RX Family

Virtual EEPROM Module Using Firmware Integration Technology

R01AN4142EU0111
Rev. 1.11
Oct 18, 2018

Introduction

The Virtual EEPROM (VEE) Module Using Firmware Integration Technology (FIT) has been developed to allow users of supported RX devices to easily emulate EEPROM functionality while only using on-chip data flash.

The source files accompanying the VEE module comply with the Renesas RX compiler only.

Target Device

The following is a list of devices that are currently supported by this API:

- Flash Types 1, 3, and 4 (all currently promoted MCUs; see R01AN2184EU for latest Flash Type MCU list).

Related Documents

- Firmware Integration Technology User's Manual (R01AN1833EU)
- Board Support Package Firmware Integration Technology Module (R01AN1685EU)
- Flash Module Using Firmware Integration Technology Module (R01AN2184EU)
- Adding Firmware Integration Technology Modules to Projects (R01AN1723EU)
- Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826EJ)

Contents

1. Overview 3
1.1 Features 3
1.2 Data Flash Segmentation 3
1.3 Record Format 4
1.4 Reference Data Area 4

2. API Information 6
2.1 Hardware Requirements 6
2.2 Software Requirements 6
2.3 Limitations 6
2.4 Supported Toolchains 6
2.5 Header Files 6
2.6 Integer Types 6
2.7 Configuration Overview 6
2.8 Code Size 8
2.9 API Data Types 8
2.10 Return Values 8
2.11 Adding the FIT VEE Module to Your Project 9
2.11.1 Adding source tree and project include paths 9
2.11.2 Setting driver use options 9

3. API Functions 10
3.1 Summary 10
3.2 R_VEE_Open() 11
3.3 R_VEE_WriteRecord() 12
3.4 R_VEE_GetRecordPtr() 14
3.5 R_VEE_WriteRefData() 15
3.6 R_VEE_GetRefDataPtr() 16
3.7 R_VEE_Control() 17
3.8 R_VEE_Close() 21
3.9 R_VEE_GetVersion() 22

4. Demo Projects 23
4.1 vee_demo_rskrx231 23
4.2 Adding a Demo to a Workspace 23

Appendix: Error Modes 24

Website and Support 25

Revision Record 26

General Precautions in the Handling of MPU/MCU Products 27

1. Overview

This VEE module emulates basic Virtual EEPROM capabilities. Support is provided for reading and writing both common records and reference data (originally programmed during product assembly or test). Records can be configured to be fixed length or variable length. A count of the number of segments erased throughout the lifetime of the application is maintained and can be accessed at any time. Wear leveling is handled automatically by the driver.

This driver supports flash types 1, 3, and 4 (MCUs RX111/113/130, RX231/24T/24U, RX64M/71M, RX65N-2M, but not RX110/23T/65N-1M which have no data flash).

1.1 Features

Below is a list of the features supported by the VEE module.

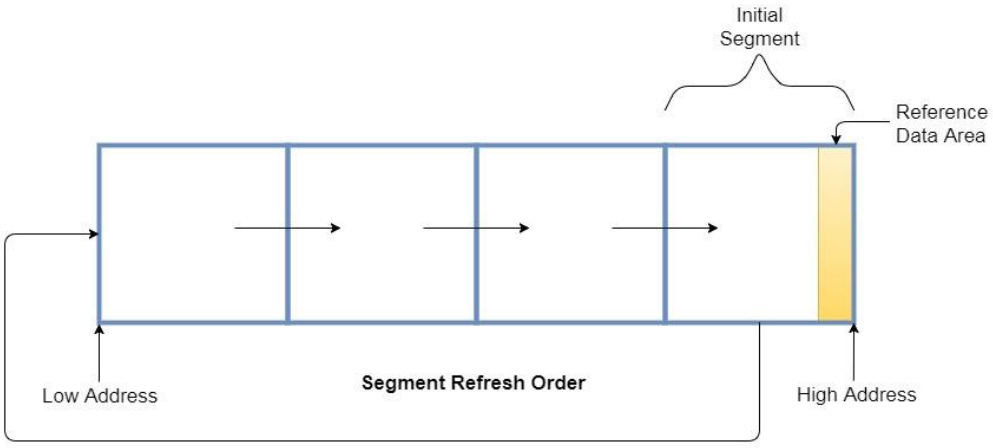
- Writing and reading user defined records to data flash.
- Records can be fixed length or variable length.
- Wear leveling is handled automatically.
- Reference data such as calibration data programmed at assembly or test time is preserved.
- Reference data can be updated at run time.

