

Common RFM2g Application Program Interface (API) and Command Line Interpreter for VMISFT/RFM2G Drivers

Reference Manual



A GE Fanuc Company

12090 South Memorial Parkway
Huntsville, Alabama 35803-3308, USA
(256) 880-0444 ♦ (800) 322-3616 ♦ Fax: (256) 882-0859

523-000447-000 Rev. A

COPYRIGHT AND TRADEMARKS

© Copyright February 2003. The information in this document has been carefully checked and is believed to be entirely reliable. While all reasonable efforts to ensure accuracy have been taken in the preparation of this manual, VMIC assumes no responsibility resulting from omissions or errors in this manual, or from the use of information contained herein.

VMIC reserves the right to make any changes, without notice, to this or any of VMIC's products to improve reliability, performance, function, or design.

VMIC does not assume any liability arising out of the application or use of any product or circuit described herein; nor does VMIC convey any license under its patent rights or the rights of others.

For warranty and repair policies, refer to VMIC's Standard Conditions of Sale.

AMXbus, BITMODULE, COSMODULE, DMAbus, IOMax, IOWorks Foundation, IOWorks Manager, IOWorks Server, MAGICWARE, MEGAMODULE, PLC ACCELERATOR (ACCELERATION), Quick Link, RTnet, Soft Logic Link, SRTbus, TESTCAL, "The Next Generation PLC", The PLC Connection, TURBOMODULE, UCLIO, UIOD, UPLC, Visual Soft Logic Control(ler), **VMEaccess**, VMEbus Access, **VMEmanager**, **VMEmonitor**, VMEnet, VMEnet II, and **VMEprobe** are trademarks and The I/O Experts, The I/O Systems Experts, The Soft Logic Experts, and The Total Solutions Provider are service marks of VMIC.



(I/O man figure)



(IOWorks man figure)



The I/O man figure, IOWorks, IOWorks man figure, UIOC, Visual IOWorks, and the VMIC logo are registered trademarks of VMIC.

ActiveX, Microsoft, Microsoft Access, MS-DOS, Visual Basic, Visual C++, Win32, Windows, Windows NT, and XENIX are registered trademarks of Microsoft Corporation.

MMX is a trademark, and Celeron, Intel, and Pentium are registered trademarks of Intel Corporation.

PICMG and CompactPCI are registered trademarks of PCI Industrial Computer Manufacturers' Group.

Other registered trademarks are the property of their respective owners.

VMIC

All Rights Reserved

This document shall not be duplicated, nor its contents used for any purpose, unless granted express written permission from VMIC.

Table of Contents

List of Tables	7
Overview	9
Overview	10
Accessing Additional Information	11
VMIC Documentation	11
VMIC Technical Support	12
Chapter 1 - Application Program Interface (API) Library	13
Using the Application Program Interface	15
Opening the RFM2g Driver	15
Routine Code for Use with API Function Examples	15
RFM2g Error Codes	17
RFM2g API Functions	19
RFM2g Opening and Closing API Functions	22
RFM2gOpen()	23
RFM2gClose()	25
RFM2g Configuration API Functions	26
RFM2gGetConfig()	27
RFM2gUserMemory()	28
RFM2gUnMapUserMemory()	30
RFM2gNodeID()	32
RFM2gBoardID()	33
RFM2gSize()	34
RFM2gFirst()	35
RFM2gDeviceName()	36
RFM2gDIIVersion()	37
RFM2gDriverVersion()	38

RFM2gGetDMAThreshold()	39
RFM2gSetDMAThreshold()	41
RFM2gGetDMAByteSwap()	43
RFM2gSetDMAByteSwap()	44
RFM2gGetPIOByteSwap()	45
RFM2gSetPIOByteSwap()	46
RFM2g Data Transfer API Functions	48
Data Transfer Considerations	48
Big Endian and Little Endian Data Conversions	49
Using Direct Memory Access (DMA)	49
RFM2gRead()	50
RFM2gWrite()	53
RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()	55
RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()	58
RFM2g Interrupt Event API Functions	60
RFM2gEnableEvent()	61
RFM2gDisableEvent()	63
RFM2gSendEvent()	65
RFM2gWaitForEvent()	67
RFM2gEnableEventCallback()	70
RFM2gDisableEventCallback()	73
RFM2gClearEvent()	75
RFM2gCancelWaitForEvent()	77
RFM2g Utility API Functions	79
RFM2gErrorMsg()	80
RFM2gGetLed()	81
RFM2gSetLed()	82
RFM2gCheckRingCont()	83
RFM2gGetDebugFlags()	84
RFM2gSetDebugFlags()	87
Chapter 2 - rfm2g_util.c Utility Program	91
RFM2g Command Line Interpreter	92
Using the Command Line Interpreter	92
Notes On Entering Numbers	93
Notes On Device Numbers	93
Command Line Interpreter Example	94

Utility Commands.....	95
boardid.....	98
cancelwait.....	99
checkring.....	100
clearevent.....	101
config.....	102
devname.....	103
disableevent.....	104
disablecallback.....	105
dllversion.....	106
driverversion.....	107
drvspecific.....	108
dump.....	109
enableevent.....	110
enablecallback.....	111
errormsg.....	112
exit.....	113
first.....	114
getdebug.....	115
getdmabyteswap.....	116
getled.....	117
getpiobyteswap.....	118
getthreshold.....	119
help.....	120
mapuser.....	122
memop.....	123
nodeid.....	125
peek8, peek16, peek32 and peek64.....	126
performancetest.....	127
poke8, poke16, poke32 and poke64.....	128
quit.....	129
read.....	130
repeat.....	132
return.....	133
send.....	134
setdebug.....	135
setdmabyteswap.....	137
setled.....	138

setpibyteswap	139
setthreshold	140
size	141
unmapuser	142
wait	143
write	144
Troubleshooting the rfm2g_util.c Command Line Interpreter	146
Errors	146

