RX Family

BYTEQ Module Using Firmware Integration Technology

Introduction

This module provides functions for creating and maintaining byte-based circular buffers.

Target Device

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

- **All RX MCUs**

When using this application note with other Renesas MCUs, careful evaluation is recommended after making modifications to comply with the alternate MCU.

Target Compilers

- **Renesas Electronics C/C++ Compiler Package for RX Family**
- **GCC for Renesas RX**
- **IAR C/C++ Compiler for Renesas RX**

For details of the confirmed operation contents of each compiler, refer to “5.1 Confirmed Operation Environment".

Contents

1. Overview

The Byte Queue (BYTEQ) module provides basic circular buffer services for buffers provided by the application.

The module allocates a Queue Control Block (QCB) for each buffer passed to the Open() function. A QCB maintains the buffer's "in" and "out" indexes for adding and removing data from the queue. The Queue Control Blocks can be allocated statically at compile time or dynamically at run time (using malloc). An equate in config.h determines whether they are statically or dynamically created. If they are statically allocated, an additional equate is utilized which specifies the maximum number of buffers to be supported.

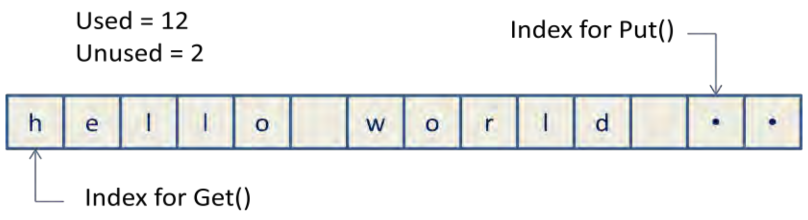
There is one control block per buffer. When an R_BYTEQ_Open() is performed, a pointer to the application's buffer and its length are passed in, and a pointer to a QCB is provided. This pointer, which is called a Handle, is then passed to all of the other API functions. The functions then operate on the queue referenced by this Handle. Because there is no global or static data shared between the queues, the API functions are re-entrant for different queues.

This module does not make use of any interrupts. If a queue can be modified at both the interrupt and application level, it is up to the application to ensure that the appropriate related interrupt is disabled whenever the queue is being accessed. Similarly, if the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

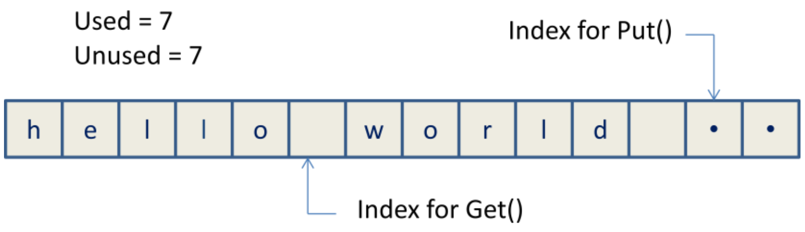
1.1 Using the BYTEQ Module

The following illustrates a queue's behavior with API calls:

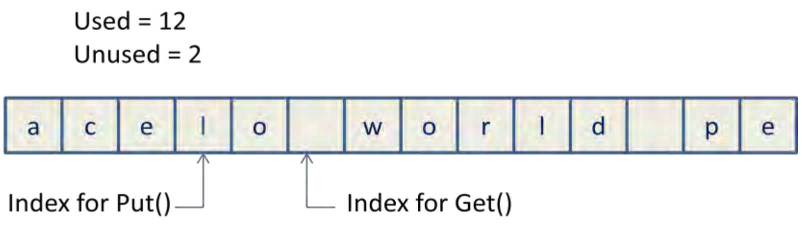
```
    #define BUFSIZE 14

    uint8_t       my_buf[BUFSIZE];
    byteq_hdl_t   my_que;
    byteq_err_t   err;
    uint8_t       byte;

    err = R_BYTEQ_Open(my_buf, BUFSIZE, &my_que);

    // add 12 bytes to queue
    R_BYTEQ_Put(my_que, 'h');
    R_BYTEQ_Put(my_que, 'e');
    R_BYTEQ_Put(my_que, 'l');
    R_BYTEQ_Put(my_que, 'l');
    R_BYTEQ_Put(my_que, 'o');
    R_BYTEQ_Put(my_que, ' ');
    R_BYTEQ_Put(my_que, 'w');
    R_BYTEQ_Put(my_que, 'o');
    R_BYTEQ_Put(my_que, 'r');
    R_BYTEQ_Put(my_que, 'l');
    R_BYTEQ_Put(my_que, 'd');
    R_BYTEQ_Put(my_que, ' ');
```



