RL78 Family

EEPROM Emulation Library Pack01
Japanese Release

ZIP file name: JP_R_EEL_RL78_P01_Vx.xx_x_E

16-Bit Single-Chip Microcontrollers

All information contained in these materials, including products and product specifications, represents information on the product at the time of publication and is subject to change by Renesas Electronics Corporation without notice. Please review the latest information published by Renesas Electronics Corporation through various means, including the Renesas Electronics Corporation website (http://www.renesas.com).

Renesas Electronics
www.renesas.com

Rev.1.03 Jan 2016
Notice

1. 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.

2. 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.

3. 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.

4. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from such alteration, modification, copy or otherwise misappropriation of Renesas Electronics product.

5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The recommended applications for each Renesas Electronics product depends on the product’s quality grade, as indicated below.

- **“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 etc.
- **“High Quality”**: Transportation equipment (automobiles, trains, ships, etc.); traffic control systems; anti-disaster systems; anti-crime systems; and safety equipment etc.

Renesas Electronics products are neither intended nor authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems, surgical implantations etc.), or may cause serious property damages (nuclear reactor control systems, military equipment etc.). 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 for which it is not intended. 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 which the product is not intended by Renesas Electronics.

6. 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.

7. 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 systems manufactured by you.

8. 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.

9. 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. You should not use Renesas Electronics products or 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. When exporting the Renesas Electronics 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.

10. It is the responsibility of the buyer or distributor of Renesas Electronics products, who distributes, disposes of, or otherwise places the product with a third party, to notify such third party in advance of the contents and conditions set forth in this document, Renesas Electronics assumes no responsibility for any losses incurred by you or third parties as a result of unauthorized use of Renesas Electronics products.

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.
General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins
   Handle unused pins in accordance with the directions given under Handling of Unused Pins in the manual.
   - The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on
   The state of the product is undefined at the moment when power is supplied.
   - The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   - In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   - In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses
   Access to reserved addresses is prohibited.
   - The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals
   After applying a reset, only release the reset line after the operating clock signal has become stable.
   - When switching the clock signal during program execution, wait until the target clock signal has stabilized.
   - When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products
   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.
   - The characteristics of Microprocessing unit or Microcontroller unit products in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.
HOW TO USE THIS MANUAL

Readers
This manual is intended for users who wish to understand the functions of the RL78 microcontrollers EEPROM Emulation Library Pack 01 and design and develop application systems and programs for these devices.

< R >
For the target MCUs, refer to the RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944EJxxxx).

Purpose
This manual is intended to give users an understanding of the methods (described in the Organization below) for using data flash memory library to rewrite the flash data memories.

Organization
The RL78 EEPROM Emulation Library Pack 01 user’s manual is separated into the following parts:

• Overview of EEPROM Emulation
• Using EEPROM Emulation
• EEPROM Emulation Function

How to Read This Manual
It is assumed that the readers of this manual have general knowledge of electrical engineering, logic circuits, and microcontrollers.

• To gain a general understanding of functions:
  → Read this manual in the order of the CONTENTS.
• To know details of the RL78 Microcontroller instructions:
  → Refer to CHAPTER 3 EEPROM EMULATION FUNCTION.

The mark < R > shows major revised points.

Conventions
Data significance: Higher digits on the left and lower digits on the right
Active low representations: ××× (overscore over pin and signal name)
Note: Footnote for item marked with Note in the text
Caution: Information requiring particular attention
Remark: Supplementary information
Numerical representations: Binary  ...×××× or ××××B
Decimal  ...××××
Hexadecimal  ...××××H

All trademarks and registered trademarks are the property of their respective owners.
EEPROM is a trademark of Renesas Electronics Corporation.
CONTENTS

CHAPTER 1 OVERVIEW OF EEPROM EMULATION ................................................................. 1
  1. 1 Basic Specifications of EEPROM Emulation ................................................................. 1
  1. 2 EEPROM Emulation Operation Flow ........................................................................... 8
    1. 2. 1 EEPROM Emulation Blocks .................................................................................. 11
    1. 2. 2 Data Structure ...................................................................................................... 12
    1. 2. 3 Block Status Flags ............................................................................................... 14
    1. 2. 4 Number of Stored User Data Items and Total User Data Size ............................... 15
  1. 3 Initializing EEPROM Emulation Blocks ...................................................................... 17
  1. 4 Adjusting EEPROM Emulation Blocks ....................................................................... 21
    1. 4. 1 Adjusting Blocks by Using the EEL_CMD_CLEANUP Command ............................ 23
    1. 4. 2 Adjusting Blocks by Using the EEL_Handler Function (Maintenance Mode) ....... 25

CHAPTER 2 USING EEPROM EMULATION ........................................................................... 30
  2. 1 Caution Points .............................................................................................................. 30
  2. 2 Total Processing Time ................................................................................................. 33
  2. 3 Software Resources .................................................................................................... 36
  2. 4 Initial Values to be Set by User ................................................................................... 39

CHAPTER 3 EEPROM EMULATION FEATURES .................................................................... 42
  3. 1 Data Flash Library Functions ....................................................................................... 42
    FAL_Init ......................................................................................................................... 43
  3. 2 EEPROM Emulation Library Functions ...................................................................... 46
    EEL_Init ......................................................................................................................... 47
    EEL_Open ...................................................................................................................... 48
    EEL_Close .................................................................................................................... 49
    EEL_Execute ............................................................................................................... 50
    EEL_Handler .............................................................................................................. 55
    EEL_TimeOut_CountDown ........................................................................................... 57
    EEL_GetDriverStatus ................................................................................................. 58
    EEL_GetSpace ........................................................................................................... 60
    EEL_GetVersionString ............................................................................................... 61

APPENDIX A REVISION HISTORY ....................................................................................... 62
  A. 1 Major Revisions in This Edition ............................................................................... 62
  A. 2 Revision History of Previous Editions ..................................................................... 63
CHAPTER 1 OVERVIEW OF EEPROM EMULATION

1.1 Basic Specifications of EEPROM Emulation

EEPROM emulation is a feature used to store data in the on-board flash memory in the same way as EEPROM. During EEPROM emulation, the data flash library and EEPROM emulation library are used, and the data flash memory is written to and read from.

The data flash library is a software library used to perform operations on the data flash memory. The EEPROM emulation library is a software library used to execute EEPROM emulation from a user-created program. The data flash library and EEPROM emulation library are placed in the code flash memory for use.

By calling the user access function processing (functions) provided by the EEPROM emulation library from a user-created program, use is possible without the awareness of data flash memory operations.

For the EEPROM emulation library Pack01, a one-byte identifier (data ID: 1 to 255) is assigned by the user for each data item, and reading and writing using any unit from 1 to 255 bytes are possible on an assigned identifier basis. (Up to 255 data items assigned on an identifier basis can be handled.)

Note that four or more continuous block areas of data flash memory are used to store the data. These blocks are called EEPROM emulation blocks.

Data written by EEPROM emulation is divided into reference data and user-specified data, and the reference data is written to the target blocks from the lower block address, while the user data is written from the higher block address.

Figure 1-1 shows the relationship between the EEPROM emulation library and data flash library. Figures 1-2 and 1-3 show a memory map and data structure example, and Figures 1-4, 1-5, and 1-6 show block usage method and transition examples. Table 1-1 shows each item to be specified for EEPROM emulation and the range of the item.
Figure 1-1. Relationship between EEPROM Emulation Library and Data Flash Library

The data flash library must be initialized before using the EEPROM emulation library (by executing the FAL_Init function).

Figure 1-2. Example of Memory Map

- The following shows an example for the R5F100 where the user-created program, data flash library, and EEPROM emulation library are placed in the code flash memory, the EEPROM emulation blocks are specified for the data flash memory, and the defined user data (user data A, user data B, and user data C) are written in order to use the data flash memory as EEPROM emulation blocks.
In this data structure example, user data of various defined sizes (user data A, user data B, and user data C) are written in a specified sequence (write sequence: user data A → user data B → user data A → user data C). User data is written from the higher address and management data is written from the lower address, and the last written user data becomes valid.
Figure 1-4. Example of Expanding Active EEPROM Emulation Block

- When writing specified data, if there is not enough free space in the active block being used to write the data, the number of active blocks is increased and the specified data is written to the new active block.
- For EEPROM emulation, blocks 0 to 3 of the data flash memory are added in order as active or prepared blocks, and then, when the last block (block 3) is reached, the first block (block 0) is specified as the next block to loop through the blocks.

**Figure 1-5. EEPROM Emulation Block Usage Method (Four Blocks)**

Active block: The currently used block
Prepared block: A block that has been prepared and is ready for use
Figure 1-6 (A). EEPROM Emulation Block Transition Example

- The following shows an example of EEPROM emulation blocks to which nothing was written after initializing them.

![Diagram of EEPROM Emulation Block Transition Example](image)

Figure 1-6 (B). EEPROM Emulation Block Transition Example

- If only writing processing is executed continuously, the number of active blocks is increased until a certain number of blocks (the number of active blocks such that there are two or fewer prepared blocks remaining) is reached.

![Diagram of EEPROM Emulation Block Transition Example](image)
Figure 1-6 (C). EEPROM Emulation Block Transition Example

- If only writing processing is executed continuously during EEPROM emulation, and the number of active blocks is increased when two or fewer prepared blocks remain, the valid data in the oldest active block is copied to the latest active block, and the oldest active block is erased.

![Diagram showing EEPROM Emulation Block Transition Example](image)

Table 1-1. Settings of Written Data and Usable Ranges

<table>
<thead>
<tr>
<th>Item</th>
<th>Range</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>User data length</td>
<td>1 to 255</td>
<td></td>
</tr>
<tr>
<td>Amount of stored user data</td>
<td>1 to 255</td>
<td>Number of data types</td>
</tr>
<tr>
<td>Data ID range</td>
<td>1 to 255</td>
<td></td>
</tr>
<tr>
<td>Number of EEPROM emulation blocks</td>
<td>4 to 255</td>
<td></td>
</tr>
<tr>
<td>Recommended user data size</td>
<td>980 \times \text{total number of blocks} \times 1/4 – 980/2 bytes</td>
<td>This also includes the management reference data provided during writing.</td>
</tr>
</tbody>
</table>

Notes 1. The total size of the user data must be such that it is possible to write all the data into within two EEPROM emulation blocks. Therefore, the range used for the number of stored user data items differs depending on the size of the stored user data. It is also necessary to consider the size of the reference data provided for each data item for management use when determining the total size. For details about the number of stored user data items and total size, see 1.2.4 Number of Stored User Data Items and Total User Data Size.