Chapter 3 - RFM2g Sample Applications147

rfm2g_sender.c	148
rfm2g_receiver.c	149
rfm2g_map.c	150
rfm2g_sender.c and rfm2g_receiver.c Example Workflow	151
rfm2g_map.c Example Workflow	153

List of Tables

- Table 1-1** Common RFM2g Error Codes. 17
- Table 1-2** RFM2g API Functions. 19
- Table 1-3** RFM2g Opening and Closing API Functions. 22
- Table 1-4** RFM2g Configuration API Functions 26
- Table 1-5** RFM2g Data Transfer API Functions. 48
- Table 1-6** RFM2g Interrupt Event API Functions. 60
- Table 1-7** RFM2g Utility API Functions. 79

- Table 2-1** RFM2g Driver Commands 95

Overview

Contents

Overview	10
Accessing Additional Information	11
VMIC Technical Support	12

Introduction

This manual provides information on the common components included in the RFM2g drivers, which enable you to access the features of a variety of VMIPCI and VMIPMC RFM2g hardware.

Overview

The RFM2g driver provides all of the necessary files, scripts and programs for you to install, test and use any of the supported Reflective Memory (RFM) Interface cards in your system.

The RFM2g driver provides the following common features:

- *Application Program Interface (API) Library* – Application programs may use the services provided by the RFM2g Application Program Interface (API) library to access the features of the RFM2g devices in a portable way. Using the API library makes it easy to use a different model of RFM interface, or to rehost your application program on a different supported host platform. See "Application Program Interface (API) Library" on page 13 for more information.
- *Command Line Interpreter* – The **rfm2g_util.c** program is a command line interpreter that enables a user to exercise various RFM2g API commands by entering commands at the keyboard. To use **rfm2g_util.c**, follow the directions in your driver-specific manual. You can enter **help** to display a list of commands. See "rfm2g_util.c Utility Program" on page 91 for more information.
- *Example Programs* – The RFM2g driver contains the **rfm2g_sender.c**, **rfm2g_receiver.c** and **rfm2g_map.c** sample programs, which provide examples on how to use the driver and API with your application. See "RFM2g Sample Applications" on page 147 for more information.

Accessing Additional Information

VMIC Documentation

NOTE: For a list of the files distributed with your RFM2g driver, see your driver-specific manual.

The following is a list of reference documentation related to RFM2g drivers:

- *VMIPCI-5565 Ultra High-Speed Fiber-Optic Reflective Memory with Interrupts Product Manual (500-855565-000)*
- *VMIPMC-5565 Ultra High-Speed Fiber-Optic Reflective Memory with Interrupts Product Manual (500-755565-000)*

Please call your VMIC sales representative for more information.

VMIC Technical Support

You may contact VMIC's customer service at:

TELEPHONE: 1-800-269-4714
256-880-0444 (outside of U.S.)

FAX: 256-650-7245

E-MAIL: software.cs@vmic.com

Service is free for 30 days after product delivery. After this time however, you must purchase VMIC's Maintenance Agreement for continued support. For more information, refer to the Maintenance Agreement documentation that was delivered with the product.

With your correspondence, please provide the following:

- Product number, version and serial number.
- Type of target hardware, processor and board
- Exact wording of any messages on your screen
- What you were doing when the error occurred
- What steps you have taken (if any) to resolve the problem

In addition, when e-mailing, please include the following:

- Your name
- Your company's name
- Your phone and fax numbers
- Your email address

Application Program Interface (API) Library

Contents

Using the Application Program Interface	15
RFM2g Error Codes	17
RFM2g API Functions	19
RFM2g Opening and Closing API Functions	22
RFM2g Configuration API Functions	26
RFM2g Data Transfer API Functions	48
RFM2g Interrupt Event API Functions	60
RFM2g Utility API Functions	79

Introduction

The application program interface (*API*) that comes with the RFM2g device driver provides the application developer with a common API for developing portable RFM2g applications that are platform-independent. The API is located in the file `rfm2g_api.h`.

The `rfm2g_api.h` file defines the common application program interface provided by the driver. Use this header file in application programs to access the rfm2g device. This file is suitable for inclusion in either a standard C or C++ compilation.

The API consists of this header file and libraries for the following development language:

- *ANSI-C Language Bindings* — A C-language API provides functions and macro definitions that assist the applications programmer in using the raw features of the device driver and its associated hardware.

Applications that take advantage of the API will be portable to other platforms because the idiosyncrasies of the host system are abstracted by the API.

VMIC currently supports an RFM2g API for the following environments:

- Wind River Tornado 2 (VxWorks) — Power PC and x86
- Microsoft Windows NT 4.0, Windows 2000 and Windows XP
- Red Hat 7.2 or 7.3 (Linux)
- Hewlett-Packard (HP) Tru64 5.1A
- SGI IRIX 6.5

The driver contains API functions that enable you to:

- Open and close the driver
- Configure the board
- Transfer data
- Control/handle interrupt events

Before an application program can access an RFM2g device, that device must be opened. When the device is opened successfully, a handle is returned to the application which is used in all subsequent operations involving the device driver. The handle's first call must be used to initialize the API.

In addition to the services provided by the driver, an application program can directly access the shared memory contained on the RFM2g interface. When the application opens the RFM2g device, the memory area of the RFM2g device can be mapped into the virtual memory space of the application program. The program can then treat the RFM2g as if it were an ordinary memory. Indirect pointer references to the RFM2g will work normally.

NOTE: The operating system does not perform memory bounds checking if the indirect method is used. Data corruption of system memory is likely to occur if a user application increments a pointer beyond the end of valid Reflective Memory.