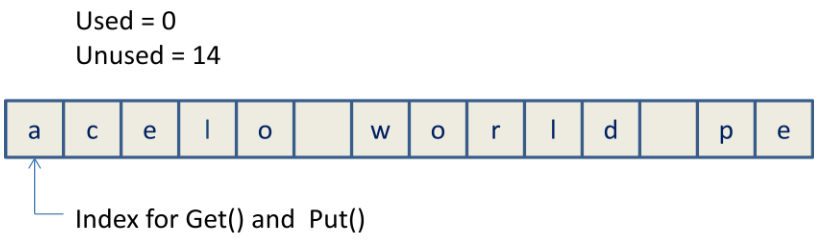
```
// remove 5 bytes from queue
R_BYTEQ_Get(my_que, &byte);      // byte = 'h'
R_BYTEQ_Get(my_que, &byte);      // byte = 'e'
R_BYTEQ_Get(my_que, &byte);      // byte = 'l'
R_BYTEQ_Get(my_que, &byte);      // byte = 'l'
R_BYTEQ_Get(my_que, &byte);      // byte = 'o'
```



```
// add 5 bytes to queue
R_BYTEQ_Put(my_que, 'p');
R_BYTEQ_Put(my_que, 'e');
R_BYTEQ_Put(my_que, 'a');
R_BYTEQ_Put(my_que, 'c');
R_BYTEQ_Put(my_que, 'e');
```



```
// flush queue
R_BYTEQ_Flush(my_que);
```



2. API Information

This Driver API follows the Renesas API naming standards.

2.1 Hardware Requirements

No hardware requirements.

2.2 Software Requirements

This driver is dependent upon the following FIT packages:

- Renesas Board Support Package (r_bsp) v3.10or higher

2.3 Limitations

No software limitations.

2.4 Supported Toolchain

This driver has been confirmed to work with the toolchain listed in 5.1, Confirmed Operation Environment.

2.5 Header Files

Compile time configurable options are located in r_byteq\ref\r_byteq_config_reference.h. This file should be copied into the r_config subdirectory of the project and renamed to r_byteq_config.h. It is this renamed file that should be modified if needed and the original kept as a reference.

All API calls and their supporting interface definitions are located in r_byteq\r_byteq_if.h. Both this file and r_byteq_config.h should be included by the User's application.

2.6 Integer Types

If your toolchain supports C99 then *stdint.h* should be described as shown below. If not, then there should be *typedefs.h* file that is included with your project as defined by the Renesas Coding Standards document.

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

All configurable options that can be set at build time are located in the file "r_byteq_config.h". A summary of these settings are provided in the following table:

Configuration options in *r_byteq_config.h*	
`#define BYTEQ_CFG_PARAM_CHECKING_ENABLE`	= 1: Include parameter checking in the build. = 0: Omit parameter checking from the build. = BSP_CFG_PARAM_CHECKING_ENABLE (default): Use the system default setting. Note: Code size can be reduced by excluding parameter checking from the build.
`#define BYTEQ_CFG_USE_HEAP_FOR_CTRL_BLKS 0`	A control block is needed for each queue to maintain in/out indexes. By default, these control blocks are allocated at compile time. To dynamically allocate memory at run time, set this equate to 1.
`#define BYTEQ_CFG_MAX_CTRL_BLKS 32`	Specifies how many control blocks to allocate at compile time. This constant is ignored if BYTEQ_CFG_USE_HEAP_FOR_CTRL_BLKS is 1.