2. EEPROM emulation blocks cannot be set more than maximum number of blocks of on-board data flash memory.
1.2 EEPROM Emulation Operation Flow

To use EEPROM emulation from a user-created program, it is necessary to initialize the EEPROM emulation library and execute functions that perform operations such as reading and writing on EEPROM emulation blocks. Figure 1-7 shows the overall status transitions, and Figure 1-8 shows an operation flow for using basic features. When using EEPROM emulation, incorporate EEPROM emulation into user-created programs by following this flow.

Figure 1-7. EEPROM Emulation Status Transitions
[Overview of status transitions]
To use EEPROM emulation library to manipulate the data flash memory, it is necessary to execute the provided functions in order to advance the processing.

(1) uninitialized
   This is the status after turning the power on or resetting. The system also transitions to this status after executing flash self programming library processing.

(2) closed
   This is the status in which the data has been initialized (the status in which operations on the data flash memory are stopped) to execute the FAL_Init() and EEL_Init() functions and then EEPROM emulation. To execute flash self programming library, STOP mode, or HALT mode processing after executing EEPROM emulation, execute EEL_Close in the opened status to switch to the closed status.

(3) opened
   This status is switched to by executing EEL_Open in the closed status and makes it possible to perform operations on the data flash memory. It is not possible to execute flash self programming library, STOP mode, or HALT mode processing until EEL_Close is executed and the system switches to the closed status.

(4) started
   This status is switched to by executing the EEL_CMD_STARTUP command in the opened status and makes it possible to execute EEPROM emulation. Writes and reads that use EEPROM emulation are performed in this status.

(5) busy
   This is the status used when executing a specified command. The status that is switched to differs depending on which command is executed and how it terminates.
Figure 1-8. Basic Operation Flow of EEPROM Emulation (when Using Enforced Mode)

START

FAL_Init() <1>

EEL_Init() <2>

Start of EEPROM emulation

EEL_Open() <3>

EEL_Execute() EEL_CMD_STARTUP command <4>

Executable status loop

Processing (switch)

Write

EEL_Execute() EEL_CMD_WRITE command <5>

EEL_Execute() EEL_CMD_READ command <6>

Can continue as executable when there is no need to end.

EEL_Execute() EEL_CMD_SHUTDOWN command <8>

EEL_Close() <9>

End of operation

End processing

Read

EEL_Execute() EEL_CMD_READ command <7>

Main processing

<1>

<2>

<3>

<4>

<5>

<6>

<7>

<8>

<9>

<10>

RAM reset required when restarting?

Yes

No
[Overview of basic operation flow]

For EEPROM emulation, the method for executing the EEL_Execute function differs depending on the mode setting.

The following three modes are available: the enforced mode, timeout mode, and polling mode. For details about the differences in the execution method for each mode, see EEL_Execute function in 3.2 EEPROM Emulation Library Functions.

<1> data Flash library initialization processing (FAL_Init)
   Because it is necessary to initialize the data flash library parameters (RAM) if using the EEPROM emulation library to access the data flash memory, the FAL_Init function must be executed in advance. If flash self programming library processing was executed after this initialization finished, the initialization processing must be re-executed.

<2> EEPROM emulation library initialization processing (EEL_Init)
   Initialize the parameters (RAM) used by the EEPROM emulation library.

<3> EEPROM emulation preparation processing (EEL_Open)
   Set the data flash memory to a status (opened) for which control is enabled to execute EEPROM emulation.

<4> EEPROM emulation execution start processing (EEL_Execute: EEL_CMD_STARTUP command)
   Set the system to a status (started) in which EEPROM emulation can be executed.

<5> EEPROM emulation data write processing (EEL_Execute: EEL_CMD_WRITE command)
   Write the specified data to an EEPROM emulation block.

<6> EEPROM emulation data confirmation processing (EEL_Execute: EEL_CMD_READ command)
   Read data, and then make sure that the data was written correctly by comparing it to the original data.

<7> EEPROM emulation data read processing (EEL_Execute: EEL_CMD_READ command)
   Read written data.

<8> EEPROM emulation execution stop processing (EEL_Execute: EEL_CMD_SHUTDOWN command)
   Set the EEPROM emulation operation to the stopped status (opened).

<9> EEPROM emulation end processing (EEL_Close)
   Set the data flash memory to a status (closed) for which control is disabled to stop EEPROM emulation.

<10> Confirmation before re-executing EEPROM emulation
   If reinitializing the RAM is necessary before re-executing EEPROM emulation, such as when executing flash self programming after EEPROM emulation stops, use the FAL_Init function to re-execute the initialization processing.

1.2.1 EEPROM Emulation Blocks

The EEPROM emulation library Pack01 uses four or more block data flash memory as EEPROM emulation blocks.
1. 2. 2 Data Structure

The data flash memory is written to in word (4 bytes) units. Therefore, when the EEPROM emulation library writes to the data flash memory, the data length is always adjusted to word (4 bytes) units.

In addition, when writing user data, because the reference data for managing data is also written, when calculating the capacity required for writing, the 2 words (8 bytes) of management reference data must be added to the size of the user data in words. Figures 1-9 to 1-11 show an example of the data structure used when user data is written to the data flash memory.

**Figure 1-9. Data Length and Data Structure Example 1 (when User Data is 8 bytes)**

```
User data
1 byte 1 byte 1 byte 1 byte
4 bytes = 1 word
```

Reference data (8 bytes)

```
Data checksum
Index number (widx) RCS Data ID
```

Total size: 4 words (16 bytes) = reference data: 2 words (8 bytes) + [user data (8 bytes)/word size (4 bytes)]

* Change the user data to word units.

**Figure 1-10. Data Length and Data Structure Example 2 (when User Data is 6 bytes)**

```
User data
1 byte 1 byte 1 byte 1 byte
4 bytes = 1 word
```

Reference data (8 bytes)

```
Data checksum
Index number (widx) RCS Data ID
```

Total size: 4 words (16 bytes) = reference data: 2 words (8 bytes) + [user data (6 bytes)/word size (4 bytes)]

* Change the user data to word units. (Round any fractions up to the nearest integer.)
Figure 1-11. Data Length and Data Structure Example 3 (when User Data is 21 bytes)

![Data Structure Diagram]

Total size: 8 words (32 bytes) = reference data: 2 words (8 bytes) + [user data (21 bytes)/word size (4 bytes)]
* Change the user data to word units. (Round any fractions up to the nearest integer.)

(1) DRP
This stands for data reference pointer. This area records the ID of the recorded user data and the reference position.

(2) Index number (widx)
This is the user data index number (reference position).

(3) RCS
This stands for reference checksum. This is the (8-bit) checksum value for the DRP.

(4) Data ID
This is a unique ID for the data being used during EEPROM emulation. User-specified IDs are registered.

(5) DCS
This stands for data checksum. This is the (32-bit) checksum value for the user data and reference data.

Note Before specifying a data ID, it must be registered in the descriptor table.
For details, see 2.4 Initial Values to be Set by User.
1. 2. 3 Block Status Flags

The block status flags start at the beginning of the block and include the P, A, I, and X flags, each of which is 4 bytes, for a total of 16 bytes of data. This data indicates the EEPROM emulation block status, and the combination of flags indicates the block status.

Figure 1-12 shows the placement status of flags, and Table 1-2 shows the combination status of flags.

Figure 1-12. Block Status Flag Placement Positions

<table>
<thead>
<tr>
<th>P Flag</th>
<th>A Flag</th>
<th>I Flag</th>
<th>X Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>55555555H</td>
<td>FFFFFFFFH</td>
<td>FFFFFFFFH</td>
<td>FFFFFFFFH</td>
</tr>
<tr>
<td>55555555H</td>
<td>55555555H</td>
<td>FFFFFFFFH</td>
<td>FFFFFFFFH</td>
</tr>
<tr>
<td>55555555H</td>
<td>55555555H</td>
<td>00000000H</td>
<td>FFFFFFFFH</td>
</tr>
</tbody>
</table>

Data other than the above: FFFFFFFFH

Other than FFFFFFFFH: Use prohibited

Note: A block for which use has been prohibited cannot be reused.
1. 2. 4 Number of Stored User Data Items and Total User Data Size

The following restriction applies to the total size of user data that can be used for EEPROM emulation; the total size of the user data must be such that it is possible to write all the data into within two EEPROM emulation blocks. Therefore, the number of stored data items that can be used differs depending on the size of user data that is actually stored. In addition, because it is not possible to place stored user data such that one data item extends across multiple blocks, if the total size necessary to write the user data exceeds one block, it is also necessary to consider the maximum size of an area for which use might not be possible if one block is exceeded.

The following shows how to calculate the size that can be used when actually writing user data, as well as the total user data size, and Figure 1-13 shows the size concepts when the total user data size is more than one blocks.

- Maximum usable size of one block that can be used to write the user data
  Size of one block of data flash memory: 1,024 bytes
  Size required for EEPROM emulation block management: 32 bytes
  Free space necessary as termination information (separator): 12 bytes

  Maximum usable size of one block = 1,024 bytes – 32 bytes – 12 bytes = 980 bytes

- Maximum size and recommended size
  The maximum size is the total of the usable sizes of EEPROM emulation blocks. The recommended size is less than the maximum size to account for problems such as writing not being possible due to momentary power loss and other issues. It is recommended to only use within a value subtracting half a block capacity from the overall capacity.

  Maximum size = 980 bytes × number of EEPROM emulation blocks × 1/4
  Recommended size = maximum size – 980/2

- Calculating the size for writing each user data item Note
  Size of each written user data item = data size (a size in bytes adjusted to word units) + reference data size (8 bytes)

- Calculating the basic total user data size
  Basic total size = (user data 1 + 8) + (user data 2 + 8) ... + (user data n + 8)

- Calculating the total size when the basic total user data size exceeds the maximum size of one block
  If the basic total user data size exceeds 988 bytes, the maximum usable size of one block, it is necessary to include the maximum size that might become unusable in the calculation when increasing the number of active blocks.

  Total size when more than one blocks are used = basic total size + ((largest user data size + 8) – minimum writing unit for EEPROM emulation (4 bytes)) × (number of necessary blocks – 1)

Note For details, see 1.2.2 Data Structure.
Figure 1-13. Size Concepts when Total User Data Size is Two Blocks (which Exceeds One Block)

- The size of the user data must fit within a quarter of EEPROM emulation blocks when recording all the data, but, if the total user data size does not fit in one block, it is necessary to include the maximum size that might become unusable in the calculation when increasing the number of active blocks. The maximum size that might become unusable is equal to the size of the largest user data that cannot be written.

\[
\text{Maximum size required for writing} = \text{largest user data size} + \text{reference data (8 bytes)} \\
\text{Maximum size that might become unusable} = \text{maximum size required for writing} - \text{size of the minimum writing unit for EEPROM emulation (4 bytes)} \\
\text{Total size when 2 blocks are used} = \text{basic total size} + \text{maximum size that might become unusable} \times (\text{number of necessary blocks - 1})
\]

- The number of active blocks is increased to write new data.