Using the Application Program Interface

The RFM2g driver's `rfm2g_util.c` program is a command line application that enables you to exercise almost all of the driver's API functions. Once you have built and are running the `rfm2g_util.c` program, enter `help` at the prompt to obtain a list of commands that can be run using `rfm2g_util.c`. To obtain detailed help for a specific command, enter `help [command]`, where `[command]` is any of the commands listed by the `help` command.

The code in the `rfm2g_util.c` file can be used as an example of how to use each API command by examining the function `do[command]()`, where `[command]` is any of the commands listed by the `help` command.

Opening the RFM2g Driver

Before using any of the RFM2g commands, you must call the `RFM2gOpen()` function to open the RFM2g device using the code shown on page 16.

Routine Code for Use with API Function Examples

The following routine, `rfm2gTestApiCommand()`, can be used with the example code listed at the end of each function.

The code does the following:

- Opens the RFM2g driver
- Executes the code for the inserted API function
- Prints an error message when an error occurs
- Closes the RFM2g driver
- Returns an `RFM2G_STATUS` code

To use this routine, replace the line:

```
/* Place API command example here */
```

with the code provided in the API function example.

```
#define DEVICE /* Place OS specific device name in quotes before this comment */
RFM2G_STATUS rfm2gTestApiCommand(void)
{
    RFM2GHANDLE Handle;
    RFM2G_STATUS result;

    /* Open the Reflective Memory device */
    result = RFM2gOpen( DEVICE, &Handle );
    if( result != RFM2G_SUCCESS )
    {
        printf( "ERROR: RFM2gOpen() failed.\n" );
        printf( "ERROR MSG: %s\n", RFM2gErrorMsg(result));
        return(-1);
    }

    {
        /* Place API command example here */
    }
    if( result != RFM2G_SUCCESS )
    {
        printf( "ERROR: API command returned error.\n" );
        printf( "ERROR MSG: %s\n", RFM2gErrorMsg(result));
    }
    /* Close the Reflective Memory device */
    RFM2gClose( &Handle );
    return(result);
}
```

NOTE: Three sample application programs (**rfm2g_sender.c**, **rfm2g_receiver.c** and **rfm2g_map.c**) are delivered with the RFM2g driver that show how to use the driver and API with your application. See Chapter 3, "RFM2g Sample Applications" on page 147 for more information.

RFM2g Error Codes

The following is a list of the common error codes that can be output by the RFM2g device driver. Drivers may define additional error codes that are driver specific. Refer to your driver-specific manual for more information.

You may call the RFM2g API's `RFM2gErrorMsg()` function with the error code to retrieve a description of the error code.

NOTES:

Error code values are driver-specific.

Use the error code name instead of the value in user applications.

Table 1-1 Common RFM2g Error Codes

Error Code	Description
RFM2G_SUCCESS	No error
RFM2G_NOT_IMPLEMENTED	Function is not currently implemented
RFM2G_DRIVER_ERROR	An error occurred during a call to the driver
RFM2G_TIMED_OUT	A wait timed out
RFM2G_LOW_MEMORY	A memory allocation failed
RFM2G_MEM_NOT_MAPPED	Memory is not mapped for this device
RFM2G_OS_ERROR	Function failed for other OS defined error
RFM2G_EVENT_IN_USE	The Event is already being waited on
RFM2G_NOT_SUPPORTED	Capability not supported by this particular Driver/Board
RFM2G_NOT_OPEN	Device not open
RFM2G_NO_RFM2G_BOARD	Driver did not find RFM2g device
RFM2G_BAD_PARAMETER_1	Parameter 1 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_2	Parameter 2 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_3	Parameter 3 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_4	Parameter 4 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_5	Parameter 5 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_6	Parameter 6 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_7	Parameter 7 to the function is either NULL or invalid

Table 1-1 Common RFM2g Error Codes (Continued)

Error Code	Description
RFM2G_BAD_PARAMETER_8	Parameter 8 to the function is either NULL or invalid
RFM2G_BAD_PARAMETER_9	Parameter 9 to the function is either NULL or invalid
RFM2G_OUT_OF_RANGE	Board offset/range extends outside board memory
RFM2G_MAP_NOT_ALLOWED	Desired board offset is not legal for board memory size
RFM2G_LINK_TEST_FAIL	Ring continuity test failed
RFM2G_MEM_READ_ONLY	Function attempted to change memory outside of User Memory area
RFM2G_UNALIGNED_OFFSET	An offset is not properly aligned for the corresponding data width
RFM2G_UNALIGNED_ADDRESS	An address is not properly aligned for the corresponding data width
RFM2G_LSEEK_ERROR	The lseek(2) operation preceding a read or write failed
RFM2G_READ_ERROR	The read(2) operation was not successful
RFM2G_WRITE_ERROR	The write(2) operation was not successful
RFM2G_HANDLE_NOT_NULL	Cannot initialize a non-NULL handle pointer
RFM2G_MODULE_NOT_LOADED	The driver module has not been loaded into the kernel
RFM2G_NOT_ENABLED	An attempt was made to use an interrupt that has not been enabled
RFM2G_ALREADY_ENABLED	An attempt was made to enable an interrupt that was already enabled
RFM2G_EVENT_NOT_IN_USE	No process is waiting on the interrupt
RFM2G_BAD_RFM2G_BOARD_ID	Invalid RFM2g board ID
RFM2G_NULL_DESCRIPTOR	RFM2GHANDLE is null
RFM2G_WAIT_EVENT_CANCELLED	Wait for event canceled
RFM2G_DMA_FAILED	DMA operation failed
RFM2G_NOT_INITIALIZED	Cannot initialize a handle pointer
RFM2G_UNALIGNED_LENGTH	An offset is not properly aligned for the corresponding data length
RFM2G_MAX_ERROR_CODE	Invalid error code

RFM2g API Functions

The following RFM2g API functions in the rfm2g_api.h file can be used with the RFM2g driver.