2.8 Code Size

The sizes of ROM and RAM of this module are listed below.

The ROM (code and constants) and RAM (global data) sizes are determined by the build-time configuration options described in 2.7, Configuration Overview.

The values in the table below are confirmed under the following conditions.

Module Revision: r_byteq rev1.80

Compiler Version: Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00

(The option of "lang = c99" is added to the default settings of the integrated development environment.)

GCC for Renesas RX 4.8.4.201801

(The option of "lang = c99" is added to the default settings of the integrated development environment.)

IAR C/C++ Compiler for Renesas RX version 4.10.1

(The default settings of the integrated development environment.)

Configuration Options: Default settings

ROM, RAM and Stack Code Sizes							
Device	Category	Memory Used					
		Renesas Compiler		GCC		IAR Compiler	
		With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking	With Parameter Checking	Without Parameter Checking
Using Heap for Control Blocks	ROM	257 bytes	170 bytes	1056 bytes	928 bytes	592 bytes	508 bytes
	RAM	12 bytes for malloc() x Control Blocks		12 bytes for malloc() x Control Blocks		12 bytes for malloc() x Control Blocks	
Using Allocated Control Blocks	ROM	300 bytes	210 bytes	624 bytes	448 bytes	272 bytes	196 bytes
	RAM	1 byte +12 bytes x BYTEQ_CFG_MAX_CTRL_BLKS		12 bytes x BYTEQ_CFG_MAX_CTRL_BLKS		12 bytes x BYTEQ_CFG_MAX_CTRL_BLKS	

2.9 Adding the Module to Your Project

This module must be added to each project in which it is used. Renesas recommends the method using the Smart Configurator described in (1) or (3) below. However, the Smart Configurator only supports some RX devices. Please use the methods of (2) or (4) for RX devices that are not supported by the Smart Configurator.

(1) Adding the FIT module to your project using the Smart Configurator in e^2 studio
By using the Smart Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(2) Adding the FIT module to your project using the FIT Configurator in e^2 studio
By using the FIT Configurator in e^2 studio, the FIT module is automatically added to your project. Refer to “Adding Firmware Integration Technology Modules to Projects (R01AN1723)” for details.

(3) Adding the FIT module to your project using the Smart Configurator in CS+
By using the Smart Configurator Standalone version in CS+, the FIT module is automatically added to your project. Refer to “Renesas e^2 studio Smart Configurator User Guide (R20AN0451)” for details.

(4) Adding the FIT module to your project in CS+
In CS+, please manually add the FIT module to your project. Refer to “Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)” for details.

2.10 “for”, “while” and “do while” statements

In this module, “for”, “while” and “do while” statements (loop processing) are used in processing to wait for register to be reflected and so on. For these loop processing, comments with “WAIT_LOOP” as a keyword are described. Therefore, if user incorporates fail-safe processing into loop processing, user can search the corresponding processing with “WAIT_LOOP”.

The following shows example of description.

```
while statement example :
/* WAIT_LOOP */
while(0 == SYSTEM.OSCOVFSR.BIT.PLOVF)
{
    /* The delay period needed is to make sure that the PLL has stabilized. */
}

for statement example :
/* Initialize reference counters to 0. */
/* WAIT_LOOP */
for (i = 0; i < BSP_REG_PROTECT_TOTAL_ITEMS; i++)
{
    g_protect_counters[i] = 0;
}

do while statement example :
/* Reset completion waiting */
do
{
    reg = phy_read(ether_channel, PHY_REG_CONTROL);
    count++;
} while ((reg & PHY_CONTROL_RESET) && (count < ETHER_CFG_PHY_DELAY_RESET)); /* WAIT_LOOP */
```

3. API Functions

3.1 Summary

The following functions are included in this design:

Function	Description
R_BYTEQ_Open()	Allocates and initializes a queue control block for a buffer provided by the user. Provides a queue handle for use with other API functions.
R_BYTEQ_Close()	Releases the queue control block associated with the handle.
R_BYTEQ_Put()	Adds a byte of data to the queue.
R_BYTEQ_Get()	Removes the oldest byte of data from the queue.
R_BYTEQ_Flush()	Resets the queue to an empty state.
R_BYTEQ_Used()	Provides the number of bytes used in the queue.
R_BYTEQ_Unused()	Provides the number of bytes unused in the queue.
R_BYTEQ_GetVersion()	Returns at runtime the module version number.