- This area becomes unusable when the new active block is added.
1.3 Initializing EEPROM Emulation Blocks

To use the data flash memory for the EEPROM emulation library, it is necessary to initialize the blocks as EEPROM emulation blocks. If there are blocks that need to be initialized, an EEL_ERR_POOL_INCONSISTENT error occurs when the EEL_CMD_STARTUP command is executed because there are no active EEPROM emulation blocks. To make the EEPROM emulation blocks usable, it is necessary to initialize the blocks by executing the EEL_CMD_FORMAT command.

In addition, if it becomes necessary to change the initial settings, such as because the data flash memory area is corrupted or because user data is added after initialization, or it otherwise becomes impossible to continue using the EEPROM emulation blocks in their current status, or if you just want to perform initialization, initialization can be performed at any time by executing the EEL_CMD_FORMAT command.

Figures 1-14 and 1-15 show the initialization flow and block status transitions used when the EEL_ERR_POOL_INCONSISTENT error occurs upon executing the EEL_CMD_STARTUP command, and Figures 1-16 and 1-17 show the flow and block status transitions used when performing initialization at an arbitrary time.

**Figure 1-14. Initialization Flow when EEL_ERR_POOL_INCONSISTENT Error Occurs**

(when Using Enforced Mode)

```
START
  FAL_Init()
  EEL_Init()
  EEL_Open()
  EEL_Execute()
    EEL_CMD_STARTUP command
    Command execution status
      Error: EEL_ERR_POOL_INCONSISTENT
      Other

    EEL_Execute()
      EEL_CMD_FORMAT command
      Command execution status
        Normal end: EEL_OK
        An error occurs.
        Check the settings and device status.

  EEPROM emulation block initialization processing
```

Perform the necessary processing according to the command execution status.
Figure 1-15. Example Block Status Transitions when Initialization is Performed upon Occurrence of EEL_ERR_POOL_INCONSISTENT Error

Block status before using EEPROM emulation

- Unused area (ALL 0xFF)
- Inactive block

EEPROM emulation blocks (data flash: 4 blocks, 4 KB)

Block status after executing the FORMAT command

- Unused area (ALL 0xFF)
- Reserved area
- Management data
- Active block

- Unused area (ALL 0xFF)
- Reserved area
- Management data
- Prepared block

The FORMAT command is executed.
Figure 1-16. Flow when Performing Initialization at Arbitrary Time (when Using Enforced Mode)

START
- FAL_Init()
- EEL_Init()
- EEL_Open()

EEPROM emulation block initialization processing
- EEL_Execute()
- EEL_CMD_FORMAT command

Command execution status
- EEL_OK: Normal end
- An error occurs.

To the next process
- Check the settings and device status.
Figure 1-17. Example Block Status Transitions when EEPROM Emulation Blocks in Use are Initialized at Arbitrary Time

The FORMAT command is executed.

Block status after executing the FORMAT command
1.4 Adjusting EEPROM Emulation Blocks

When writing to the EEPROM emulation blocks, a prepared block is erased each time an active block is added, but, if it is necessary to add an active block when there are two or fewer prepared blocks remaining, the active block is added, the valid data remaining in the old block is moved, and the old block is erased before writing the specified data. If writing is performed when erasing a block is required, the time necessary to move the data and erase the block is added to the time necessary for writing.

If this additional processing time is not permissible, it is possible to perform maintenance at a time that will not adversely affect the system in order to avoid moving and erasing data at the same time when high-priority data must be written.

To perform maintenance, either adjust blocks by executing the EEL_CMD_CLEANUP command or execute the maintenance mode processing of the EEL_Handler function.

Figure 1-18 shows an example of the block status when moving and erasing data upon adding an active block, and Figure 1-19 shows how the processing time is changed according to the block status when writing data.

**Figure 1-18. Example Block Status when Moving and Erasing Data upon Adding Active Block**

![](image_url)

If there are few prepared blocks remaining when an active block is added, the old block is erased.

Data is written to add the active block.

Data is moved.

After all the processing finishes, the specified data is written.
**Figure 1-19. Changes in Processing Time according to Block Status when Writing Data (when Using Enforced Mode)**

<table>
<thead>
<tr>
<th>Command execution status</th>
<th>Change in the writing operation and time according to the block status</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE command processing</td>
<td>When only performing writing</td>
</tr>
<tr>
<td></td>
<td>When performing writing while adding an active block</td>
</tr>
<tr>
<td></td>
<td>When performing writing while adding a block and erasing data</td>
</tr>
</tbody>
</table>

- **EEL_Execute()**
- **WRITE command processing**
- **Processing to write the specified data**
- **Active block creation**
- **Old block erasure processing**
- **Prepared block creation**
- **Processing to write the specified data**

When an active block is added, how long it takes for writing to finish differs depending on the block status.

Function execution timing

Processing completion timing

Difference compared to when only writing is performed
### 1.4.1 Adjusting Blocks by Using the EEL_CMD_CLEANUP Command

Blocks can be adjusted by executing the EEL_CMD_CLEANUP command from the EEL_Execute function.

Because this feature moves all the data to the latest newly created active block even if there is capacity remaining in the latest active block, the possible number of rewrites decreases by worth of data of blocks necessary to store data, but the feature makes it possible to change to a status in which there is only valid data (a status in which invalid data does not exist).

Figure 1-20 shows the flow when executing the EEL_CMD_CLEANUP command, and Figure 1-21 shows the block status after executing the EEL_CMD_CLEANUP command.

**Figure 1-20. Operation Flow of the EEL_CMD_CLEANUP Command (when Using Enforced Mode)**

```
START
  FAL_Init()
  EEL_Init()
  EEL_Open()
  EEL_Execute()
    EEL_CMD_CLEANUP command
      Command execution status
        Normal end: EEL_OK
        Other
          Perform the necessary processing according to the command execution status.

  EEL_Execute()
    EEL_CMD_CLEANUP command
      Command execution status
        Normal end: EEL_OK
        Other
          Perform the necessary processing according to the command execution status.

  To the next process
```
A new active block is added, and the valid data in all the old active blocks is moved to the latest active block. Blocks for which moving the data finishes are erased and changed to prepared blocks.

The latest data is moved.
1.4.2 Adjusting Blocks by Using the EEL_Handler Function (Maintenance Mode)

Block adjustment processing can be automatically performed by executing the EEL_Handler function while no commands are executing. This is called the maintenance mode. When the EEL_Handler function is executed while there are no commands to execute, the block status is checked, block adjustment is judged to be necessary if the number of active blocks exceeds the specified initial value EEL_REFRESH_BLOCK_THRESHOLD (described in 2.4 Initial Values to Be Set by User), all necessary data in old active blocks is moved to the latest block, and the unnecessary blocks are erased and changed to prepared blocks.

The processing can be interrupted in the maintenance mode. If a new command is executed by the EEL_Execute function, the maintenance mode is paused, and executing the newly specified command is prioritized. To judge whether maintenance mode processing is being executed, it is necessary to execute the separate EEL_GetDriverStatus function and check the execution status of the EEL_Handler function. If nothing is being performed even when the EEL_Handler function is executed four times, adjustment is finished (the adjustment completion status).

Figure 1-22 and Table 1-3 show how to check the execution status of the EEL_Handler function, Figures 1-23 to 1-26 show an execution flow example, and Figure 1-27 shows the block transitions when the maintenance mode processing finishes (the adjustment completion status).

**Figure 1-22. Flow for Checking Execution Status of the EEL_Handler Function**

- EEL_Handler()
- EEL_GetDriverStatus()
- Command execution status: EEL_OK
  - eel_operation_status_t: EEL_OPERATION_IDLE
    - Command execution and maintenance mode processing have finished.
- Command execution status: EEL_BUSY
  - eel_operation_status_t: EEL_OPERATION_BUSY
    - A command is being executed.
- Command execution status: EEL_OK
  - eel_operation_status_t: EEL_OPERATION_BUSY
    - Maintenance is being performed.

**Table 1-3. EEL_Handler Function Execution Status**

<table>
<thead>
<tr>
<th>EEL_Handler() Command Execution Status</th>
<th>EEL_GetDriverStatus() eel_operation_status_t Status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_OK</td>
<td>EEL_OPERATION_IDLE</td>
<td>This is the command execution and maintenance mode processing completion status. (This includes when the maintenance mode processing is paused.) If all the results are this status when the EEL_Handler function is executed four times, adjustment is complete.</td>
</tr>
<tr>
<td>EEL_OK</td>
<td>EEL_OPERATION_BUSY</td>
<td>Maintenance mode processing is being executed.</td>
</tr>
<tr>
<td>EEL_BUSY</td>
<td>EEL_OPERATION_BUSY</td>
<td>A command is being executed.</td>
</tr>
</tbody>
</table>
After the STARTUP command finishes

- EEL_Handler()
- EEL_GetDriverStatus()

If the following statuses are continued four times
- Command execution status:
  - EEL_OK
  - eel_operation_status_t: EEL_OPERATION_IDLE

Adjustment completion status
- To the next process
- Perform the necessary processing according to the command execution status.

The command execution status indicates an error.
Figure 1-25. Execution Flow when Performing Adjustment upon Starting EEPROM Emulation (Adjustment Completion Status)

START

FAL_Init()

EEL_Init()

EEL_Open()

EEL_Execute() EEL_CMD_STARTUP command

Command execution status

Normal end: EEL_OK

Other

Perform the necessary processing according to the command execution status.

Block adjustment processing

EEL_Handler()

EEL_GetDriverStatus()

Other

EEL_Handler() execution status

if the following statuses are continued four times

Command execution status:

eel_operation_status_T1: EEL_OK
EEL_OPERATION_IDLE

Other

The command execution status indicates an error.

Adjustment completion status

To the next process

Perform the necessary processing according to the command execution status.
START

FAL_Init()

EEL_Init()

EEL_Open()

EEL_Execute()

EEL_Execute()

EEL_CMD_STARTUP

command

EEL_Execute()

EEL_CMD_WRITE

command

EEL_Execute()

EEL_CMD_SHUTDOWN

command

EEL_Close()

END

Polling processing (user processing loop)

Status and event judgment (user processing)

Judgment processing (switch)

Write

Read

Polling processing

Event processing (user processing)

Various events

Event parameter specification

No processing

Processing status judgment (user processing)

Can continue as executable when there is no need to end

The command is executed, as well as maintenance processing if there is time.