Table 1-2 RFM2g API Functions

RFM2g Opening and Closing API Functions	
API Function	Description
RFM2gOpen()	Opens the RFM2g driver and returns an RFM2g handle.
RFM2gClose()	Closes an open RFM2g handle and releases resources allocated to it.
RFM2g Configuration API Functions	
API Function	Description
RFM2gGetConfig()	Obtains a copy of the RFM2GCONFIG configuration structure.
RFM2gUserMemory()	Maps RFM2g memory to the user space.
RFM2gUnMapUserMemory()	Unmaps RFM2g memory from the user space that was mapped using RFM2gUserMemory() .
RFM2gNodeID()	Returns the RFM2g device node ID.
RFM2gBoardID()	Returns the ID of the board corresponding to the passed-in handle.
RFM2gSize()	Returns the total amount of memory space available on the RFM2g device.
RFM2gFirst()	Returns the first available RFM2g offset.
RFM2gDeviceName()	Returns the device name associated with an RFM2g handle.
RFM2gDllVersion()	Returns the DLL version.
RFM2gDriverVersion()	Returns the RFM2g device driver version.
RFM2gGetDMAThreshold()	Returns the current DMA threshold value.
RFM2gSetDMAThreshold()	Sets the transfer size at which reads and writes will use DMA.
RFM2gGetDMAByteSwap()	Returns the state of DMA (Direct Memory Access) byte swapping specified by RFM2gSetDMAByteSwap() .
RFM2gSetDMAByteSwap()	Sets the current on/off state of DMA (Direct Memory Access) byte swapping.

Table 1-2 RFM2g API Functions (Continued)

RFM2g Configuration API Functions	
API Function	Description
RFM2gGetPIOByteSwap()	Returns the state of PIO (Programmed IO) byte swapping specified by RFM2gSetPIOByteSwap() .
RFM2gSetPIOByteSwap()	Sets the current on/off state of PIO (Programmed IO) byte swapping.
RFM2g Data Transfer API Functions	
API Function	Description
RFM2gRead()	Reads one or more bytes starting at an offset in Reflective Memory.
RFM2gWrite()	Writes one or more bytes starting at an offset in Reflective Memory.
RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()	Reads a single byte, word or longword from an offset in Reflective Memory.
RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()	Writes a single byte, word or longword to an offset in Reflective Memory.
RFM2g Interrupt Event API Functions	
API Function	Description
RFM2gEnableEvent()	Enables reception of an RFM2g interrupt event.
RFM2gDisableEvent()	Disables the reception of an RFM2g event.
RFM2gSendEvent()	Transmits the specified RFM2g interrupt event to one or all other RFM2g node IDs.
RFM2gWaitForEvent()	Blocks the calling process until an occurrence of the specified RFM2g interrupt event is received or a timeout (if enabled) expires.
RFM2gEnableEventCallback()	Enables the interrupt notification for one event on one board.
RFM2gDisableEventCallback()	Disables the interrupt notification for one event on one board.
RFM2gClearEvent()	Flushes all pending events for a specified event type.

Table 1-2 RFM2g API Functions (Continued)

RFM2g Interrupt Event API Functions	
API Function	Description
RFM2gCancelWaitForEvent()	Cancels any pending RFM2gWaitForEvent() calls for a specified event type.
RFM2g Utility API Functions	
API Function	Description
RFM2gErrorMsg()	Returns a pointer to a text string describing an error code.
RFM2gGetLed()	Retrieves the current on/off state of the Reflective Memory board's STATUS LED.
RFM2gSetLed()	Sets the on/off state of the Reflective Memory board's STATUS LED.
RFM2gCheckRingCont()	Returns the fiber ring continuity through nodes.
RFM2gGetDebugFlags()	Retrieves a copy of all RFM2g device driver debug control flags.
RFM2gSetDebugFlags()	Sets or clears the device driver debug control flags.

RFM2g Opening and Closing API Functions

The following API functions in the `rfm2g_api.h` file can be used to open and close the RFM2g driver.

Table 1-3 RFM2g Opening and Closing API Functions

API Function	Description
RFM2gOpen()	Opens the RFM2g driver and returns an RFM2g handle.
RFM2gClose()	Closes an open RFM2g handle and releases resources allocated to it.

RFM2gOpen()

The `RFM2gOpen()` function connects the application program to the RFM2g device driver and API library. The API library will open the specified RFM2g device and return a handle which the program must use in all further references to the RFM2g device.

Several programs and execution threads may have the same RFM2g interface open at any given time. The driver and the API library are thread-safe; however, it is the responsibility of the application program to perform whatever access synchronization is needed for any data structures managed by the program in the RFM2g area.

NOTE: Because RFM2g interface device names are dynamically assigned, users who have multiple RFM2g devices in a chassis should exercise care when replacing RFM2g boards. Removing an RFM2g interface may cause the name assigned to other RFM2g boards to be changed.

Operation

Most services available via the API require the use of an RFM2GHANDLE to identify the connection between the application program and the opened RFM2g interface.

Syntax

```
STDRFM2GCALL RFM2gOpen( char *DevicePath,  
                        RFM2GHANDLE *rh );
```

Parameters

<i>DevicePath</i>	Path to special device file (I). Refer to your driver-specific manual for the format of <i>DevicePath</i> .
<i>rh</i>	Pointer to an RFM2GHANDLE structure (IO).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_BAD_PARAMETER_1 — NULL or invalid <i>DevicePath</i> .
	RFM2G_BAD_PARAMETER_2 — <i>rh</i> is NULL
	RFM2G_HANDLE_NOT_NULL — <i>*rh</i> is not NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_LOW_MEMORY — System refused request.
	RFM2G_NO_RF2G_BOARD — No RFM2g device found.
	RFM2G_BAD_RF2G_BOARD_ID — RFM2g device has bad board ID.

