	CHAPTER 2
	Modification of Figure 2-2. Directory Structure of Sample Programs	INSTALLATION
	Modification of Figure 3-2. System Building Steps	CHAPTER 3 SYSTEM
	Modification of 3. 2. 1 (2) Configurator's output data	BUILD
	Modification of Figure 3-3. Correlations Between Application Program and CAN Software Driver/Configurator	
	Modification of 3. 2. 4 Creation of load module files	
	Modification of Figure 4-1. Main Screen	CHAPTER 4
	Modification of Figure 4-2. Device Selection Menu Screen	CONFIGURATION
	Modification of 4. 4. 2 Baud rate setting	
	Modification of 4. 4. 3 Mask settings	
	Modification of 4. 4. 4 Message buffer settings	
	Modification of 4. 4. 5 Other settings	
	Modification of 4. 4. 6 (1) Output option setting for CAN software driver source files	
	Modification of Figure 4-23. Screen for Saving and Opening Project Files	
	Modification of 4. 5 Error/Warning Message List	
	Addition of function in 5. 1. 1 Initialization and setting (4 types) Addition of function in 5. 1. 3 Buffer data acquisition (4 types) Addition of function in 5. 1. 4 Buffer data setting (4 types) Addition of function in 5. 1. 5 Transmit/receive confirmation (4 types)	CHAPTER 5 DRIVER
	Addition of macro in Table 5-3. Macros for Parameters	FUNCTIONS
	Modification of 5. 5 Single-Channel Specification CAN Software Driver Functions	
	Modification of 5. 6 CAN Software Driver Functions with Improved Performance	
	5. 8. 1 Initialization and setting Modification of the following functions. CanChEnable, CanChInit	
	5. 8. 2 Operation modes Addition of Note in the CanChGetMode.	
	5. 8. 3 Buffer data acquisition Modification of the following functions. CanMsgGetDatDlc, CanMsgGetIdDatDlc	

Edition	Description	Chapter
2nd	5. 8. 4 Buffer data setting Modification of the following functions. CanMagSetDat, CanMsgSetIdDatDlc	CHAPTER 5 DRIVER FUNCTIONS
	5. 8. 6 CAN channel status access Modification of the following functions. CanChGetStatus, CanChClrStatus, CanGetBusStatus	
	Modification of 6. 1. 3 Settings by Startup File (V850ES/FJ2 Device Dependencies)	CHAPTER 6 SAMPLE PROGRAM

*For further information,
please contact:*

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668,
Japan
Tel: 044-435-5111
<http://www.necel.com/>

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554, U.S.A.
Tel: 408-588-6000
800-366-9782
<http://www.am.necel.com/>

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211-65030
<http://www.eu.necel.com/>

Hanover Office
Podbielskistrasse 166 B
30177 Hannover
Tel: 0 511 33 40 2-0

Munich Office
Werner-Eckert-Strasse 9
81829 München
Tel: 0 89 92 10 03-0

Stuttgart Office
Industriestrasse 3
70565 Stuttgart
Tel: 0 711 99 01 0-0

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908-691-133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01-3067-5800

Sucursal en España
Juan Esplandiu, 15
28007 Madrid, Spain
Tel: 091-504-2787

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 638 72 00

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02-667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven
The Netherlands
Tel: 040 265 40 10

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27 ZhiChunLu Haidian
District, Beijing 100083, P.R.China
Tel: 010-8235-1155
<http://www.cn.necel.com/>

Shanghai Branch
Room 2509-2510, Bank of China Tower,
200 Yincheng Road Central,
Pudong New Area, Shanghai, P.R.China P.C:200120
Tel:021-5888-5400
<http://www.cn.necel.com/>

Shenzhen Branch
Unit 01, 39/F, Excellence Times Square Building,
No. 4068 Yi Tian Road, Futian District, Shenzhen,
P.R.China P.C:518048
Tel:0755-8282-9800
<http://www.cn.necel.com/>

NEC Electronics Hong Kong Ltd.
Unit 1601-1613, 16/F., Tower 2, Grand Century Place,
193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: 2886-9318
<http://www.hk.necel.com/>

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R. O. C.
Tel: 02-8175-9600
<http://www.tw.necel.com/>

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253-8311
<http://www.sg.necel.com/>

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku,
Seoul, 135-080, Korea
Tel: 02-558-3737
<http://www.kr.necel.com/>