1.2 Data Flash Segmentation

Wear leveling is handled by changing the location in the data flash a record is stored when it is updated. This change in physical location of the record is transparent to the user. Any time an update for a specific record ID is written, it is written to the next unused location in data flash and its location is stored in RAM for quick look-up later. Periodically, only the most recent version of these records is copied to the next blank segment in data flash.

The data flash area is divided into a number of equal-size segments as specified in "r_vee_rx_config.h". The default is two 4K segments. There is only one segment active at a time. A segment contains two areas- the record area (which is the vast majority of the segment) and the reference data area which contains optional data typically programmed during assembly or final test. Records and updated reference data are written to this segment until one of the two areas becomes full. The record area must be able to hold at least one of every record ID possible and still have space left over for record updates.

When a segment does not have sufficient space for additional records or updated reference data, a Refresh occurs. This process copies the most recent record for each ID as well as the latest version of reference data (if any) to the next segment. The very first time VEE runs on an MCU, it marks the last segment as active whether there is reference data configured or not. The end of data flash memory is used to provide an easily identified physical flash address that can be used while programming reference data without requiring Virtual EEPROM middleware.

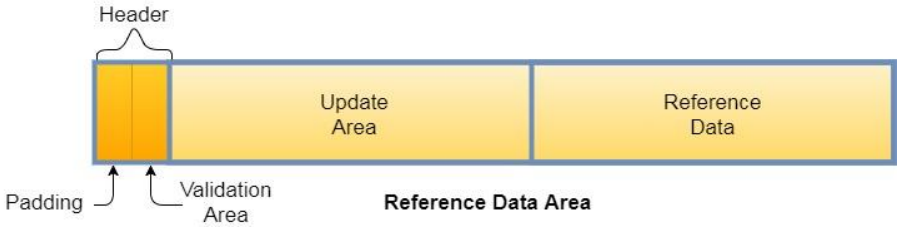


1.3 Record Format

VEE can be configured in "r_vee_rx_config.h" to store fixed length or variable length records. By default, records are configured for variable length. Fixed length and variable length records are stored in the same format in data flash with the exception of a header prepended for variable length records which specifies the record's data length. The trailer contains a validation code which is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the record data passed to the driver. Record data is limited to 500 bytes.

1.4 Reference Data Area

VEE can be configured in "r_vee_rx_config.h" for the presence of reference data. By default, no reference data is configured. The original programmed reference data must be located at the end of data flash. An area of equal size is reserved below this in case updated reference data becomes available later. Below that is a header which indicates whether the update area has been written to.



Just as with records, the validation code is used for internal purposes only and is not a 16-bit CRC or ECC value. If that level of error checking is desired, the user should include that in the updated reference data passed to the driver. Reference data is limited to 508 bytes.

2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

This driver requires that your MCU supports the following peripheral(s):

- Data flash

2.2 Software Requirements

This driver makes use of the following FIT packages:

- Renesas Board Support Package v3.60 (optional)
- FIT Flash Module v3.30 (this version of FIT Flash Module adds a “Close” function required by VEE module)

2.3 Limitations

- This code is not re-entrant and protects against multiple concurrent function calls

2.4 Supported Toolchains

This driver is tested and working with the following toolchains:

- Renesas RX Toolchain v2.07.00

2.5 Header Files

All API calls and their supporting interface definitions are located in “r_vee_rx_if.h”. This file should be included by all files which utilize the Flash API.

Build-time configuration options are selected or defined in the file "r_vee_rx_config.h”.

2.6 Integer Types

This project uses ANSI C99 “Exact width integer types” in order to make the code clearer and more portable. These types are defined in stdint.h.

2.7 Configuration Overview

Configuring this module is done through the supplied r_vee_rx_config.h header file. Each configuration item is represented by a macro definition in this file. Each configurable item is detailed in the table below.

Configuration options in *r_vee_rx_config.h*		
Equate	**Default Value**	**Description**
VEE_CFG_PARAM_CHECKING_ENABLE	1	Setting to 1 includes parameter checking. 0 compiles out parameter checking.
VEE_CFG_NUM_SEGMENTS	MCU_DATA_FLASH_SIZE_BYTES / 4096	Set value to number of segments desired in data