Example

See "Routine Code for Use with API Function Examples" on page 15 for an example of the `RFM2gOpen()` command.

Related Commands

- `RFM2gClose()`

RFM2gClose()

The `RFM2gClose()` function allows an application program to terminate its connection with the RFM2g services. Once the RFM2g handle is closed, all of the facilities using that handle are no longer accessible, including the local RFM2g memory, which may be mapped into the application program's virtual memory space.

Syntax

```
STDRFM2GCALL RFM2gClose( RFM2GHANDLE *rh );
```

Parameters

rh Initialized previously with a call to **RFM2gOpen()** (I).

Return Values

Success `RFM2G_SUCCESS`

Failure `RFM2G_NULL_DESCRIPTOR` — *rh* is NULL.

`RFM2G_OS_ERROR` — Operating system (OS) returned an error.

`RFM2G_NOT_IMPLEMENTED` — API function is not implemented in the driver.

`RFM2G_NOT_OPEN` — Device is not open.

Example

See "Routine Code for Use with API Function Examples" on page 15 for an example of the `RFM2gClose()` command.

Related Commands

- `RFM2gOpen()`

RFM2g Configuration API Functions

The following API functions in the `rfm2g_api.h` file can be used to perform configuration on the RFM2g driver.

Table 1-4 RFM2g Configuration API Functions

API Function	Description
RFM2gGetConfig()	Obtains a copy of the RFM2GCONFIG configuration structure.
RFM2gUserMemory()	Maps RFM2g memory to the user space.
RFM2gUnMapUserMemory()	Unmaps RFM2g memory from the user space that was mapped using RFM2gUserMemory() .
RFM2gNodeID()	Returns the RFM2g device node ID.
RFM2gBoardID()	Returns the ID of the board corresponding to the passed-in handle.
RFM2gSize()	Returns the total amount of memory space available on the RFM2g device.
RFM2gFirst()	Returns the first available RFM2g offset.
RFM2gDeviceName()	Returns the device name associated with an RFM2g handle.
RFM2gDllVersion()	Returns the DLL version.
RFM2gDriverVersion()	Returns the RFM2g device driver version.
RFM2gGetDMAThreshold()	Returns the current DMA threshold value.
RFM2gSetDMAThreshold()	Sets the transfer size at which reads and writes will use DMA.
RFM2gGetDMAByteSwap()	Returns the state of DMA (Direct Memory Access) byte swapping specified by RFM2gSetDMAByteSwap() .
RFM2gSetDMAByteSwap()	Sets the current on/off state of DMA (Direct Memory Access) byte swapping.
RFM2gGetPIOByteSwap()	Returns the state of PIO (Programmed IO) byte swapping specified by RFM2gSetPIOByteSwap() .
RFM2gSetPIOByteSwap()	Sets the current on/off state of PIO (Programmed IO) byte swapping.

RFM2gGetConfig()

The `RFM2gGetConfig()` function allows an application program to obtain a copy of the RFM2GCONFIG hardware configuration structure created by the device driver during its initialization.

The RFM2GCONFIG structure is driver-specific. Refer to your driver's installation manual for structure definition information.

Syntax

```
STDRFM2GCALL RFM2gGetConfig( RFM2GHANDLE rh,  
                             RFM2GCONFIG *Config );
```

Parameters

rh Handle to open RFM2g device (I).
Config Pointer to RFM2GCONFIG structure to be filled (O).

Return Values

Success RFM2G_SUCCESS
Failure RFM2G_NULL_DESCRIPTOR — *rh* is NULL.
RFM2G_OS_ERROR — Operating system (OS) returned an error.
RFM2G_NOT_OPEN — Device is not open.
RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
RFM2G_BAD_PARAMETER_2 — *Config* is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2GCONFIG Rfm2gConfig;  
result = RFM2gGetConfig(Handle, &Rfm2gConfig);
```

Related Commands

- RFM2gNodeID()
- RFM2gBoardID()
- RFM2gSize()
- RFM2gFirst()
- RFM2gDeviceName()
- RFM2gDllVersion()
- RFM2gDriverVersion()

RFM2gUserMemory()

The `RFM2gUserMemory()` function maps the Reflective Memory address space to a user-level pointer, allowing direct access to RFM memory via pointer dereferencing. All transfers using this pointer will use Programmed IO (PIO) and will not use DMA.

Syntax

```
STDRFM2GCALL RFM2gUserMemory( RFM2GHANDLE rh, void **UserMemoryPtr,
                               RFM2G_UINT32 Offset, RFM2G_UINT32 Pages );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>UserMemoryPtr</i>	Where to put the pointer to mapped RFM2g space (IO).
<i>Offset</i>	Base byte offset of RFM2g memory to map (I).
<i>Pages</i>	Number of pages to map.

NOTES:

Page size is system-dependent.
Refer to your driver-specific manual for information on using this parameter.

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>UserMemoryPtr</i> is NULL.
	RFM2G_BAD_PARAMETER_4 — <i>Pages</i> is 0.
	RFM2G_MAP_NOT_ALLOWED — Invalid map <i>Offset</i> and <i>Pages</i> (Page size beyond size of memory on RFM2g device.).
RFM2G_OUT_OF_RANGE — Mapping would go beyond end of RFM2g memory.	

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 Offset = 0;
RFM2G_UINT32 Pages = 1;
void* pUser;
result = RFM2gUserMemory(Handle, (void**)&pUser, Offset, Pages);
```

Related Commands

- RFM2gUnMapUserMemory()

RFM2gUnMapUserMemory()

The `RFM2gUnMapUserMemory()` function unmaps a memory space mapped by `RFM2gUserMemory()`.

Syntax

```
STDRFM2GCALL RFM2gUnMapUserMemory( RFM2GHANDLE rh, void **UserMemoryPtr,  
                                     RFM2G_UINT32 Pages );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>UserMemoryPtr</i>	Pointer to mapped RFM2g space (IO).
<i>Pages</i>	The number of pages originally mapped to <i>UserMemoryPtr</i> (I). Refer to your driver-specific manual for information on using this parameter.