To check the status every time, also call EEL_GetDriverStatus().

Block adjustment processing
Figure 1-27. Example of Block Adjustment in EEL_Handler Function Maintenance Mode (Adjustment Completion Status)

The latest data is moved.
CHAPTER 2 USING EEPROM EMULATION

EEPROM emulation can store a maximum of 255 data items each consisting of 1 to 255 bytes in the flash memory by using eight blocks of flash memory. EEPROM emulation can be executed by incorporating the EEPROM emulation library into a user-created program and executing that program.

Note For details about the number of user data items that can be stored, see 1.2.4 Number of Stored User Data Items and Total User Data Size.

2.1 Caution Points

EEPROM emulation is achieved by using a feature for manipulating the on-board microcontroller data flash memory. Therefore, it is necessary to note the following:

1. Before using the EEPROM emulation library, always close the flash self programming library. Also, do not run the flash self programming library while the EEPROM emulation library is being used. When using the flash self programming library, be sure to execute all of the processing up to and including the EEL_Close function to finish EEPROM emulation. When using EEPROM emulation after executing flash self programming library processing, it is necessary to start processing from the initializing function (the FAL_Init function).

2. Do not execute STOP mode or HALT mode processing while the EEPROM emulation is being used. If it is necessary to execute STOP mode or HALT mode processing, be sure to execute all of the processing up to and including the EEL_Close function to finish EEPROM emulation.

3. The watchdog timer does not stop during the execution of the EEPROM emulation Library Pack 01.

4. The data flash memory cannot be read during data flash memory operation by the EEPROM emulation Library Pack 01.

5. In address above 0xFFE20 (0xFE20), do not place data buffer (argument) or stack which is used by EEPROM emulation library functions and data flash library functions.

6. When using data transfer controller (DTC) during EEPROM emulation, do not place RAM area used by DTC in self RAM and in address above 0xFFE20 (0xFE20).

7. Until EEPROM emulation is finished, do not corrupt RAM area (including self RAM) used by EEPROM emulation.
(8) Do not execute any of the EEPROM emulation library functions during interrupt processing except for the EEL_TimeOut_CountDown function. Because the EEPROM emulation library functions do not support execution more than once at the same time, the operation is not guaranteed when executing these functions during interrupt processing.

(9) When executing EEPROM emulation library processing in an OS, except for the EEL_TimeOut_CountDown function, do not execute any of the EEPROM emulation library functions from multiple tasks. Because the EEPROM emulation library functions do not support execution more than once at the same time, the operation is not guaranteed when executing these functions from multiple tasks.

(10) Before starting the EEPROM emulation, be sure to start up the high-speed on-chip oscillator first.

(11) About an operating frequency of RL78 microcontrollers and an operating frequency value set by the initializing function (FAL_Init), be aware of the following points:
- When using a frequency lower than 4 MHz as an operating frequency of RL78 microcontrollers, only 1 MHz, 2 MHz and 3 MHz can be used (frequencies other than integer values like a 1.5 MHz cannot be used). Also, set an integer value 1, 2, or 3 to the operating frequency value set by the initializing function.
- When using a frequency of 4 MHz or higher \(^{Note1}\) as an operating frequency of RL78 microcontrollers, a certain frequency can be used as an operating frequency of RL78 microcontrollers.
- This operating frequency is not the frequency of the high-speed on-chip oscillator.

**Note 1.** For a maximum frequency, see the target RL78 microcontroller user’s manual.

(12) Do not operate the DFTCTL during the execution of the EEPROM emulation library.

(13) Initialize the argument (RAM) that is used by the EEPROM emulation library function. When not initialized, a RAM parity error is detected and the RL78 microcontroller might be reset. For a RAM parity error, refer to the user’s manual of the target RL78 microcontroller.

(14) Initialize the SADDR area that is used by the EEPROM emulation library function. When not initialized, a RAM parity error is detected and the RL78 microcontroller might be reset. For a RAM parity error, refer to the user’s manual of the target RL78 microcontroller.

(15) To use the data flash memory for EEPROM emulation, it is necessary to execute the EEL_CMD_FORMAT command upon first starting up to initialize the data flash memory and make it usable as EEPROM emulation blocks.
(16) After initializing the EEPROM emulation blocks, do not change the initial values specified as fixed values for stored data and the blocks. If these values are changed, the specified parameters become inconsistent with the data written to the blocks, and there is a risk of no longer being able to correctly execute EEPROM emulation. If the values must be changed, execute the EEL_CMD_FORMAT command to reinitialize the EEPROM emulation blocks.

(17) Four or more blocks of data flash memory are necessary to use the EEPROM Emulation Library Pack 01. If the number of data flash memory blocks is less than four, this library is unusable.
2.2 Total Processing Time

The total processing time is the time until successful termination. This does not include the time until abnormal termination due to errors in the input data or other errors.

Figure 2-1 Overview of Total Processing Time
### Table 2-1 Total Processing Time of EEPROM Emulation Library Pack 01 (when All Data Fits in 1 Block)

<table>
<thead>
<tr>
<th>Functions</th>
<th>MAX time (Full Speed Mode)</th>
<th>MAX time (Wide Voltage Mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>. FAL_Init</td>
<td>2580 / fclk + 443 μs</td>
<td>2536 / fclk + 968 μs</td>
</tr>
<tr>
<td>. EEL_Init</td>
<td>(2123 + 80 × Data Num) / fclk μs</td>
<td>(2123 + 80 × Data Num) / fclk μs</td>
</tr>
<tr>
<td>. EEL_Open</td>
<td>87 / fclk + 12 μs</td>
<td>87 / fclk + 12 μs</td>
</tr>
<tr>
<td>. EEL_Close</td>
<td>866/ fclk + 443 μs</td>
<td>822/ fclk + 968 μs</td>
</tr>
<tr>
<td>. EEL_GetDriverStatus</td>
<td>47 / fclk μs</td>
<td>47 / fclk μs</td>
</tr>
<tr>
<td>. EEL_GetSpace</td>
<td>57 / fclk μs</td>
<td>57 / fclk μs</td>
</tr>
<tr>
<td>. EEL_GetVersionString</td>
<td>10 / fclk μs</td>
<td>10 / fclk μs</td>
</tr>
<tr>
<td>. EEL_TimeOut_CountDown</td>
<td>30 / fclk μs</td>
<td>30 / fclk μs</td>
</tr>
<tr>
<td>. EEL_Execute/EEL_Handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>. EEL_CMD_FORMAT</td>
<td>(352494 / fclk + 330988) × Block Num μs</td>
<td>(311793 / fclk + 374134) × Block Num μs</td>
</tr>
<tr>
<td>. EEL_CMD_START</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Minimum data length is 1 to 4 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Num 8</td>
<td>(1082020 + 491768 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(1054886 + 491768 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 64</td>
<td>(1926490 + 4482900 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(1899356 + 4482900 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 128</td>
<td>(2891690 + 9044100 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(2864556 + 9044100 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 255</td>
<td>(4806090 + 18097500 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(4777956 + 18097500 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>2. Minimum data length is 13 to 16 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Num 8</td>
<td>(771606 + 256268 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(744472 + 256268 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 64</td>
<td>(1189076 + 2363400 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(1161942 + 2363400 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 128</td>
<td>(1666676 + 4771500 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(1639542 + 4771500 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 255</td>
<td>(2613676 + 9550500 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(2586542 + 9550500 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>3. Minimum data length is 61 to 64 bytes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Num 8</td>
<td>(569540 + 101109 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(542406 + 101109 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 64</td>
<td>(705816 + 967779 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(678682 + 967779 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
<tr>
<td>Data Num 128</td>
<td>(861528 + 1958100 × (Block Num - 2)) / fclk + 330988 μs</td>
<td>(834394 + 1958100 × (Block Num - 2)) / fclk + 374134 μs</td>
</tr>
</tbody>
</table>

Remarks. fclk: CPU/peripheral hardware clock frequency (for example, at 20 MHz, fclk = 20)

Data Num: Number of data entries registered
Block Num: Number of EEPROM emulation blocks

Note: If you do not intend to use any of the minimum data lengths listed above, check the processing time for a smaller minimum data length than the nearest longer range.
<table>
<thead>
<tr>
<th>Functions</th>
<th>MAX (Full Speed Mode)</th>
<th>MAX (Wide Voltage Mode)</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_Excute/EEL_Handler</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• EEL_CMD_CLEANUP</td>
<td>(352494 / fclk + 330988) × Block Num + 4236841 / fclk + 223893 μs</td>
<td>(311793 / fclk + 374134) × Block Num + 4219942 / fclk + 851467 μs</td>
</tr>
<tr>
<td>• EEL_CMD_WRITE (Max data size)</td>
<td>5763174 / fclk + 1024588 μs</td>
<td>5650473 / fclk + 1906134 μs</td>
</tr>
<tr>
<td>• EEL_CMD_READ</td>
<td>58563 / fclk μs</td>
<td>58563 / fclk μs</td>
</tr>
<tr>
<td>• EEL_CMD_SHUTDOWN</td>
<td>1674 / fclk μs</td>
<td>1674 / fclk μs</td>
</tr>
</tbody>
</table>

Remarks. fclk: CPU/peripheral hardware clock frequency (for example, at 20 MHz, fclk = 20)

Block Num: Number of EEPROM emulation blocks
2. 3 Software Resources

In the EEPROM emulation library, program areas corresponding to parts of the library to be used, RAM areas for variables to be used in the library, and RAM areas for work area (self RAM) are used to assign an appropriate program to the user area. Also, since the data flash library will be used, the EEPROM emulation library must have a separate area for use by the data flash library.

Tables 2-2 and 2-3 list the software resources\(^{\text{Notes 1, 2}}\). Figures 2-2 and 2-3 show examples of arrangements of RAM.