3.2 Return Values

These are the different error codes API functions can return. The enum is found in r_byteq_if.h along with

the API function declarations.

```
typedef enum _byteq_err          // BYTEQ API error codes
{
    BYTEQ_SUCCESS = 0,
    BYTEQ_ERR_NULL_PTR,          // received null ptr; missing required arg
    BYTEQ_ERR_INVALID_ARG,       // argument is not valid for parameter
    BYTEQ_ERR_MALLOC_FAIL,       // cannot allocate mem for ctrl block;
                                 // increase heap
    BYTEQ_ERR_NO_MORE_CTRL_BLKS, // no more ctrl blocks;
                                 // increase BYTEQ_MAX_CTRL_BLKS
    BYTEQ_ERR_QUEUE_FULL,        // queue full; cannot add another byte
    BYTEQ_ERR_QUEUE_EMPTY        // queue empty; no byte to fetch
} byteq_err_t;
```

3.3 R_BYTEQ_Open()

This function allocates and initializes a queue control block for a buffer provided by the user. A queue handle is provided for use with other API functions.

Format

```
byteq_err_t R_BYTEQ_Open(uint8_t * const          p_buf,
                         uint16_t const            size,
                         byteq_hdl_t * const   p_hdl)
```

Parameters

p_buf
  Pointer to byte buffer.
size
  Buffer size in bytes.
p_hdl
  Pointer to a handle for queue (value set here)

Return Values

BYTEQ_SUCCESS:	*Successful; queue initialized*
BYTEQ_ERR_NULL_PTR:	*p_buf is NULL*
BYTEQ_ERR_INVALID_ARG:	*Size is less than or equal to 1.*
BYTEQ_ERR_MALLOC_FAIL:	*Cannot allocate control block. Increase heap size.*
BYTEQ_ERR_NO_MORE_CTRL_BLKS:	*Cannot assign control block.* *Increase BYTEQ_MAX_CTRL_BLKS in config.h.*

Properties

Prototyped in file “r_byteq_if.h”

Description

This function allocates or assigns a queue control block for the buffer pointed to by *p_buf*. Initializes the queue to an empty state and provides a Handle to its control structure in *p_hdl* which is then used as a queue ID for the other API functions.

Reentrant

Function is re-entrant for different buffers.

Example

```
    #define BUFSIZE 80

    uint8_t       tx_buf[BUFSIZE];
    byteq_hdl_t   tx_que;
    byteq_err_t   byteq_err;

    byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);
```

Special Notes:

None.

3.4 R_BYTEQ_Close()

This function releases the queue control block associated with a handle.

Format

byteq_err_t R_BYTEQ_Close(byteq_hdl_t const   hdl)

Parameters

hdl

Handle for queue.

Return Values

BYTEQ_SUCCESS:	*Successful; control block released.*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

If the control block associated with this Handle was allocated dynamically at run time (BYTEQ_USE_HEAP_FOR_CTRL_BLKS set to 1 in config.h), then that memory is free()d by this function. If the control block was statically allocated at compile time (BYTEQ_USE_HEAP_FOR_CTRL_BLKS set to 0 in config.h), then this function marks the control block as available for use by another buffer. Nothing is done to the contents of the buffer referenced by this Handle.

Reentrant

Function is re-entrant for different queues.

Example

```
    byteq_hdl_t   tx_que;
    byteq_err_t   byteq_err;

    byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);

    byteq_err = R_BYTEQ_Close(tx_que);
```

Special Notes:

None.

3.5 R_BYTEQ_Put()

This function adds a byte of data to the queue.