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>UserMemoryPtr</i> is NULL.
	RFM2G_BAD_PARAMETER_3 — <i>Pages</i> is 0.
	RFM2G_OUT_OF_RANGE — Mapping would go beyond end of RFM2g memory.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15 after including the example code for RFM2gUserMemory():

```
RFM2G_UINT32 Offset = 0;
RFM2G_UINT32 Pages = 1;
void* pUser = NULL;
result = RFM2gMapUserMemory(Handle, (void*)&pUser, Offset, Pages);
if (result == RFM2G_SUCCESS)
{
    result = RFM2gUnMapUserMemory(Handle, &pUser, Pages);
}
```

Related Commands

- RFM2gUserMemory()

RFM2gNodeID()

The `RFM2gNodeID()` function returns the value of the RFM2g device node ID. Each RFM2g device on an RFM2g network is uniquely identified by its node ID, which is manually set by jumpers on the device when the RFM2g network is installed. The driver determines the node ID when the device is initialized.

Syntax

```
STDRFM2GCALL RFM2gNodeID( RFM2GHANDLE rh,  
                           RFM2G_NODE *NodeIdPtr );
```

Parameters

rh Handle to currently opened RFM2g device (I).
NodeIdPtr Node ID of the currently opened RFM2g device (O).

Return Values

Success RFM2G_SUCCESS
Failure RFM2G_NULL_DESCRIPTOR — *rh* is NULL.
RFM2G_OS_ERROR — Operating system (OS) returned an error.
RFM2G_NOT_OPEN — Device is not open.
RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
RFM2G_BAD_PARAMETER_2 — *NodeIdPtr* is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_NODE NodeId;  
  
result = RFM2gNodeID(Handle, &NodeId);
```

Related Commands

- RFM2gBoardID()
- RFM2gGetConfig()

RFM2gBoardID()

The `RFM2gBoardID()` function returns the RFM2g interface model type. Each RFM2g model type is uniquely identified by a numeric value assigned by VMIC and recorded as a fixed constant in an RFM2g hardware register. The driver and support library read this value when the device is opened. The application program uses `RFM2gBoardId()` to obtain that value.

Syntax

```
STDRFM2GCALL RFM2gBoardID( RFM2GHANDLE rh,  
                             RFM2G_UINT8 *BoardIdPtr );
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>BoardIdPtr</i>	Board ID of the currently opened RFM2g device (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>BoardIdPtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT8 BoardId;  
  
result = RFM2gBoardID(Handle, &BoardId)
```

Related Commands

- RFM2gNodeID()
- RFM2gGetConfig()

RFM2gSize()

The `RFM2gSize()` function returns the total amount of memory space available on the RFM2g device. The application program may access RFM2g space between offset `RFM2gFirst()` and `RFM2gSize(rh)-1`.

RFM2g boards may be configured with a variety of memory sizes. The device driver and API library determine the amount of memory contained on an RFM2g device as it is opened. An application program may then use `RFM2gSize()` to obtain the number of bytes on the board.

Syntax

```
STDRFM2GCALL long RFM2gSize( RFM2GHANDLE rh, RFM2g_UINT32 *SizePtr );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>SizePtr</i>	Pointer to variable that is filled with the RFM2g size value (I).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>SizePtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 Size
result = RFM2GgSize(Handle, &Size);
if (result == RFM2G_SUCCESS)
{
    printf( "The RFM2g interface contains %lu bytes of memory.\n", Size );
}
```

Related Commands

- `RFM2gGetConfig()`
- `RFM2gFirst()`

RFM2gFirst()

The `RFM2gFirst()` function returns the first RFM2g offset available for use by an application program. The entire memory space of the RFM2g device is mapped into the virtual address space of the application program.

Syntax

```
STDRFM2GCALL RFM2gFirst( RFM2GHANDLE rh,  
                        RFM2G_UINT32 *FirstPtr );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>FirstPtr</i>	Pointer to the variable filled with the offset of the first location of RFM memory.

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>FirstPtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 first;  
RFM2gFirst(Handle, &first);
```

Related Commands

- `RFM2gGetConfig()`
- `RFM2gSize()`

RFM2gDeviceName()

The `RFM2gDeviceName()` function returns a null-terminated string containing the first 64 characters of the device file name associated with the given RFM2g file handle.

Syntax

```
STDRFM2GCALL RFM2gDeviceName( RFM2GHANDLE rh,
                               char *NamePtr );
```

Parameters

<i>rh</i>	Initialized previously with a call to <code>RFM2gOpen()</code> (I).
<i>NamePtr</i>	Pointer to the char array that is filled with the device filename for the given RFM2g device (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>NamePtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_CHAR name[64];
name[0] = 0;

result = RFM2gDeviceName(Handle, name);
if(result == RFM2G_SUCCESS)
{
    printf("RFM2gDeviceName : %s\n", name);
}
```

Related Commands

- `RFM2gGetConfig()`

RFM2gDllVersion()

The `RFM2gDllVersion()` function returns an ASCII string with which an application program can determine the version of the DLL or API library. This string contains the production release level of the library and is unique between different versions of the API library.

Syntax

```
STDRFM2GCALL RFM2gDllVersion( RFM2GHANDLE rh,
                              char *VersionPtr );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>VersionPtr</i>	Pointer to where the string containing the production release level of the DLL or API library (O) will be copied.

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>VersionPtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_CHAR version[64];
version[0] = 0;

result = RFM2gDllVersion(Handle, version);
if(result == RFM2G_SUCCESS)
{
    printf("RFM2gDllVersion:%s\n", version);
}
```

Related Commands

- `RFM2gDriverVersion()`
- `RFM2gGetConfig()`

RFM2gDriverVersion()

The `RFM2gDriverVersion()` function returns an ASCII string with which an application program can determine the VMIC production release version of the underlying RFM2g device driver.