Table 2-2 Software Resources Used by EEPROM Emulation Library Pack01 Ver. 1.14

<table>
<thead>
<tr>
<th>Item</th>
<th>Size(Byte)</th>
<th>Restrictions on Allocation and Usage(^{\text{Note 1}})</th>
</tr>
</thead>
<tbody>
<tr>
<td>Self RAM(^{\text{Note 2}})</td>
<td>0 to 1024(^{\text{Note 2}})</td>
<td>Use of the self-RAM area by RL78 Family EEPROM Emulation Library Pack 01 differs with the device. For details, refer to &quot;RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944)&quot;.</td>
</tr>
<tr>
<td>Stack</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>Data buffer(^{\text{Note 3}})</td>
<td>1 to 256</td>
<td>Can be allocated to a RAM area other than the self RAM and the area from FFE20H to FFEFFH.</td>
</tr>
<tr>
<td>Requester (argument)</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>Library search area</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>SADDR RAM work area</td>
<td>11</td>
<td>Can be allocated to a short-addressing RAM area.</td>
</tr>
<tr>
<td>(fdl:2)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(eel:9)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Library size</td>
<td>8200</td>
<td>Can be allocated to a program area other than the self RAM, the area from FFE20H to FFEFFH, and the internal ROM.</td>
</tr>
<tr>
<td>(fdl:1500)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(eel:6700)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data table</td>
<td>((n + 1) \times 4)</td>
<td>Can be allocated to a program area other than the self RAM, the area from FFE20H to FFEFFH, and the internal ROM.</td>
</tr>
<tr>
<td>(n = \text{number of data items stored})</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fixed-parameter area (default)</td>
<td>72</td>
<td></td>
</tr>
<tr>
<td>(fdl:64)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(eel:4)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEPROM emulation block</td>
<td>4,096 or more (at least 4 blocks)</td>
<td>Only data flash memory can be used.</td>
</tr>
</tbody>
</table>

Notes: 1. For devices not shown in the RL78 Family Self RAM list of Flash Self Programming Library (R20UT2944), contact your Renesas sales agency.
2. An area used as the working area by the EEPROM Emulation library Pack 01 is called self-RAM in this manual and the user's manual. The self-RAM requires no user setting because it is an area that is not mapped and automatically used at execution of the EEPROM Emulation library (previous data is discarded). When the EEPROM Emulation library is not used, the self-RAM can be used as a normal RAM space.
3. The data buffer is used as the working area for EEPROM Emulation Library Pack01 internal processing or the area where the data to be set is allocated in the EEL_Execute function. The required size depends on the function to be used.
Table 2-3. Data Buffer Size Used by EEPROM Emulation Library Pack01 Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAL_Init</td>
<td>0</td>
</tr>
<tr>
<td>EEL_Init</td>
<td>0</td>
</tr>
<tr>
<td>EEL_Open</td>
<td>0</td>
</tr>
<tr>
<td>EEL_Close</td>
<td>0</td>
</tr>
<tr>
<td>EEL_Execute(\text{note})</td>
<td>0 to 256</td>
</tr>
<tr>
<td>EEL_Handler(\text{note})</td>
<td>0 to 256</td>
</tr>
<tr>
<td>EEL_TimeOut_CountDown(\text{note})</td>
<td>0</td>
</tr>
<tr>
<td>EEL_GetDriverStatus</td>
<td>3</td>
</tr>
<tr>
<td>EEL_GetSpace</td>
<td>2</td>
</tr>
<tr>
<td>EEL_GetVersionString</td>
<td>0</td>
</tr>
</tbody>
</table>

Note. Another 6 bytes area of the requester is used.
Figure 2-2 Example 1 of RAM Allocation with Self-RAM
(RL78/G13: product with 4 KB RAM and 64 KB ROM)

Figure 2-3 Example 2 of RAM Allocation without Self-RAM
(RL78/G13: product with 2 KB RAM and 32 KB ROM)
2. 4 Initial Values to be Set by User

As the initial values for the EEPROM emulation library, be sure to set the items indicated below. In addition, before executing the EEPROM emulation library, be sure to execute the high-speed on-chip oscillator.

- Number of stored data items, and specific data IDs and data size

```c
<Data flash library user include file (fdl_descriptor.h)>\textsuperscript{Notes 1, 2}
#define FDL_SYSTEM_FREQUENCY    2000000: (1) Operating frequency
#define FDL_WIDE_VOLTAGE_MODE: (2) Voltage mode
#define FAL_POOL_SIZE 4: (3) FAL pool size
#define EEL_POOL_SIZE 0: (4) EEL pool size

<EEPROM emulation library user include file (eel_descriptor.h)>\textsuperscript{Notes 1, 2}
#define EEL_STORAGE_TYPE 'D': Flash memory type (D: Data flash memory)
#define EEL_VAR_NO 5: Number of stored data items
#define EEL_REFRESH_BLOCK_THRESHOLD 1: Threshold setting

<EEPROM emulation library user program file (eel_descriptor.c)>\textsuperscript{Notes 1, 2}
__far const eel_u08 eel_descriptor[EEL_VAR_NO+1][4]:
    = (eel_u08)EEL_REFRESH_BLOCK_THRESHOLD;
__far const eel_u08 eel_refresh_bth_u08:
    = (eel_u08)EEL_STORAGE_TYPE;
__far const eel_u08 eel_storage_type_u08:
    = (eel_u08)EEL_VAR_NO;
__far const eel_u08 eel_var_number_u08:
    = (eel_u08)EEL_VAR_NO;
```

\textbf{Notes} 1. The macros and macro names that are being used have common parameters with the EEPROM emulation library, so changes should be made to numerical values only.
2. After initializing the EEPROM emulation blocks (after executing the EEL_CMD_FORMAT command), do not change the values. If the values are changed, reinitialize the EEPROM emulation blocks.
(1) Operating frequency

Set an operating frequency which is used in RL78 microcontrollers.  

The setting value is set to the FAL_Init frequency parameter by the following expressions (The frequency is calculated by raising its decimals. The result calculated omits its decimals.).

Setting value of FAL_Init operating frequency = ((FDL_SYSTEM_FREQUENCY + 999999) / 1000000)

Ex.1:  When FDL_SYSTEM_FREQUENCY is 20000000 (20 MHz),

\[(20000000 + 999999) / 10000000 \approx 20.999999 = 20\]

Ex.2:  When FDL_SYSTEM_FREQUENCY is 4500000 (4.5 MHz),

\[(4500000 + 999999) / 10000000 \approx 5.499999 = 5\]

Ex.3:  When FDL_SYSTEM_FREQUENCY is 5000001 (5.000001 MHz),

\[(5000001 + 999999) / 10000000 \approx 6.000000 = 6\]

Note  This setting is a value required to control data flash memory. This setting does not change the operating frequency of RL78 microcontrollers. In addition, this operating frequency is not the frequency of the high-speed on-chip oscillator.

(2) Voltage mode

Set the voltage mode of data flash memory.  

When FDL_WIDE_VOLTAGE_MODE is not defined: Full-speed mode
When FDL_WIDE_VOLTAGE_MODE is defined: Wide voltage mode

Notes 1. The FDL_WIDE_VOLTAGE_MODE is commented out and not defined in the initial setting. To use RL78 microcontrollers in the wide voltage mode, cancel the comment-out to define the mode.
2. For details of the voltage mode, see the corresponding RL78 microcontrollers user’s manual.

(3) FAL pool size

Set the number of blocks of data flash memory mounted on your RL78 microcontrollers.

(4) EEL pool size

Set the number of blocks of data flash memory mounted on your RL78 microcontrollers.

Note  Set the value of 4 (four blocks) or more.

(5) Data identifier (data ID) and size

This table specifies the data identifiers (data IDs) and sizes. It is called the EEL descriptor table. With RL78 EEPROM emulation library Pack01, it is not possible to add identifiers while the program is running. Accordingly, the data to be written must be registered to the EEL descriptor table in advance.

Figure 2-4.  EEL Descriptor Table (when There are Four Instances of Different Data)

__far const eel_u08  eel_descriptor [Number of stored data items + 1][4]

<table>
<thead>
<tr>
<th>Data ID (A)</th>
<th>Word size</th>
<th>Byte size</th>
<th>0x01</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data ID (B)</td>
<td>Word size</td>
<td>Byte size</td>
<td>0x01</td>
</tr>
<tr>
<td>Data ID (C)</td>
<td>Word size</td>
<td>Byte size</td>
<td>0x01</td>
</tr>
<tr>
<td>Data ID (D)</td>
<td>Word size</td>
<td>Byte size</td>
<td>0x01</td>
</tr>
<tr>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
<td>0x00</td>
</tr>
</tbody>
</table>
Data ID
The data ID is specified by the user.

Word size
This is the word size of the data to be written.

Byte size
This is the byte size of the data to be written.

RAM reference flag (0x01)
This flag is used for reference settings. Specify 1 as the registration data.

Termination area (0x00)
Specify 0 as the termination information.

(6) Threshold setting
This is the number of active blocks used as the adjustment reference when executing maintenance mode processing. The maintenance mode processing is executed when the number of active blocks exceeds this setting. For this setting, specify the number of blocks necessary for valid data storage + 1. When the capacity necessary for data storage is less than a half of maximum usable size of one block, “+ 1” is not required.

Note For details about the number of blocks necessary for valid data storage, see 1.2.4 Number of Stored User Data Items and Total User Data Size.

(7) Flash memory type
This does not have to be changed. Use the initial value as is.

(8) Number of stored data items
Set the number of data items used for EEPROM emulation. The setting range is from 1 to 255.
CHAPTER 3  EEPROM EMULATION FEATURES

This chapter describes the functions used for EEPROM emulation.

3. 1 Data Flash Library Functions

When executing EEPROM emulation by using the data flash memory, operations such as erasing or writing to the data flash memory are performed by manipulating the data flash library from the EEPROM emulation library. Therefore, before executing EEPROM emulation, it is necessary to initialize the data flash library, specify the settings for accessing the data flash memory, and make other preparations. Table 3-1 shows the data flash library function necessary for executing EEPROM emulation library processing.

Table 3-1. Data Flash Library Function that must be Executed before Using EEPROM Emulation

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAL_Init</td>
<td>This initializes the data flash library.</td>
</tr>
</tbody>
</table>
FAL_Init

[Outline]
Data flash library initialization processing

[Format]

<C language>

fal_status_t __far FAL_Init(const __far fal_descriptor_t* descriptor_pstr)

<Assembler>

CALL !_FAL_Init or CALL !!_FAL_Init

Remark  Call this function with "!" if you are placing the data flash library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
• The flash self programming library and EEPROM emulation library processing must be either not executing or finished.
• Before executing this function, be sure to execute the high-speed on-chip oscillator.

[Function]
This function initializes parameters for accessing the data flash memory.