		flash (minimum 2). The fewer the segments, the fewer refreshes occur, but the longer refreshes take to complete (erase time).
VEE_CFG_REF_DATA_SIZE	0	Set value to size of reference data at the end of data flash. A value of 0 implies no reference data is present. Size must be a multiple of FLASH_DF_MIN_PGM_SIZE.
VEE_CFG_RECORD_FIXED_SIZE	0	Set value to record data size. Setting to 0 implies variable length records will be used. Using fixed length records optimizes speed and flash space used. Size must be a multiple of FLASH_DF_MIN_PGM_SIZE.
VEE_CFG_RECORD_MAX_ID	16	IDs must be consecutive starting with 0. An array is maintained with the locations of the most recent record for each ID. Set this value to the highest ID in use.
VEE_CFG_REFRESH_BUF_SIZE	32	An internal buffer is needed when copying data from one flash segment to another during a Refresh operation. The most efficient buffer size matches the largest record size in use + 4 bytes (or + 8 bytes for variable length records). Size must be a multiple of FLASH_DF_MIN_PGM_SIZE.
VEE_CFG_FLASH_INT_PRIORITY	8	Flash operations occur in BGO (interrupt) mode. Set flash interrupt priority from 1 (lowest) to 15 (highest).
VEE_CFG_ALLOW_GETS_AFTER_FAIL	0	Set to 1 if desire to call GetRecordPtr() or GetRefDataPtr (do flash reads) after a system error has been detected. Set to 0 to prohibit any API calls except for Close() and the Control() command Get Status.

Table 1: VEE general configuration settings

2.8 Code Size

The code size is based on optimization level 2 and optimization type for size for the RXC toolchain specified in Section 2.4.

VEE ROM and RAM usage	
ROM usage: **PARAM_CHECKING_ENABLE 1 > PARAM_CHECKING_ENABLE 0** **VEE_CONFIG_REF_DATA_SIZE non-zero > VEE_CONFIG_REF_DATA_SIZE 0** **VEE_CFG_RECORD_FIXED_SIZE 0 > VEE_CFG_RECORD_FIXED_SIZE non-zero** **VEE_CFG_ALLOW_GETS_AFTER_FAIL 1 > VEE_CFG_ALLOW_GETS_AFTER_FAIL 0**	
Minimum Size	ROM: 2071 bytes
	RAM: 97 bytes + VEE_CFG_REFRESH_BUF_SIZE + (VEE_CFG_RECORD_MAX_ID + 1) * 2
Maximum Size	ROM: 2508 bytes
	RAM: 97 bytes + VEE_CFG_REFRESH_BUF_SIZE + (VEE_CFG_RECORD_MAX_ID + 1) * 2

2.9 API Data Types

The API data structures are located in the file “r_vee_rx_if.h” and discussed in Section 3.

2.10 Return Values

This shows the different values API functions can return. This return type is defined in “r_vee_rx_if.h”.

```
typedef enum e_vee_err
{
    VEE_SUCCESS = 0,
    VEE_READY = 0,              // returned by VEE_CMD_GET_STATUS
    VEE_SUCCESS_RECOVERY,       // Open() succeeded; corrupted data erased
    VEE_ERR_CORRUPTION_FOUND,   // (for driver internal use only)
    VEE_ERR_ALREADY_OPEN,       // Open() called again without a Close()in between
    VEE_ERR_ILLEGAL_CONFIG,     // invalid setting in vee or flash config.h file
    VEE_ERR_UNKNOWN_CMD,        // unrecognized Control() command
    VEE_ERR_NULL_PTR,           // missing argument pointer
    VEE_ERR_ILLEGAL_ARG,        // parameter value invalid
    VEE_ERR_FLASH_INIT_ERR,     // Flash FIT driver could not open properly
    VEE_ERR_BUSY,               // another VEE operation is still executing
    VEE_ERR_BUSY_REFRESH,       // a Refresh operation is in progress
    VEE_ERR_RECORD_NOT_FOUND,   // no record has been written for specified ID
    VEE_ERR_OVERFLOW,           // segments too small to hold 1 record for each ID
    VEE_ERR_FLASH_PE_FAIL,      // should never happen; flash hardware failure
    VEE_ERR_TIMEOUT,            // should never happen; interrupts disabled by app
    VEE_ERR_UNKNOWN_WRITE       // should never happen; development error
} vee_err_t;
```

2.11 Adding the FIT VEE Module to Your Project

For detailed explanation of how to add a FIT Module to your project, see document R01AN1723EU “Adding FIT Modules to Projects”.

2.11.1 Adding source tree and project include paths

In general, a FIT Module may be added in 3 ways:

1. Using an e2studio FIT tool, such as File>New>Renesas FIT Module (prior to v5.3.0), Renesas Views->e2 solutions toolkit->FIT Configurator (v5.3.0 or later), or projects created using the Smart Configurator (v5.3.0 or later). This adds the module and project include paths.
2. Using e2studio File>Import>General>Archive File from the project context menu.
3. Unzipping the .zip file into the project directory directly from Windows.