Format

```
byteq_err_t R_BYTEQ_Put(byteq_hdl_t const   hdl,
                        uint8_t const        byte)
```

Parameters

hdl
Handle for queue.

byte
Byte to add to queue.

Return Values

BYTEQ_SUCCESS:	*Successful; byte added to queue*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*
BYTEQ_ERR_QUEUE_FULL	*Queue full; cannot add byte to queue.*

Properties

Prototyped in file "r_byteq_if.h"

Description

This function adds the contents of *byte* to the queue associated with *hdl*.

Reentrant

Function is re-entrant for different queues.

Example

```
     byteq_hdl_t tx_que;
    byteq_err_t byteq_err;
    uint8_t     byte = 'A';

    byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);
    byteq_err = R_BYTEQ_Put(tx_que, byte);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. If the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

3.6 R_BYTEQ_Get()

This function removes a byte of data from the queue.

Format

```
byteq_err_t R_BYTEQ_Get(byteq_hdl_t const   hdl,
                        uint8_t * const      p_byte)
```

Parameters

hdl
Handle for queue.

p_byte
Pointer to load byte to.

Return Values

BYTEQ_SUCCESS:	*Successful; byte removed from queue*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*
BYTEQ_ERR_INVALID_ARG:	*p_byte is NULL.*
BYTEQ_ERR_QUEUE_EMPTY:	*Queue empty; no data available to fetch*

Properties

Prototyped in file “r_byteq_if.h”

Description

This function removes the oldest byte of data in the queue associated with *hdl* and loads it into the location

pointed to by *p_byte*.

Reentrant

Function is re-entrant for different queues.

Example

```
 byteq_hdl_t rx_que;
byteq_err_t byteq_err;
uint8_t     byte;

byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

/* queue filled with data by R_BYTEQ_Put()elsewhere */

byteq_err = R_BYTEQ_Get(rx_que, &byte);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. If the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

3.7 R_BYTEQ_Flush()

This function resets a queue to an empty state.

Format

byteq_err_t R_BYTEQ_Flush(byteq_hdl_t const   hdl)

Parameters

hdl

Handle for queue.

Return Values

BYTEQ_SUCCESS:	*Successful; queue reset*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

This function resets the queue identified by *hdl* to an empty state.

Reentrant

Function is re-entrant for different queues.

Example

```
    byteq_hdl_t rx_que;
    byteq_err_t byteq_err;


    byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

    /* queue filled with data by R_BYTEQ_Put()elsewhere */

    byteq_err = R_BYTEQ_Flush(rx_que);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. If the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

3.8 R_BYTEQ_Used()

This function provides the number of data bytes in the queue.

Format

```
byteq_err_t R_BYTEQ_Used(byteq_hdl_t const   hdl,
                         uint16_t * const     p_cnt)
```

Parameter

hdl

Handle for queue.

p_cnt

Pointer to load queue data count to.

Return Values

BYTEQ_SUCCESS:	*Successful; *p_cnt loaded with the number of bytes in the queue*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*
BYTEQ_ERR_INVALID_ARG:	*p_cnt is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

This function loads the number of bytes in the queue associated with *hdl* and into the location pointed to by

p_cnt.

Reentrant

Yes.

Example

```
    byteq_hdl_t rx_que;
    byteq_err_t byteq_err;
    uint16_t    count;

    byteq_err = R_BYTEQ_Open(rx_buf, BUFSIZE, &rx_que);

    /* queue filled with data by R_BYTEQ_Put()elsewhere */

    byteq_err = R_BYTEQ_Used(rx_que, &count);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. If the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

3.9 R_BYTEQ_Unused()

This function provides the number of data bytes available for storage in the queue.

Format

```
byteq_err_t R_BYTEQ_Unused(byteq_hdl_t const   hdl,
                           uint16_t * const     p_cnt)
```

Parameters

hdl
  Handle for queue.

p_cnt
  Pointer to load queue unused byte count to.