Cautions  1. Be sure to execute this function when starting EEPROM emulation to make it possible to start accessing the data flash memory.
   2. This function is mutually exclusive with the flash self programming library (FSL). Before executing this function, be sure to close the flash self programming library. Also, never use any flash self programming library functions during EEPROM emulation.
   3. To use the flash self programming library after this function is executed, the RAM must be reinitialized, so always execute this function when restarting the EEPROM emulation library.
   4. To execute this function again, always close the EEPROM emulation library.
   5. The table used for this function cannot be modified. Be sure to use a defined table.

[Register status after calling this function]
Return value: C
Corrupted registers: AX (argument) and BC (argument)
### Argument fal_descriptor_t Details (Defined Fixed Values Cannot be Changed by Users)

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fal_pool_first_addr_u32</td>
<td>fal_u32</td>
<td>Start address of data flash memory</td>
</tr>
<tr>
<td>eel_pool_first_addr_u32</td>
<td>fal_u32</td>
<td>Start address of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_pool_first_addr_u32</td>
<td>fal_u32</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_pool_last_addr_u32</td>
<td>fal_u32</td>
<td>End address of data flash memory</td>
</tr>
<tr>
<td>eel_pool_last_addr_u32</td>
<td>fal_u32</td>
<td>End address of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_pool_last_addr_u32</td>
<td>fal_u32</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_pool_first_block_u16</td>
<td>fal_u16</td>
<td>First block number of data flash memory</td>
</tr>
<tr>
<td>eel_pool_first_block_u16</td>
<td>fal_u16</td>
<td>First block number of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_pool_first_block_u16</td>
<td>fal_u16</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_pool_last_block_u16</td>
<td>fal_u16</td>
<td>Last block number of data flash memory</td>
</tr>
<tr>
<td>eel_pool_last_block_u16</td>
<td>fal_u16</td>
<td>Last block number of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_pool_last_block_u16</td>
<td>fal_u16</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_first_widx_u16</td>
<td>fal_u16</td>
<td>Access start number of data flash memory</td>
</tr>
<tr>
<td>eel_first_widx_u16</td>
<td>fal_u16</td>
<td>Access start number of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_first_widx_u16</td>
<td>fal_u16</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_last_widx_u16</td>
<td>fal_u16</td>
<td>Access end number of data flash memory</td>
</tr>
<tr>
<td>eel_last_widx_u16</td>
<td>fal_u16</td>
<td>Access end number of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_last_widx_u16</td>
<td>fal_u16</td>
<td>Unused</td>
</tr>
<tr>
<td>fal_pool_wsize_u16</td>
<td>fal_u16</td>
<td>Area size of all blocks (word units: 4 bytes units)</td>
</tr>
<tr>
<td>eel_pool_wsize_u16</td>
<td>fal_u16</td>
<td>Area size of all blocks used for EEL (word units: 4 bytes units)</td>
</tr>
<tr>
<td>user_pool_wsize_u16</td>
<td>fal_u16</td>
<td>Unused</td>
</tr>
<tr>
<td>block_size_u16</td>
<td>fal_u16</td>
<td>Area size of one block (byte units)</td>
</tr>
<tr>
<td>block_wsize_u16</td>
<td>fal_u16</td>
<td>Area size of one block (word units: 4 bytes units)</td>
</tr>
<tr>
<td>fal_pool_size_u08</td>
<td>fal_u08</td>
<td>Block size of data flash memory</td>
</tr>
<tr>
<td>eel_pool_size_u08</td>
<td>fal_u08</td>
<td>Block size of data flash memory used for EEL</td>
</tr>
<tr>
<td>user_pool_size_u08</td>
<td>fal_u08</td>
<td>Unused</td>
</tr>
<tr>
<td>fx_MHz_u08</td>
<td>fal_u08</td>
<td>Setting of operating frequency of RL78 microcontroller</td>
</tr>
<tr>
<td>wide_voltage_mode_u08</td>
<td>fal_u08</td>
<td>Setting of voltage mode</td>
</tr>
</tbody>
</table>

#### Notes

1. Users must not modify this defined table.
2. EEL stands for EEPROM emulation library.

### Development Tool

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Argument Type/Register</th>
<th>Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENESAS small and medium model</td>
<td>const__far fal_descriptor_t* descriptor_pstr</td>
<td>AX (0 to 15), BC (16 to 23)</td>
</tr>
<tr>
<td>RENESAS large model</td>
<td>const__far fal_descriptor_t* descriptor_pstr</td>
<td>AX (0 to 15), BC (16 to 23)</td>
</tr>
</tbody>
</table>
**Return value**

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fal_status_t</td>
<td>FAL_OK</td>
<td>Normal end</td>
</tr>
<tr>
<td></td>
<td>FAL_ERR_CONFIGURATION</td>
<td>Initialization error. The setting is incorrect. Or high-speed on-chip oscillator does not run. Make sure that the defined data has not been changed or the high-speed on-chip oscillator is running.</td>
</tr>
</tbody>
</table>

**Remark** Assembly language return values are stored in register C.
3.2 EEPROM Emulation Library Functions

The EEPROM emulation library consists of the following library functions.

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_Init</td>
<td>Processing to initialize the RAM used for EEPROM emulation</td>
</tr>
<tr>
<td>EEL_Open</td>
<td>EEPROM emulation preparation processing</td>
</tr>
<tr>
<td>EEL_Close</td>
<td>EEPROM emulation end processing</td>
</tr>
<tr>
<td>EEL_Execute</td>
<td>EEPROM emulation command execution processing</td>
</tr>
<tr>
<td>EEL_Handler</td>
<td>Processing to continue executing EEPROM emulation commands or block adjustment processing</td>
</tr>
<tr>
<td></td>
<td>* This is used for command execution in modes other than the enforced mode.</td>
</tr>
<tr>
<td>EEL_TimeOut_CountDown</td>
<td>Timeout counting processing for EEPROM emulation command execution</td>
</tr>
<tr>
<td></td>
<td>* This is used only in the timeout mode.</td>
</tr>
<tr>
<td>EEL_GetDriverStatus</td>
<td>This obtains the EEPROM emulation library status.</td>
</tr>
<tr>
<td>EEL_GetSpace</td>
<td>This obtains the status indicating the number of free EEPROM emulation blocks.</td>
</tr>
<tr>
<td>EEL_GetVersionString</td>
<td>This obtains the version information of the EEPROM emulation library (EEL).</td>
</tr>
</tbody>
</table>
EEL_Init

[Outline]
Processing to initialize the RAM used for EEPROM emulation

[Format]

<C language>
```
eel_status_t __far EEL_Init (void)
```

<Assembler>
```
CALL !_EEL_Init or CALL !!_EEL_Init
```

Remark Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
1. The flash self programming library and EEPROM emulation library processing must be either not executing or finished.
2. The FAL_Init function must have finished normally.

[Function]
This function initializes parameters used to execute EEPROM emulation.

Cautions
1. Be sure to execute this function when starting EEPROM emulation to initialize the RAM to be used.
2. This function is mutually exclusive with the flash self programming library (FSL). Before executing this function, be sure to close the flash self programming library. Also, never use any flash self programming library functions during EEPROM emulation.
3. To use the flash self programming library after this function is executed, the RAM must be reinitialized, so always execute this function when restarting the EEPROM emulation library.
4. To execute this function again, always close the EEPROM emulation library.

[Register status after calling this function]
Return value: C

[Argument]
None

[Return value]

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eel_status_t</td>
<td>EEL_OK</td>
<td>Normal end</td>
</tr>
<tr>
<td></td>
<td>EEL_ERR_CONFIGURATION</td>
<td>Initialization error. Either the FAL_Init function is not being executed or the value specified for FAL_Init and EEL_Init functions cannot be used to execute EEL processing.</td>
</tr>
</tbody>
</table>

Remark Assembly language return values are stored in register C.
EEPROM Emulation Library Pack 01

CHAPTER 3  EEPROM EMULATION FEATURES

EEL_Open

[Outline]
EEPROM emulation preparation processing

[Format]

<C language>

```c
void __far EEL_Open(void)
```

<Assembler>

```assembly
CALL !_EEL_Open or CALL !!_EEL_Open
```

Remark  Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
1. The FAL_Init and EEL_Init functions must have finished normally.
2. If EEPROM emulation was executed, the processing up to EEL_Close must be executed to stop the processing (closed status).

[Function]
This function changes the system to a status in which the data flash memory can be manipulated to make it possible to execute EEPROM emulation.

Note  After executing the EEL_Open function and switching to the EEPROM emulation start status (opened), flash self programming library processing cannot be executed. It also becomes impossible to execute STOP mode and HALT mode processing. If it is necessary to execute flash self programming library, STOP mode, or HALT mode processing, execute the EEL_Close function to switch EEPROM emulation to the stopped status (closed).

[Register status after calling this function]
No registers are corrupted.

[Argument]
None

[Return value]
None
EEL_Close

[Outline]
EEPROM emulation end processing

[Format]

<C language>
void __far EEL_Close(void)

<Assembler>
CALL !_EEL_Close or CALL !!_EEL_Close

Remark Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
If EEPROM emulation was executed, the EEL_CMD_SHUTDOWN command must be used to set EEPROM emulation to the stopped status (the opened status).

[Function]
This function changes the system to a status in which the data flash memory can be manipulated to make it impossible to execute EEPROM emulation.

[Register status after calling this function]
No registers are corrupted.

[Argument]
None

[Return value]
None
[Outline]
EEPROM emulation execution function

[Format]

<C language>

```c
void __far EEL_Execute( eel_request_t* request )
```

<Assembler>

```assembly
CALL !_EEL_Execute or CALL !!_EEL_Execute
```

Remark Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]
Each type of processing for performing EEPROM emulation operations is specified for this function as an argument in the command format, and the processing is executed. When executing the EEL_Execute function, it is possible to specify the execution mode and select the EEPROM emulation execution method. Table 3-3 and Figure 3-1 show the status of each mode.