When using methods 2or 3, the include paths must be manually added to the project. This is done in e2studio from the project context menu by selecting Properties>C/C++ Build>Settings and selecting Compiler>Source in the ToolSettings tab. The green “+” sign in the box to the right is used to pop a dialog box to add the include paths. In that box, click on the Workspace button and select the directories needed from the project tree structure displayed. The directories needed for this module are:

- ${workspace_loc:/${ProjName}/r_vee_rx
- ${workspace_loc:/${ProjName}/r_vee_rx/src
- ${workspace_loc:/${ProjName}/r_config

2.11.2 Setting driver use options

The VEE-specific options are found and edited in \r_config\r_vee_rx_config.h.

A reference copy (not for editing) containing the default values for this file is stored in \r_vee_rx\ref\r_vee_rx_config_reference.h.

The FIT Flash Module configuration file \r_config\r_flash_rx_config.h must be configured for BGO (interrupt) operation.

Similarly, the Debug Configuration for the project must allow writes to data flash. (See application note Flash Module Using Firmware Integration Technology Module (R01AN2184EU) for complete Flash driver installation and configuration descriptions.)

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_VEE_Open()	Initializes the driver's internal structures and opens the FIT Flash driver.
R_VEE_WriteRecord ()	Writes a record to data flash.
R_VEE_GetRecordPtr()	Gets the pointer to the most recent version of a given record.
R_VEE_WriteRefData()	Writes new Reference data to the reference update area in data flash.
R_VEE_GetRefDataPtr()	Gets a pointer to the most recent valid reference data.
R_VEE_Control()	Performs special operations.
R_VEE_Close()	Closes the Flash driver and VEE driver.
R_VEE_GetVersion()	Returns the version of the driver.

3.2 R_VEE_Open()

Initializes the driver's internal structures and opens the FIT Flash driver.

Format

```
vee_err_t R_VEE_Open(void);
```

Parameters

None

Return Values

VEE_SUCCESS:	*Successful*
VEE_SUCCESS_RECOVERY:	*Successful, but corrupted data found and erased*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_ALREADY_OPEN:	*This function has already been called*
VEE_ERR_FLASH_INIT_ERR:	*Error opening or configuring Flash driver*
VEE_ERR_TIMEOUT:	*Interrupts disabled outside of VEE*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

Initializes the driver's internal structures and opens the FIT Flash driver. The FIT Flash driver must be closed prior to opening VEE.

The error code VEE_SUCCESS_RECOVERY indicates that VEE detected corrupted data; most likely due to a power loss during a data flash write or erase. In these cases, an automatic internal Refresh is performed and the partially written data is lost. The Control() command VEE_CMD_GET_LAST_ID_WRITTEN will indicate the last record ID successfully written. The function R_VEE_GetRefDataPtr() provides a pointer to the most recent good reference data available.

Reentrant

No.

Example:

```
    vee_err_t err;

    err = R_VEE_Open();
    if ((err != VEE_SUCCESS) || (err != VEE_SUCCESS_RECOVERY))
    {
        while(1) ;      // fatal error
    }
```

Special Notes:

None.

3.3 R_VEE_WriteRecord()

Writes a record to data flash.

Format

```
vee_err_t R_VEE_WriteRecord(uint16_t rec_id,
                            uint8_t *rec_data_ptr,
                            uint16_t num_bytes);
```

Parameters

rec_id
ID of record to write.

rec_data_ptr
Pointer to record data to write.

num_bytes
Length of data to write.

Return Values

VEE_SUCCESS:	*Write started successfully*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_NULL_PTR:	*"rec_data_ptr" is NULL*
VEE_ERR_ILLEGAL_ARG:	*Invalid ID or "num_bytes" value*
VEE_ERR_OVERFLOW:	*Should never happen. See appendix.*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

This function writes "num_bytes" of data pointed to by "rec_data_ptr" to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes. This includes exiting the calling function when the data buffer is a local variable (stack may be used by another function and corrupt the data buffer contents). An error code of BUSY will be returned by other API calls until this write completes. Note that if insufficient record space is available to write the record, a segment Refresh process is automatically started. Any other API calls made prior to the Refresh process completing will return BUSY REFRESH.

Reentrant

No

Example:

```
    vee_err_t        err;
    uint8_t          rec_data[TEMPERATURE_DATA_SIZE];
    struct temp_data rec_data2;
    uint16_t         rec_length;

    /* Example: record data is an array of bytes */
    err = R_VEE_WriteRecord(ID_TEMPERATURE, rec_data, TEMPERATURE_DATA_SIZE);

    /* Example: record data has a structure format */
    err = R_VEE_WriteRecord(ID_TEMPERATURE,
                            (uint8_t *)&rec_data2,
                            sizeof(rec_data2));
```

Special Notes:
If a refresh is triggered, the VEE will be locked for an extended amount of time and will return BUSY REFRESH until completed.