Syntax

```
STDRFM2GCALL RFM2gDriverVersion( RFM2GHANDLE rh,
                                  char *VersionPtr );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>VersionPtr</i>	Pointer to where the string containing the production version of the RFM2g device driver will be copied (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>VersionPtr</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_CHAR drvVersion[64];
drvVersion[0] = 0;

result = RFM2gDriverVersion(Handle, drvVersion);
if(result == RFM2G_SUCCESS)
{
    printf("RFM2gDriverVersion : %s\n", drvVersion);
}
```

Related Commands

- RFM2gDllVersion()
- RFM2gGetConfig()

RFM2gGetDMAThreshold()

The `RFM2gGetDMAThreshold()` function returns the length of the current minimum DMA I/O request of the device driver. The RFM2g device driver will use the bus master DMA feature present on some RFM2g devices if an I/O request qualifies (i.e. if the size is larger than or equal to the *Threshold*). One of the criteria for performing the DMA is that the I/O transfer be long enough that the time saved by performing the DMA offsets the overhead processing involved with initializing the DMA itself. The default DMA threshold is driver-dependent. Refer to your driver-specific manual for the default DMA threshold value.

This command is useful since the amount of this overhead can vary between host computer configurations. The application program can set a new threshold using the `RFM2gSetDMAThreshold()` function.

Syntax

```
STDRFM2GCALL RFM2gGetDMAThreshold( RFM2GHANDLE rh,  
                                     RFM2G_UINT32 *Threshold );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Threshold</i>	Pointer to the variable where the current DMA threshold value will be copied (I).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver. RFM2G_BAD_PARAMETER_2 — <i>Threshold</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 Threshold;  
result = RFM2gGetDMAThreshold(Handle, &Threshold);
```

Related Commands

- RFM2gSetDMAThreshold()
- RFM2gRead()
- RFM2gWrite()

RFM2gSetDMAThreshold()

The `RFM2gSetDMAThreshold()` function sets the transfer size at which reads and writes will use DMA to transfer data. If `RFM2gRead()` or `RFM2gWrite()` is called, DMA will be used if the size of the data is larger than or equal to the *Threshold*. A threshold can be set per device.

The amount of cycles taken to set up a DMA transfer can increase the transfer time for small transfer sizes. The transfer size for which DMAs are more efficient than standard transfers varies, depending on the system.

DMA is generally preferred over the Programmed IO (PIO) method for transferring data. PIO operations require the usage of the CPU to process the transfer, while DMA enables the Reflective Memory controller to access system memory while leaving the CPU's resources unaffected. However, the best value to use (i.e. PIO vs. DMA) is system-dependent. The RFM2g driver performs approximately 5 PCI accesses to set up and process a DMA request and generates an interrupt on completion of the DMA operation. In general, DMA is the preferred method if a PIO transfer requires more than 6 to 10 PCI cycles to complete.

A *Threshold* value of `0xFFFFFFFF` specifies that DMAs will never be used for data transfer.

NOTE: The default value for the DMA *Threshold* is driver-dependent and should be changed *only* if recommended by the driver's documentation. Refer to your driver-specific manual for more information, including the default value.

Syntax

```
STDRFM2GCALL RFM2gSetDMAThreshold( RFM2GHANDLE rh,  
                                     RFM2G_UINT32 Threshold );
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>Threshold</i>	New DMA threshold value (I).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
/* Set DMA threshold to 256 bytes */  
  
result = RFM2gSetDMAThreshold(Handle, 256);
```

Related Commands

- RFM2gGetDMAThreshold()
- RFM2gWrite()
- RFM2gRead()

RFM2gGetDMAByteSwap()

The `RFM2gGetDMAByteSwap()` function returns the state of DMA (Direct Memory Access) byte swapping hardware, which is specified by the `RFM2gSetDMAByteSwap()` function.

Syntax

```
STDRFM2GCALL RFM2gGetDMAByteSwap( RFM2GHANDLE rh,  
                                   RFM2G_BOOL *byteSwap )
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>byteSwap</i>	Pointer to where the state of the DMA byte swap hardware is written (RFM2G_TRUE when DMA byte swapping is ON, or RFM2G_FALSE when DMA byte swapping is OFF) (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver. RFM2G_BAD_PARAMETER_2 — <i>byteSwap</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_BOOL byteSwap;  
  
result = RFM2gGetDMAByteSwap(Handle, &byteSwap);
```

Related Commands

- RFM2gSetDMAByteSwap()
- RFM2gWrite()
- RFM2gRead()

RFM2gSetDMAByteSwap()

The `RFM2gSetDMAByteSwap()` function enables or disables byte swapping DMA transfers to or from an RFM2g device. This function provides 4-byte swaps only (i.e. byte swapping based on size is not performed by the RFM2g device).

NOTE: DMA byte swapping may be enabled by default when the driver has been built for use on big endian systems. Refer to your driver-specific manual for the default setting of DMA byte swapping.

Syntax

```
STDRFM2GCALL RFM2gSetDMAByteSwap( RFM2GHANDLE rh,
                                   RFM2G_BOOL byteSwap )
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>byteSwap</i>	The state of the DMA byte swap (RFM2G_TRUE=>ON or RFM2G_FALSE=>OFF) (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_BOOL byteSwap = 1; /* Turn byte swap on */

result = RFM2gSetDMAByteSwap(Handle, byteSwap);
```

Related Commands

- RFM2gGetDMAByteSwap()
- RFM2gWrite()
- RFM2gRead()

RFM2gGetPIOByteSwap()

The `RFM2gGetPIOByteSwap()` function returns the state of PIO (Programmed IO) byte swapping, which is specified using the `RFM2gSetPIOByteSwap()` function.