### Table 3-3. Command Execution Method in Each Mode

<table>
<thead>
<tr>
<th>Execution Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enforced mode</td>
<td>All the processing of specified commands is executed using only the EEL_Execute function. The control does not return from the function until all the command processing finishes.</td>
</tr>
<tr>
<td>Timeout mode</td>
<td>Note  The processing of specified commands is continuously executed until the timeout value specified for the EEL_Execute function is set to 0 by the EEL_TimeOut_CountDown function. If there is remaining processing when the timeout occurs, the processing is continued by using the EEL_Handler function. If all the processing finishes before the timeout value reaches 0, the function terminates and the control returns to the user-created program, but, when using this mode, be sure to use the EEL_TimeOut_CountDown function to make it possible for the timeout value to reach 0.</td>
</tr>
<tr>
<td>Polling mode</td>
<td>Note  The processing of specified commands is executed using a certain unit. After executing the EEL_Execute function, the processing is continued by using the EEL_Handler function.</td>
</tr>
</tbody>
</table>

Note The execution mode can be re-specified when executing EEL_Handler.
### Polling mode

- **Start of command**
- **Command specification**
  - `EEL_Execute`
  - `EEL_Handler`
- **Status check**
  - `eel_request_t.status`
- **Other**
- **EEL_BUSY**

*In the polling mode, not all the processing is finished by the `EEL_Execute` function, and it is necessary to execute the `EEL_Handler` function when continuing the specified processing.*

### Timeout mode

- **Start of command**
- **Command specification**
  - `EEL_Execute`
  - `EEL_Handler`
- **Status check**
  - `eel_request_t.status`
- **Other**
- **EEL_BUSY**

*The processing continues until the specified timeout value reaches 0. Control is returned to the user-created program when either the timeout value reaches 0 or the processing finishes.*

### Enforced mode

- **Start of command**
- **Command specification**
  - `EEL_Execute`
- **End of command**

*The control is not returned until all the instructions finish.*

### Countdown processing

A function for counting down by using a timer or other means is called from a separate user-created program, and the specified timeout value is set to 0. (Execution is possible during interrupt servicing.) If all the processing finishes before 0 is reached, it is not necessary to call the function for counting down and set the timeout value to 0.

- **Countdown processing**
  - `EEL_TimeOut_CountDown`

*The specified timeout value is decremented (−1) each time the countdown function is called.*

**Remaining timeout value = current timeout value − 1**

### Note

The execution mode can be re-specified when executing `EEL_Handler`. 
[Register status after calling this function]
Corrupted register: AX (argument)

[Argument]

eel_request_t Details

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eel_request_t.address_pu8</td>
<td>eel_u08 *</td>
<td>Data buffer for storing write and read data Note</td>
</tr>
<tr>
<td>eel_request_t.identifier_u8</td>
<td>eel_u08</td>
<td>Parameter for setting command to be executed</td>
</tr>
<tr>
<td>eel_request_t.timeout_u8</td>
<td>eel_u08</td>
<td>Timeout value (command execution mode setting)</td>
</tr>
<tr>
<td>eel_request_t.command_enu</td>
<td>eel_command_t</td>
<td>Command to be executed</td>
</tr>
<tr>
<td>eel_request_t.status_enu</td>
<td>eel_status_t</td>
<td>Command execution status</td>
</tr>
</tbody>
</table>

Note Specify this parameter only for a command that requires the parameter. Set up the data buffer size according to the byte sizes of the write and read data.

Timeout Value (timeout_u08) Setting Details

<table>
<thead>
<tr>
<th>Timeout Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFF</td>
<td>Processing is executed in the enforced mode.</td>
</tr>
<tr>
<td>0x01 to 0xFE</td>
<td>Processing is executed in the timeout mode.</td>
</tr>
<tr>
<td>0x00</td>
<td>Processing is executed in the polling mode.</td>
</tr>
</tbody>
</table>

Execution Commands (eel_command_t)

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_CMD_STARTUP</td>
<td>This command checks the block status and sets the system to the EEPROM emulation start (started) status. If all the blocks are active, the oldest active block is forcibly erased to create a prepared block. Be sure to execute this command before executing commands other than the EEL_CMD_FORMAT command, which is used when initializing (erasing) Note 2 EEPROM emulation blocks, and make sure that the command finishes normally.</td>
</tr>
<tr>
<td>EEL_CMD_WRITE Note 1</td>
<td>This command writes the specified data to the EEPROM emulation blocks. * The following arguments must be specified prior to execution. • eel_request_t.address: Specifies the start address of the RAM area where the write data is stored. • eel_request_t.identifier: Specifies the data ID of the write data.</td>
</tr>
<tr>
<td>EEL_CMD_READ Note 1</td>
<td>This command reads the latest data from the EEPROM emulation blocks corresponding to the specified data ID. * The following arguments must be specified prior to execution. • eel_request_t.address: Specifies the address of the RAM area from which the data are to be read. • eel_request_t.identifier: Specifies the data ID of the data to be read.</td>
</tr>
<tr>
<td>EEL_CMD_CLEANUP Notes 1, 3</td>
<td>This command moves only the latest EEPROM emulation block data to the newly created active block and initializes (erases) Note 2 all the other blocks, which are unnecessary. If all the blocks are active, the oldest active block is forcibly erased to create a prepared block, and then the processing is executed.</td>
</tr>
</tbody>
</table>
### Command Description

**EEL_CMD_FORMAT**

This command initializes (erases) everything, including the data recorded in the EEPROM emulation blocks. Be sure to issue this command before using EEPROM emulation for the first time. Note that issuing this command is also necessary to initialize all blocks if a malfunction occurs in an EEPROM emulation block (such as a valid block disappearing) or the initial values (those which are fixed values that cannot be changed) are modified.

Because EEPROM emulation switches to the stopped status (opened) regardless of the results after the processing finishes, execute the EEL_CMD_STARTUP command to continue using EEPROM emulation.

**EEL_CMD_SHUTDOWN**

This command sets EEPROM emulation to the stopped status (opened).

#### Notes

1. Do not execute this command until the EEL_CMD_STARTUP command has finished normally.
2. Blocks for which usage is prohibited are not initialized (erased).
3. For the detailed operations, see [1.4.1 Adjusting Blocks by Using the EEL_CMD_CLEANUP Command](#).
4. For the detailed operations, see [1.3 Initializing EEPROM Emulation Blocks](#).

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Argument Type/Register</th>
<th>Assembly Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENESAS small and medium model</td>
<td>eel_request_t* request</td>
<td>AX (0 to 15)</td>
</tr>
<tr>
<td>RENESAS large model</td>
<td>eel_request_t* request</td>
<td>AX (0 to 15)</td>
</tr>
</tbody>
</table>

### Command Execution Statuses (eel_status_t)

<table>
<thead>
<tr>
<th>Command Execution Status</th>
<th>Description</th>
<th>Corresponding Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_OK</td>
<td>Normal end</td>
<td>All commands</td>
</tr>
<tr>
<td>EEL_BUSY</td>
<td>A command is being executed.</td>
<td>All commands</td>
</tr>
<tr>
<td>EEL_ERR_INITIALIZATION</td>
<td>Initialization error: The FAL_Init(), EEL_Init(), or EEL_Open function has not finished normally.</td>
<td>All commands</td>
</tr>
<tr>
<td>EEL_ERR_ACCESS_LOCKED</td>
<td>EEPROM emulation lock error: EEPROM emulation cannot be executed. Make sure that the EEL_CMD_STARTUP command has finished normally.</td>
<td>Commands other than EEL_CMD_STARTUP</td>
</tr>
<tr>
<td>EEL_ERR_COMMAND</td>
<td>Command error: A command that does not exist has been specified.</td>
<td>–</td>
</tr>
<tr>
<td>EEL_ERR_PARAMETER</td>
<td>Parameter error: An incorrect command parameter has been specified. Check the specified parameter.</td>
<td>All commands</td>
</tr>
<tr>
<td>EEL_ERR_REJECTED</td>
<td>Reject error: A different command is being executed.</td>
<td>All commands</td>
</tr>
<tr>
<td>EEL_ERR_NO_INSTANCE</td>
<td>Identifier error: The specified data is not in the descriptor table or data are not recorded.</td>
<td>EEL_CMD_READ</td>
</tr>
<tr>
<td>Command Execution Status</td>
<td>Description</td>
<td>Corresponding Commands</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>-------------------------------</td>
</tr>
<tr>
<td>EEL_ERR_POOL_FULL</td>
<td>Pool full error: There is no area that can be used to write the data. This error can be recovered by using the EEL_CMD_STARTUP or EEL_CMD_CLEANUP command to forcibly erase the oldest block, but some of the data might disappear even if this is done.</td>
<td>EEL_CMD_WRITE</td>
</tr>
<tr>
<td>EEL_ERR_POOL_INCONSISTENT</td>
<td>EEPROM emulation block inconsistency error: An EEPROM emulation block has the undefined status (such as there are no active blocks). Execute the EEL_CMD_FORMAT command to initialize the EEPROM emulation blocks.</td>
<td>EEL_CMD_STARTUP</td>
</tr>
<tr>
<td>EEL_ERR_POOL_EXHAUSTED</td>
<td>EEPROM emulation block exhaustion error: There are no more EEPROM emulation blocks that can be used to continue. Stop EEPROM emulation.</td>
<td>Commands other than EEL_CMD_READ and EEL_CMD_SHUTDOWN</td>
</tr>
<tr>
<td>EEL_ERR_INTERNAL</td>
<td>Internal error: An unexpected error has occurred. Check the device status. In addition, if this error occurred during EEL_CMD_SHUTDOWN command execution, execute the EEL_Close function to stop EEPROM emulation.</td>
<td>Commands other than EEL_CMD_READ and EEL_CMD_WRITE</td>
</tr>
</tbody>
</table>

[Return value]

None
EEL_Handler

[Outline]
Processing to continue executing EEPROM emulation when EEL_Execute is executed in a mode other than the enforced mode, or the maintenance mode execution processing.

[Format]

<C language>
void __far EEL_Handler(eel_u08 timeout_u08);

<Assembler>
CALL !_EEL_Handler or CALL !!_EEL_Handler

Remark  Call this function with “!” if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with “!!” otherwise.

[Advance setting]
The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]
This function continues executing the EEPROM emulation processing specified for the EEL_Execute function. Note 1 In addition, by executing this function while no commands are being executed, it is possible to adjust EEPROM emulation blocks by using the maintenance mode. Note 2

Notes 1. The command execution status for the EEL_Handler function is specified for eel_request_t* request, which is used as the EEL_Execute function argument. Therefore, when using EEL_Handler, do not free the eel_request_t* request variable.
2. For details about the maintenance mode, see 1.4.2 Adjusting Blocks by Using EEL_Handler Function.
[Register status after calling this function]
Corrupted register: AX (argument)

[Argument]

<table>
<thead>
<tr>
<th>Timeout Value</th>
<th>Description</th>
</tr>
</thead>
</table>
| 0x01 to 0xFF  | Executing processing continues in the timeout mode.  
                (It is necessary to use the EEL_TimeOut_CountDown function to set the timeout value to 0.) |
| 0x00          | Executing processing continues in the polling mode. |

**Note** For details about each mode, see the description of the EEL_Execute function.

### Development Tool

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Argument Type/Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENESAS small and medium model</td>
<td>eel_u08 timeout_u08</td>
</tr>
<tr>
<td>RENESAS large model</td>
<td>eel_u08 timeout_u08</td>
</tr>
</tbody>
</table>

### Return value

Post-execution status information is specified for eel_request_t* request, an argument of the EEL_Execute function.