3.4 R_VEE_GetRecordPtr()

This function gets the pointer to the most recent version of a record specified by ID.

Format

```
vee_err_t R_VEE_GetRecordPtr(uint16_t rec_id,
                             uint8_t  **rec_data_ptr,
                             uint16_t *num_bytes_ptr);
```

Parameters

rec_id

ID of record to locate.

rec_data_ptr

Pointer to set to the most recent version of the record.

num_bytes_ptr

Variable to load with record length.

Return Values

VEE_SUCCESS:	*Successful*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_NULL_PTR:	*"rec_data_ptr" or "num_bytes_ptr" is NULL*
VEE_ERR_ILLEGAL_ARG:	*ID value is out of range*
VEE_ERR_RECORD_NOT_FOUND:	*No record found for specified ID*
VEE_ERR_OVERFLOW:	*Should never happen. See appendix.*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

This function sets "rec_data_ptr" to the most recent version of a record specified by "rec_id". The function also loads the variable pointed to by "num_bytes_ptr" with the length of the record data. NOTE: Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at "rec_data_ptr" must be completed prior to initiating any type of Flash write.

Reentrant

No

Example:

```
vee_err_t err;
uint8_t          *rec_ptr;
struct temp_data *rec2_ptr;
uint16_t         rec_length;

/* Example: record is an array of bytes */
err = R_VEE_GetRecordPtr(ID_TEMPERATURE, &rec_ptr, &rec_length);

/* Example: record has a structure format */
err = R_VEE_GetRecordPtr(ID_TEMPERATURE, (uint8_t **)&rec2_ptr, &rec_length);
```

Special Notes:

Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at "rec_data_ptr" must be completed prior to initiating any type of Flash write.

3.5 R_VEE_WriteRefData()

Writes new Reference data to the reference update area.

Format

```
vee_err_t R_VEE_WriteRefData(uint8_t *ref_data_ptr);
```

Parameters

ref_data_ptr

Pointer to data to write to the reference data update area.

Return Values

VEE_SUCCESS:	*Write started successfully*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_NULL_PTR:	*"ref_data_ptr" is NULL*
VEE_ERR_ILLEGAL_CONFIG:	*Reference data not configured in r_vee_rx_config.h*
VEE_ERR_OVERFLOW:	*Should never happen. See appendix.*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

This function writes VEE_CFG_REF_DATA_SIZE bytes pointed to by "ref_data_ptr" to data flash. This function returns immediately after starting the flash write. BE SURE NOT TO MODIFY the data buffer contents until after the write completes. This includes exiting the calling function when the buffer is a local variable (stack may be used by another function and corrupt the data buffer contents). An error code of BUSY will be returned by other API calls until this write is completed. Note that if the reference data has already been updated once since the last segment Refresh, another Refresh process is automatically started. Any other API calls made prior to the Refresh process completing will return BUSY REFRESH.

Reentrant

No

Example:

```
vee_err_t err;
uint8_t      ref_buf[VEE_CFG_REF_DATA_SIZE];
struct refdata  ref_type2;

/* Example: data is in byte array */
err = R_VEE_WriteRefData(ref_buf);

/* Example: data is in a structure */
err = R_VEE_WriteRefData((uint8_t *)&ref_type2);
```

Special Notes:

If a refresh is triggered, VEE will be locked for an extended amount of time and return BUSY REFRESH until completed.

3.6 R_VEE_GetRefDataPtr()

Gets a pointer to the most recent reference data.

Format

```
vee_err_t R_VEE_GetRefDataPtr(uint8_t **ref_data_ptr);
```

Parameters

ref_data_ptr

Pointer to set to the most recent valid reference data.

Return Values

VEE_SUCCESS:	*Successful*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_NULL_PTR:	*"ref_data_ptr" is NULL*
VEE_ERR_ILLEGAL_CONFIG:	*Reference data not configured in r_vee_rx_config.h*
VEE_ERR_OVERFLOW:	*Should never happen. See appendix.*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

This function sets the argument pointer to the most recent version of the reference data in flash. Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at "ref_data_ptr" must be completed prior to initiating any type of Flash write.