Return Values

BYTEQ_SUCCESS:	*Successful; *p_cnt loaded with the number of bytes not used in the queue*
BYTEQ_ERR_NULL_PTR:	*hdl is NULL.*
BYTEQ_ERR_INVALID_ARG:	*p_cnt is NULL.*

Properties

Prototyped in file “r_byteq_if.h”

Description

This function loads the number of unused bytes in the queue associated with *hdl* and into the location

pointed to by *p_cnt*.

Reentrant

Yes.

Example

```
    byteq_hdl_t tx_que;
    byteq_err_t byteq_err;
    uint16_t    count;

    byteq_err = R_BYTEQ_Open(tx_buf, BUFSIZE, &tx_que);

    /* queue filled with data by R_BYTEQ_Put()elsewhere */

    byteq_err = R_BYTEQ_Unused(tx_que, &count);
```

Special Notes:

If the queue is accessed at both the interrupt and application level, it is up to the user to disable and enable the associated interrupt before and after calling this function from the application level. If the queue is accessed by tasks of different priorities, it is up to the user to prevent task switching or to utilize a mutex or semaphore to reserve the queue.

3.10 R_BYTEQ_GetVersion()

This function returns the driver version number at runtime.

Format

uint32_t R_BYTEQ_GetVersion(void)

Parameters

None

Return Values

Version number.

Properties

Prototyped in file “r_byteq_if.h”

Description

Returns the version of this module. The version number is encoded such that the top 2 bytes are the major version number and the bottom 2 bytes are the minor version number.

Reentrant

Yes

Example

```
uint32_t   version;

version = R_BYTEQ_GetVersion();
```

Special Notes:

None.

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_bsp). This FIT module has the following demo projects.

4.1 byteq_demo_rskrx231

The byteq_demo_rskrx71m project demonstrates how to use some of the BYTEQ API calls. The demo project opens and initializes a queue, puts characters into the queue, queries the number of characters in the queue, gets the characters from the queue and prints them to the virtual console. The demo project also prints out the version number of the BYTEQ module. Virtual console can be enabled by selecting Open Console > Renesas Debug Virtual Console.

4.2 byteq_demo_rskrx71m

The byteq_demo_rskrx71m project demonstrates how to use some of the BYTEQ API calls. The demo project opens and initializes a queue, puts characters into the queue, queries the number of characters in the queue, gets the characters from the queue and prints them to the virtual console. The demo project also prints out the version number of the BYTEQ module. Virtual console can be enabled by selecting Open Console > Renesas Debug Virtual Console.

4.3 Adding a Demo to a Workspace

Demo projects are found in the FITDemos subdirectory of the distribution file for this application note. 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. “Browse” to the FITDemos subdirectory, select the desired demo zip file, then click “Finish”.

4.4 Downloading Demo Projects

Demo projects are not included in the RX Driver Package. When using the demo project, the FIT module needs to be downloaded. To download the FIT module, right click on this application note and select “Sample Code (download)” from the context menu in the *Smart Brower* >> *Application Notes* tab.

5. Appendices

5.1 Confirmed Operation Environment

This section describes confirmed operation environment for the r_byteq FIT module.

Table 5.1 Confirmed Operation Environment (Rev. 1.60)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 4.2.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.04.01
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.60
Board used	Renesas Starter Kit+ for RX65N (型名.: RTK500565NSxxxxxBE)

Table 5.2 Confirmed Operation Environment (Rev. 1.70)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.0.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V2.08.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.70
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231SxxxBE) Renesas Starter Kit+ for RX71M (product No.: R0K50571MSxxxBE)

Table 5.3 Confirmed Operation Environment (Rev. 1.71)

Item	Contents
Integrated development environment	Renesas Electronics e^2 studio Version 7.1.0
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.00.00
	Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
Endian	Big endian/little endian
Revision of the module	Rev.1.71

Table 5.4 Confirmed Operation Environment (Rev. 1.80)

Item	**Contents**
Integrated development environment	Renesas Electronics e^2 studio Version 7.3.0 IAR Embedded Workbench for Renesas RX 4.10.1
C compiler	Renesas Electronics C/C++ Compiler Package for RX Family V3.01.00 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	GCC for Renesas RX 4.8.4.201801 Compiler option: The following option is added to the default settings of the integrated development environment. -lang = c99
	IAR C/C++ Compiler for Renesas RX version 4.10.1 Compiler option: The default settings of the integrated development environment.
Endian	Big endian/little endian
Revision of the module	Rev.1.80
Board used	Renesas Starter Kit for RX231 (product No.: R0K505231xxxxxx)

5.2 Troubleshooting

(1) Q: I have added the FIT module to the project and built it. Then I got the error: Could not open source file "platform.h".

