

## **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.....	3
1.1 Using the BYTEQ Module .....	3
2. API Information.....	5
2.1 Hardware Requirements .....	5
2.2 Software Requirements .....	5
2.3 Limitations .....	5
2.4 Supported Toolchain .....	5
2.5 Header Files .....	5
2.6 Integer Types .....	5
2.7 Configuration Overview .....	6
2.8 Code Size .....	7
2.9 Adding the Module to Your Project .....	8
2.10 “for”, “while” and “do while” statements .....	9
3. API Functions .....	10
3.1 Summary .....	10
3.2 Return Values.....	10
3.3 R_BYTEQ_Open() .....	11
3.4 R_BYTEQ_Close() .....	12
3.5 R_BYTEQ_Put() .....	13
3.6 R_BYTEQ_Get() .....	14
3.7 R_BYTEQ_Flush() .....	15
3.8 R_BYTEQ_Used() .....	16
3.9 R_BYTEQ_Unused() .....	17
3.10 R_BYTEQ_GetVersion() .....	18
4. Demo Projects .....	19
4.1 byteq_demo_rskrx231 .....	19
4.2 byteq_demo_rskrx71m .....	19
4.3 Adding a Demo to a Workspace .....	19
4.4 Downloading Demo Projects .....	19
5. Appendices.....	20
5.1 Confirmed Operation Environment.....	20
5.2 Troubleshooting.....	22
6. Reference Documents.....	23
Related Technical Updates .....	23
Revision Record.....	24

## 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;

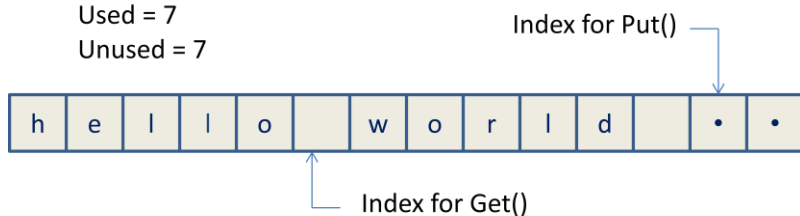
err = R_BYTEQ_Open(my_buf, BUFSIZE, &my_que);

// 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, ' ');
```



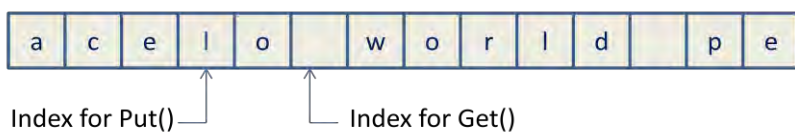
```
// remove 5 bytes from queue
R_BYTEQ_Get(my_que, &byte);    // byte = 'h'
R_BYTEQ_Get(my_que, &byte);    // byte = 'e'
R_BYTEQ_Get(my_que, &byte);    // byte = 'l'
R_BYTEQ_Get(my_que, &byte);    // byte = 'l'
R_BYTEQ_Get(my_que, &byte);    // byte = 'o'
```

Used = 7  
Unused = 7



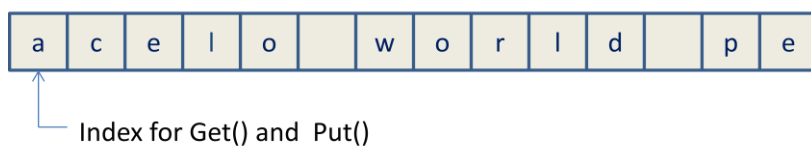
```
// add 5 bytes to queue
R_BYTEQ_Put(my_que, 'p');
R_BYTEQ_Put(my_que, 'e');
R_BYTEQ_Put(my_que, 'a');
R_BYTEQ_Put(my_que, 'c');
R_BYTEQ_Put(my_que, 'e');
```

Used = 12  
Unused = 2



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

Used = 0  
Unused = 14



## 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) v3.10 or 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 <i>r_byteq_config.h</i>	
<pre>#define BYTEQ_CFG_PARAM_CHECKING_ENABLE</pre>	= 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.
<pre>#define BYTEQ_CFG_USE_HEAP_FOR_CTRL_BLKs 0</pre>	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.
<pre>#define BYTEQ_CFG_MAX_CTRL_BLKs</pre> <div>32</div>	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_BLS		12 bytes x BYTEQ_CFG_MAX_CTRL_BLS		12 bytes x BYTEQ_CFG_MAX_CTRL_BLS	

---

## 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<sup>2</sup> studio  
By using the Smart Configurator in e<sup>2</sup> studio, the FIT module is automatically added to your project. Refer to “Renesas e<sup>2</sup> studio Smart Configurator User Guide (R20AN0451)” for details.
- (2) Adding the FIT module to your project using the FIT Configurator in e<sup>2</sup> studio  
By using the FIT Configurator in e<sup>2</sup> 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<sup>2</sup> 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

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

#### 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 <sup>2</sup> 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.: RTK500565NSxxxxBE)

**Table 5.2 Confirmed Operation Environment (Rev. 1.70)**

Item	Contents
Integrated development environment	Renesas Electronics e <sup>2</sup> 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 <sup>2</sup> 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 <sup>2</sup> 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<sup>2</sup> 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.



# 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. Precaution against Electrostatic Discharge (ESD)

A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity.

Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

## 2. Processing at power-on

The state of the product is undefined at the time 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 time 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 time 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 time when power is supplied until the power reaches the level at which resetting is specified.

## 3. Input of signal during power-off state

Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

## 4. 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 the 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.

## 5. Clock signals

After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is 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. Additionally, 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.

## 6. Voltage application waveform at input pin

Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between  $V_{IL}$  (Max.) and  $V_{IH}$  (Min.).

## 7. Prohibition of access to reserved addresses

Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

## 8. Differences between products

Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems.

The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of 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.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.

(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.4.0-1 November 2017)

## Corporate Headquarters

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

[www.renesas.com](http://www.renesas.com)

## Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

## 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/](http://www.renesas.com/contact/).