Reentrant

No

Example:

```
vee_err_t        err;
uint8_t         *ref_data_ptr;
struct refdata  *ref_type2_ptr;

/* Example: reference data is a byte array */
err = R_VEE_GetRefDataPtr(&ref_data_ptr);

/* Example: reference data is in a structure format */
err = R_VEE_GetRefDataPtr((uint8_t **)&ref_type2_ptr);
```

Special Notes:

Flash cannot be accessed for reading and writing at the same time. Therefore, reading the data at "ref_data_ptr" must be completed prior to initiating any type of Flash write.

3.7 R_VEE_Control()

This function handles special operations.

Format

```
vee_err_t R_VEE_Control(vee_cmd_t cmd,
                        void      *arg_ptr);
```

Parameters

cmd

Specifies which command to run.

```
typedef enum _vee_cmd
{
    VEE_CMD_GET_STATUS,
    VEE_CMD_GET_LAST_ID_WRITTEN,
    VEE_CMD_GET_RECORD_SPACE_AVAIL,
    VEE_CMD_QUERY_REFDATA_UPDATED,
    VEE_CMD_REFRESH,
    VEE_CMD_GET_SEG_ERASE_COUNT,
    VEE_CMD_FORMAT,
    VEE_CMD_END_ENUM
} vee_cmd_t;
```

arg_ptr

Pointer to the argument which is specific to a command.

Return Values

VEE_SUCCESS:	*Successful*
VEE_READY:	*Last API call completed (returned only with GET_STATUS)*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*
VEE_ERR_UNKNOWN_CMD:	*Unrecognized command*
VEE_ERR_NULL_PTR:	*"arg_ptr" is NULL and argument is required for command*
VEE_ERR_ILLEGAL_CONFIG:	*Reference data not configured in r_vee_rx_config.h*
VEE_ERR_RECORD_NOT_FOUND:	*No records found for specified ID*
VEE_ERR_OVERFLOW:	*Should never happen. See appendix.*
VEE_ERR_FLASH_PE_FAIL:	*Should never happen. See appendix.*
VEE_ERR_UNKNOWN_WRITE:	*Should never happen. See appendix.*

Properties

Prototyped in file "r_vee_rx_if.h"

Description

This function is for handling special operations with the VEE driver. A command may or may not require an argument (see examples below).

Reentrant

No

Example: See if last API call completed – Get Status

This command is typically used to verify that the last Write or Refresh command has completed before attempting to perform another API call. Normal return codes are READY, BUSY, or BUSY REFRESH.

```
    vee_err_t     err;
    flash_interrupt_event_t event;

     /* Check that that VEE is not in an error mode */
    err = R_VEE_Control(VEE_CMD_GET_STATUS, &event);
    if ((err == VEE_ERR_OVERFLOW) || (err == VEE_ERR_UNKNOWN_WRITE))
```

```
    {
        while(1) ;      // development error
    }
    else if (err == VEE_ERR_FLASH_PE_FAIL)
    {
        /* trying closing and re-opening VEE to clear error */
    }
```

Example: Get the ID of the last record written

This command loads the ID of the last record written. This call is often used after a reset (and Open call) to determine where a process was before the reset occurred.

```
    vee_err_t err;
    uint16_t  id;

    /* variable "id" loaded with the ID of the last record written*/
    err = R_VEE_Control(VEE_CMD_GET_LAST_ID_WRITTEN, &id);
    if (err != VEE_SUCCESS)
    {
        /* handle error (most likely BUSY; try again later) */
    }
```

Example: Get the number of bytes left in the record area

This command can be used to check the number of unused bytes in the record area. This is useful when it is known that a large amount of data is about to be received, and it is desired to know if there is sufficient space left in the record area to write this data without an automatic refresh being triggered causing a delay in the ability to perform additional writes.

```
    vee_err_t err;
    uint32_t  bytes_available;

    /* "bytes_available" is loaded with number of unused bytes in record area */
    err = R_VEE_Control(VEE_CMD_GET_RECORD_SPACE_AVAIL, &bytes_available);
    if (err != VEE_SUCCESS)
    {
        /* handle error (most likely BUSY; try again later) */
    }

    /* NOTE: When calculating how many records can fit in the remaining bytes,
     * be sure to include 4 bytes of internal overhead per record for fixed
     * length records, and 8 bytes of overhead for variable length records.
     */
```

Example: Determine if reference data already updated

This command serves a similar purpose as the GET RECORD SPACE AVAIL command. Namely, this command is called if there are system timing concerns, and it is desired to know in advance whether a WriteRefData() will initiate a segment Refresh or not.

```
vee_err_t err;
bool is_reference_updated;

/* "is_reference_updated" is loaded with "true" if reference data update area
 * has been used. If true, performing a WriteRefData() will trigger a Refresh.
 */
err = R_VEE_Control(VEE_CMD_QUERY_REFDATA_UPDATED, &is_reference_updated);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}
```