A: The FIT module may not be added to the project properly. Check if the method for adding FIT modules is correct with the following documents:

- Using CS+:

  Application note "Adding Firmware Integration Technology Modules to CS+ Projects (R01AN1826)"

- Using e^2 studio:

  Application note "Adding Firmware Integration Technology Modules to Projects (R01AN1723)"

When using this FIT module, the board support package FIT module (BSP module) must also be added to the project. Refer to the application note "Board Support Package Module Using Firmware Integration Technology (R01AN1685)".

6. Reference Documents

User’s Manual: Hardware
The latest versions can be downloaded from the Renesas Electronics website.

Technical Update/Technical News
The latest information can be downloaded from the Renesas Electronics website.

User’s Manual: Development Tools
RX Family C/C++ Compiler CC-RX User's Manual (R20UT3248)
The latest version can be downloaded from the Renesas Electronics website.

Related Technical Updates

This module reflects the content of the following technical updates.
None

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Jul.24.13	—	First edition issued
1.10	Jul.21.14	—	Updated XML file for new supported MCUs.
1.20	Nov.21.14	—	Removed dependency to BSP. Updated XML file for new supported MCUs.
1.30	Jan.22.15	—	Updated XML file for new supported MCUs.
1.40	Jun.30.15	—	Added support for the RX231 Group.
1.50	Sep.30.15	—	Added support for the RX23T Group.
		5	Added r_bsp in Section 2.2 Software Requirements.
		6	Updated code sizes in 2.8 Code Size.
1.60	Jan.29.16	6	Updated code sizes in 2.8 Code Size.
		18	Added the section of 4. Demo Projects.
		program	Changed the XML in order not to depend on the series / group / board of the RX Family. Fixed the initial setting procedure in the R_BYTEQ_Open function. Fixed a program according to the Renesas coding rules.
1.70	Jun.01.18	—	Added support setting function of configuration option Using GUI on Smart Configurator.
		—	Updated Demo projects.
		5	Changed toolchain in 2.4 Supported Toolchain.
		6	Changed the default value of BYTEQ_CFG_MAX_CTRL_BLKS in 2.7 Configuration Overview. Updated code sizes in 2.8 Code Size.
		7	Added the section of 2.9 Adding the Module to Your Project.
		8	Added the section of 2.10 “for”, “while” and “do while” statements.
		18	Added the section of 4.4 Downloading Demo Projects.
		19	Added the section of 5. Appendices. Added the section of 5.1 Confirmed Operation Environment.
		20	Added the section of 5.2 Troubleshooting.
		21	Added the section of 6. Reference Documents.
		program	Changed the default value of the following macro definition: - BYTEQ_CFG_MAX_CTRL_BLKS: Value: (4) -> (32)
1.71	Dec.03.18	19	5.1 Operation Confirmation Environment: Added Table 5.3 Confirmed Operation Environment (Rev. 1.71).
		program	Added document number of the application note accompanying the sample program of the FIT module to xml file.
1.80	Feb.07.19	—	Supported the following compilers: - GCC for Renesas RX - IAR C/C++ Compiler for Renesas RX
		1	Added the section of Target compiler. Deleted related documents.
		7	Updated the section of 2.8 Code Size.
		18	Updated the section of 3.10 R_BYTEQ_GetVersion().
		21	Updated the section of 5.1 Confirmed Operation Environment.
		23	Deleted the section of Website and Support.
		program	Deleted the inline expansion of the R_BYTEQ_GetVersion function.

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit:
www.renesas.com/contact/.

Trademarks