<table>
<thead>
<tr>
<th>Data to be Input to eel_request_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>Argument</td>
</tr>
<tr>
<td>eel_request_t.status</td>
</tr>
<tr>
<td>Type</td>
</tr>
<tr>
<td>eel_status_t</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Command execution status</td>
</tr>
</tbody>
</table>
EEL_TimeOut_CountDown

[Outline]
Timeout counting processing for EEPROM emulation command execution

[Format]

<C language>

```c
void __far EEL_TimeOut_CountDown(void)
```

<Assembler>

```assembly
CALL !EEL_TimeOut_CountDown or CALL !!EEL_TimeOut_CountDown
```

Remark Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
The FAL_Init, EEL_Init, and EEL_Open functions must have finished normally.

[Function]
This function is used when executing EEPROM emulation in the timeout mode.
When this function is executed, the timeout value specified when executing an EEPROM emulation command is decremented (−1), and the loop processing of the EEL_Execute and EEL_Handler functions ends when the timeout value reaches 0.
For details about handling the timeout value, see the sections that describe the EEL_Execute and EEL_Handler functions.

[Register status after calling this function]
No registers are corrupted.

[Argument]
None

[Return value]
None
**EEL_GetDriverStatus**

[Outline]
This obtains the EEPROM emulation library status.

[Format]

<C language>

```c
void __far EEL_GetDriverStatus(__near eel_driver_status_t *driverStatus_pstr)
```

<Assembler>

```assembly
CALL !_EEL_GetDriverStatus or CALL !!_EEL_GetDriverStatus
```

**Remark** Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
The FAL_Init and EEL_Init functions must have finished normally.

[Function]
This function obtains the status of the EEPROM emulation library. This makes it possible to check the status of the EEL_Handler function and EEPROM emulation.

[Register status after calling this function]
No registers are corrupted.

[Argument]

<table>
<thead>
<tr>
<th>Argument</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eel_driver_status_t.operationStatus_enu</td>
<td>eel_operation_status_t</td>
<td>EEL_Handler() execution status</td>
</tr>
<tr>
<td>eel_driver_status_t.accessStatus_enu</td>
<td>eel_access_status_t</td>
<td>EEPROM emulation status</td>
</tr>
<tr>
<td>eel_driver_status_t.backgroundStatus_enu</td>
<td>eel_status_t</td>
<td>Library status (not available for users)</td>
</tr>
</tbody>
</table>

**eel_operation_status_t Details**

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_OPERATION_PASSIVE</td>
<td>The EEL_CMD_STARTUP command has not finished normally when no commands are being executed.</td>
</tr>
<tr>
<td>EEL_OPERATION_IDLE</td>
<td>No command or maintenance mode processing is being executed.</td>
</tr>
<tr>
<td>EEL_OPERATION_BUSY</td>
<td>Command or maintenance mode processing is being executed.</td>
</tr>
</tbody>
</table>

**Note** For details about the maintenance mode, see 1.4.2 Adjusting Blocks by Using the EEL_Handler Function.
### eel_access_status_t Details

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EEL_ACCESS_LOCKED</td>
<td>Data reading and writing cannot be executed (opened, closed).</td>
</tr>
<tr>
<td>EEL_ACCESS_UNLOCKED</td>
<td>Data reading and writing can be executed (started).</td>
</tr>
</tbody>
</table>

#### Development Tool

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Argument Type/Register</th>
</tr>
</thead>
<tbody>
<tr>
<td>RENESAS small and medium model</td>
<td>__near eel_driver_status_t *driverStatus_pstr</td>
</tr>
<tr>
<td>RENESAS large model</td>
<td>__near eel_driver_status_t *driverStatus_pstr</td>
</tr>
<tr>
<td></td>
<td>AX (0 to 15)</td>
</tr>
</tbody>
</table>

[Return value]
- None
# EEL_GetSpace

[Outline]
This obtains the free EEPROM emulation block space (in word units).

[Format]

**<C language>**
```c
eel_status_t __far EEL_GetSpace(__near eel_u16* space_pu16)
```

**<Assembler>**
```
CALL !_EEL_GetSpace or CALL !!_EEL_GetSpace
```

**Remark** Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
The FAL_Init, EEL_Init, and EEL_Open functions, and the EEL_CMD_STARTUP command must have finished normally.

[Function]
This function obtains the free EEPROM emulation block space (in word units).

[Register status after calling this function]
Return value: C
Corrupted register: AX (argument)

[Argument]

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__near eel_u16*</td>
<td>The address at which the information about the total free space of the current active block and prepared blocks is input (a 2 bytes area): The data is in word units (1 word = 4 bytes).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Development Tool</th>
<th>Argument Type/Register</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C Language</td>
</tr>
<tr>
<td>RENESAS small and medium model</td>
<td>__near eel_u16* space_pu16</td>
</tr>
<tr>
<td>RENESAS large model</td>
<td>__near eel_u16* space_pu16</td>
</tr>
</tbody>
</table>

[Return value]

<table>
<thead>
<tr>
<th>Type</th>
<th>Symbol Definition</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eel_status_t</td>
<td>EEL_OK</td>
<td>Acquisition succeeded.</td>
</tr>
<tr>
<td></td>
<td>EEL_ERR_INITIALIZATION</td>
<td>EEL_Init has not been executed.</td>
</tr>
<tr>
<td></td>
<td>EEL_ERR_ACCESS_LOCKED</td>
<td>The EEL_CMD_STARTUP command has not finished normally.</td>
</tr>
<tr>
<td></td>
<td>EEL_ERR_REJECTED</td>
<td>A command is being executed.</td>
</tr>
</tbody>
</table>

**Remark** Assembly language return values are stored in register C.
EEL_GetVersionString

[Outline]
This obtains the version information of the EEPROM emulation library (EEL).

[Format]

<C language>

```c
__far eel_u08* __far EEL_GetVersionString(void)
```

<Assembler>

CALL !_EEL_GetVersionString or CALL !!_EEL_GetVersionString

**Remark** Call this function with "!" if you are placing the EEPROM emulation library at 00000H to 0FFFFH or with "!!" otherwise.

[Advance setting]
The FAL_Init and EEL_Init functions must have finished normally.

[Function]
None

[Register status after calling this function]
Return value: BC and DE
Corrupted registers: BC and DE

[Argument]
None

[Return value]

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>eel_u08*</td>
<td>The address at which the version information of the EEPROM emulation library (EEL) is input (a 24-bit address area)</td>
</tr>
</tbody>
</table>

Example: For EEPROM emulation library Pack01 V1.14 (ASCII code)

```
"ERL78T01R110GVxxx"
```

- Version information: V114 → V1.14
- Corresponding tool: Renesas Electronics version
- Type name: Type 01
- Corresponding device: RL78
- Target library: EEL
# APPENDIX A REVISION HISTORY

## A. 1 Major Revisions in This Edition

<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
<th>Classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughout the document</td>
<td>− Statements of the target devices were deleted.</td>
<td>(c)</td>
</tr>
<tr>
<td></td>
<td>− References to the list of target MCUs were added.</td>
<td>(e)</td>
</tr>
<tr>
<td>Chapter 1 OVERVIEW OF EEPROM EMULATION</td>
<td>p.2 In figures 1-1 and 1-2, the description “flash data library” was corrected to “data flash library”.</td>
<td>(a)</td>
</tr>
<tr>
<td>Chapter 2 USING EEPROM EMULATION</td>
<td>p.36 The version number was added to the heading for table 2-2.</td>
<td>(c)</td>
</tr>
<tr>
<td></td>
<td>p.36 The description of the self-RAM area in table 2-2 was changed.</td>
<td>(c)</td>
</tr>
<tr>
<td></td>
<td>p.36 Note 1 for table 2-2 was changed to include a statement regarding inquiries about device specifications.</td>
<td>(c)</td>
</tr>
<tr>
<td></td>
<td>p.36 The previous note 2 under table 2-2 was deleted.</td>
<td>(c)</td>
</tr>
<tr>
<td></td>
<td>p.36 The previous note 4 under table 2-2 was deleted.</td>
<td>(c)</td>
</tr>
<tr>
<td>Chapter 3 EEPROM EMULATION FEATURES</td>
<td>p.52 The description in the table was changed.</td>
<td>(c)</td>
</tr>
</tbody>
</table>

**Remark**

Symbols under “Classification” in the above table are used to classify revisions as follows.

- (a): Correction of errors
- (b): Addition or change to specifications
- (c): Addition or change to descriptions or notes
- (d): Addition or change to packages, part numbers, or management divisions
- (e): Addition or change to related documents
## A. 2 Revision History of Previous Editions

The following shows the revision history of the previous editions. The Chapter column indicates the chapter in the edition.

<table>
<thead>
<tr>
<th>Rev. No.</th>
<th>Description</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.02</td>
<td>The document on the data flash library, which was classified as the application note (old version of R01AN0351), was changed to the user’s manual. The corresponding ZIP file name and release version were added to the cover page. Contents of the notes, processing time, and software resources were moved from the usage note to this document. Accordingly, the reference destination described in this document was also changed. The supported device was added. The notation of high-speed OCO was deleted to unify the notation of high-speed on-chip oscillator. The description of the operating frequency was unified to the CPU operating frequency since individual descriptions had different notations. Errors in the capacity of the separator and the calculation method for the maximum size were modified.</td>
<td>Throughout the document</td>
</tr>
<tr>
<td></td>
<td>Note on the frequency of the high-speed on-chip oscillator was added.</td>
<td>Chapter 1 OVERVIEW OF EEPROM EMULATION</td>
</tr>
<tr>
<td></td>
<td>Note on the RAM parity error was added.</td>
<td>Chapter 2 USINGEEPROM EMULATION</td>
</tr>
<tr>
<td></td>
<td>Note on the data flash control register (DFLCTL) was added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note on the number of data flash memory blocks was added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Items regarding the processing time were added (the description of the processing time was moved from the usage note to this document).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Items regarding the resources were added and description was also changed (the description on the resources was moved from the usage note to this document).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Note on the high-speed on-chip oscillator was added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description of the return value of the Fal_Init function was added.</td>
<td>Chapter 3 EEPROM EMULATION FEATURES</td>
</tr>
<tr>
<td></td>
<td>Description of the argument of the EEL_Execute function was added.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Description of the return value of the EEL_GetVersionString function was changed.</td>
<td></td>
</tr>
</tbody>
</table>