Example: Force a Refresh

This command is used to start a segment Refresh at any time. The Refresh process by default occurs automatically when no more record or reference data space is available and a Write is requested. However, the app may desire to force a refresh when it knows it is running low on space and large amounts of data are about to be recorded.

```
vee_err_t err;

err = R_VEE_Control(VEE_CMD_REFRESH, NULL);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}
```

Example: Get segment erase count

This command returns the number of times segments have been erased. This may be used to estimate data flash life. Note that should a reset occur during a segment erase, the partial erase will not be included in the total count.

```
vee_err_t err;
uint32_t  erase_count;

/* "erase_count" loaded with number of times segments have been erased */
err = R_VEE_Control(VEE_CMD_GET_SEG_ERASE_COUNT, &erase_count);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}
```

Example: Format data flash

This command is meant for development purposes only. It will erase all of data flash, write reference data if configured, and mark the last segment as the active segment.

```
vee_err_t           err;
struct my_refdata   refdata1;
uint8_t             refdata2[VEE_CFG_REF_DATA_SIZE];

// (load reference data here)

/* Call format using reference data structure */
err = R_VEE_Control(VEE_CMD_FORMAT, &refdata1);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}

/* Call format using reference data array */
err = R_VEE_Control(VEE_CMD_FORMAT, refdata2);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}

/* Call format when reference data is not configured in r_vee_rx_config.h */
err = R_VEE_Control(VEE_CMD_FORMAT, NULL);
if (err != VEE_SUCCESS)
{
    /* handle error (most likely BUSY; try again later) */
}
```

Special Notes:
None.

3.8 R_VEE_Close()

Closes the Flash driver and VEE driver.

Format

```
vee_err_t R_VEE_Close(void);
```

Parameters

None

Return Values

VEE_SUCCESS:	*Successful*
VEE_ERR_BUSY:	*Last API call still executing*
VEE_ERR_BUSY_REFRESH:	*A segment refresh is in progress*

Properties

Prototyped in file “r_vee_rx_if.h”

Description

This Function will close the flash driver and the VEE driver. If another API operation is still in progress, a BUSY error is returned. NOTE: If interrupts are disabled outside of VEE, it may be impossible to close the Flash driver (always get BUSY). This type of error should only occur during the user’s development process.

Reentrant

No

Example:

```
vee_err_t err;

err = R_VEE_Close();
```

Special Notes:

None.

3.9 R_VEE_GetVersion()

Returns the version of the driver.

Format

```
uint32_t R_VEE_GetVersion(void);
```

Parameters

None

Return Values

Version Number

Properties

Prototyped in file “r_vee_rx_if.h”

Description

Returns the version of this module. The version number is encoded such that the top two bytes are the major version number and the bottom two bytes are the minor version number.

Example

```
    uint32_t   version;

    version = R_VEE_GetVersion();
```

Special Notes:

This function is inlined using the “#pragma inline” directive.

4. Demo Projects

Demo projects are complete stand-alone programs. They include function main() that utilizes the module and its dependent modules (e.g. r_flash_rx). The standard naming convention for the demo project is <module>_demo_<board> where <module> is the peripheral acronym (e.g. s12ad, cmt, sci) and the <board> is the standard RSK (e.g. rskrx113). For example, s12ad FIT module demo project for RSKRX113 will be named as s12ad_demo_rskrx113. Similarly the exported .zip file will be <module>_demo_<board>.zip. For the same example, the zipped export/import file will be named as s12ad_demo_rskrx113.zip

4.1 vee_demo_rskrx231

This is a simple demo for the RSKRX231 starter kit. The demo uses the r_vee_rx API to write and read-back records for different record IDs.

Setup and Execution

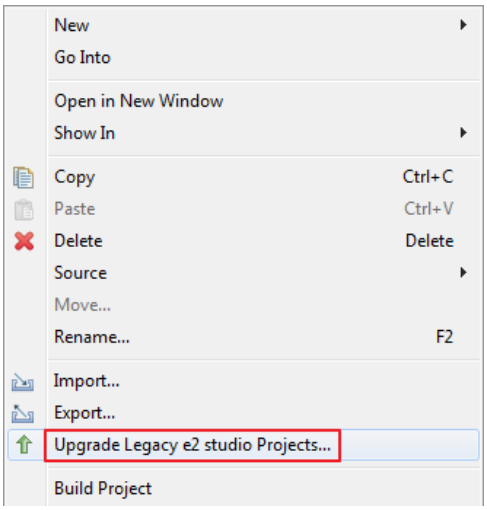
1. Compile and download the sample code.
2. Click 'Reset Go' to start the software. If the program stops at main(), press F8 to resume.

Boards Supported

RSKRX231

4.2 Adding a Demo to a Workspace

To add a demo project to a workspace, select File>Import>General>Existing Projects into Workspace, then click “Next”. From the Import Projects dialog, choose the “Select archive file” radio button and Browse to the .zip file for the demo. If you are using e2studio v6.0.0 or later, you may need to update the project after importing it in order for the project to build properly. This is done by right-clicking on the project folder and selecting “Upgrade Legacy e2studio Projects”.



Appendix: Error Modes

NOTE: All of these error return codes denote that an error was encountered while processing the previous API call, and that VEE has entered an error mode.

Once in error mode, the only functions that may be called are R_VEE_Control() with VEE_CMD_GET_STATUS and R_VEE_Close(). The functions R_VEE_GetRecordPtr() and R_VEE_GetRefDataPtr() may also be called if the option VEE_CFG_ALLOW_GETS_AFTER_FAIL is set to 1 in “r_vee_rx_config.h”.

VEE_ERR_OVERFLOW
This error can only be detected at run time. It occurs when the segment size is too small to hold at least one record for each ID defined by the user. This is a user app design issue. This error is typically resolved by reducing the number of segments defined by VEE_CFG_NUM_SEGMENTS in “r_vee_rx_config.h”. If the error still occurs when there are only 2 segments defined, an MCU with more data flash is necessary.

VEE_ERR_UNKNOWN_WRITE
This error can only occur if the VEE driver itself has been modified by the user. It indicates that a new Write State has been created by the user but no corresponding processing code was implemented. It is strongly recommended not to modify the VEE driver.

VEE_ERR_FLASH_PE_FAIL
This error indicates that a flash programming, erase, or blankcheck operation has failed in hardware. This condition can occur either due to a power loss or excessive noise during a flash operation, or due to extreme flash degradation. If the error was caused by a power loss or noise, closing VEE and re-opening it will reset the flash hardware and clear any corrupted (incomplete write or erase) item in data flash.

Website and Support

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

Inquiries
http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Nov.29.17	—	First edition issued.
1.01	Mar.1.18		Added Japanese app note.
1.10	Oct.2.18		Fixed bug where if the system rebooted with the first segment active and no reference data was present, writes would eventually fail and segment refresh would not occur.
1.11	Oct.18.18		Updated readme.txt file and demo.

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU 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 an MPU or MCU 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.

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 or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving 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, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. 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 shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
5. Renesas Electronics products are classified according to the following two quality grades: “Standard” and “High Quality”. The intended 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; industrial robots; etc.

   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.

   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or 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 damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user’s manual or other Renesas Electronics document.
6. When using Renesas Electronics products, refer to the latest product information (data sheets, user’s manuals, application notes, “General Notes for Handling and Using Semiconductor Devices” in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
7. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, 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 and impractical, you are responsible for evaluating 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. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
9. Renesas Electronics products and technologies shall 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 shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
10. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
11. This document shall not be reprinted, 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.

(Note 1) “Renesas Electronics” as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

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

(Rev.4.0-1 November 2017)

RENESAS

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics Corporation
TOYOSU FORESIA, 3-2-24 Toyosu, Koto-ku, Tokyo 135-0061, Japan

Renesas Electronics America Inc.
1001 Murphy Ranch Road, Milpitas, CA 95035, U.S.A.
Tel: +1-408-432-8888, Fax: +1-408-434-5351

Renesas Electronics Canada Limited
9251 Yonge Street, Suite 8309 Richmond Hill, Ontario Canada L4C 9T3
Tel: +1-905-237-2004

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-6503-0, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
Room 1709 Quantum Plaza, No.27 ZhichunLu, Haidian District, Beijing, 100191 P. R. China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 301, Tower A, Central Towers, 555 Langao Road, Putuo District, Shanghai, 200333 P. R. China
Tel: +86-21-2226-0888, Fax: +86-21-2226-0999

Renesas Electronics Hong Kong Limited
Unit 1601-1611, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2265-6688, Fax: +852 2886-9022

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei 10543, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre, Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 1207, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics India Pvt. Ltd.
No.777C, 100 Feet Road, HAL 2nd Stage, Indiranagar, Bangalore 560 038, India
Tel: +91-80-67208700, Fax: +91-80-67208777

Renesas Electronics Korea Co., Ltd.
17F, KAMCO Yangjae Tower, 262, Gangnam-daero, Gangnam-gu, Seoul, 06265 Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5338

Colophon 7.2