Refer to "Data Transfer Considerations" on page 48 for information on byte swapping and PIO.

Syntax

```
STDRFM2GCALL RFM2gGetPIOByteSwap( RFM2GHANDLE rh,
                                   RFM2G_BOOL *byteSwap )
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>byteSwap</i>	Pointer to where the state of the PIO byte swap is written (RFM2G_TRUE when PIO byte swapping is ON, or RFM2G_FALSE when PIO byte swapping is OFF) (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>byteSwap</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_BOOL byteSwap;

result = RFM2gGetPIOByteSwap(Handle, &byteSwap);
```

Related Commands

- RFM2gSetPIOByteSwap()
- RFM2gRead()
- RFM2gWrite()
- RFM2gUserMemory()
- RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()
- RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()

RFM2gSetPIOByteSwap()

The `RFM2gSetPIOByteSwap()` function enables or disables byte swapping of PIO (Programmed IO) transfers to or from an RFM2g device. This function provides 4-byte swaps (i.e. byte swapping based on size is not performed by the RFM2g device).

Refer to "Data Transfer Considerations" on page 48 for information on byte swapping and PIO.

NOTE: PIO byte swapping is enabled by default when the driver has been built for use on big endian systems. Refer to your driver-specific manual for the default setting of PIO byte swapping.

Syntax

```
STDRFM2GCALL RFM2gSetPIOByteSwap( RFM2GHANDLE rh,  
                                   RFM2G_BOOL byteSwap )
```

Parameters

<i>rh</i>	Handle to currently opened RFM2g device (I).
<i>byteSwap</i>	The state of the PIO byte swap (RFM2G_TRUE=>ON or RFM2G_FALSE=>OFF) (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_BOOL byteSwap = 1 /* Turn byte swap on */  
  
result = RFM2gSetPIOByteSwap(Handle, byteSwap);
```

Related Commands

- RFM2gGetPIOByteSwap()
- RFM2gRead()
- RFM2gWrite()
- RFM2gUserMemory()
- RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()
- RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()

RFM2g Data Transfer API Functions

The following API functions in the `rfm2g_api.h` file can be used to transfer data with the RFM2g driver.

Table 1-5 RFM2g Data Transfer API Functions

API Function	Description
RFM2gRead()	Reads one or more bytes starting at an offset in Reflective Memory.
RFM2gWrite()	Writes one or more bytes starting at an offset in Reflective Memory.
RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()	Reads a single byte, word or longword from an offset in Reflective Memory.
RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()	Writes a single byte, word or longword to an offset in Reflective Memory.

Data Transfer Considerations

The following information should be considered when transferring data using the API commands in this section pointers obtained from `RFM2gUserMemory()`, or any of the following `rfm2g_util.c` command line interpreter commands:

- **peek8**
- **peek16**
- **peek32**
- **peek64**
- **poke8**
- **poke16**
- **poke32**
- **poke64**
- **read**
- **write**

See "rfm2g_util.c Utility Program" on page 91 for more information on the command line interpreter.

Big Endian and Little Endian Data Conversions

X86 (Intel-based) processors use little endian byte ordering when storing sequences of bytes while other processors, such as the Sun family of SPARC processors and Power PC, use the big endian method.

The RFM2g API accesses Reflective Memory using little endian byte ordering. If some systems on the Reflective Memory network are using little endian ordering and others are using big endian ordering, you may have to perform the necessary byte swapping prior to using the RFM2g driver with the multibyte data shared between the systems, depending on the DMA and PIO byte swap settings. See `RFM2gSetDMAByteSwap()` on page 44 and `RFM2gSetPIOByteSwap()` on page 46 for more information.

Using Direct Memory Access (DMA)

Based on the size of the data, the user must determine whether or not to use DMA to transfer data. DMA bypasses a system's CPU, allowing the system CPU to continue execution while system memory is being accessed by the RFM2g device.

An application program will use DMA according to its own I/O requirements. The RFM2g driver will attempt to fulfill data moving requests using the bus master DMA feature of the RFM2g interfaces if the transfer is greater than the current DMA threshold.

If the request does not meet the constraints listed above, the driver will move the data using `memcpy()`.

Some systems may require cache management routines to be called before and/or after DMA accesses, and may also place restrictions on the size of the DMA transfer. Refer to your driver-specific manual to determine whether or not cache management functions should be called and for any restrictions placed on DMA transfers. See `RFM2gSetDMAByteSwap()` on page 44 for more information.

RFM2gRead()

The `RFM2gRead()` function is used to transfer one or more bytes from RFM2g memory to system memory.

The RFM2g driver attempts to fulfill the `RFM2gRead()` request using the bus master DMA feature available on the RFM2g device. The driver will move the data using the DMA feature if the length of the I/O request is at least as long as the minimum DMA threshold.

NOTES:

See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

The DMA feature can be used as an alternative method for transferring data. See "Using Direct Memory Access (DMA)" on page 49 for more information.

If byte swapping is enabled on the RFM2g device, the *Offset* and *Length* must be width aligned.

If the RFM2g device does not support the bus master DMA feature, or if the I/O request does not meet the constraints listed above, then the driver will move the data using Programmed IO (PIO).

Refer to "Data Transfer Considerations" on page 48 for information on byte swapping.

CAUTION: An application program must not attempt to access the RFM2g contents at an offset less than that returned by the `RFM2gFirst()` function.

Syntax

```
STDRFM2GCALL RFM2gRead( RFM2GHANDLE rh,
                        RFM2G_UINT32 Offset,
                        void *Buffer,
                        RFM2G_UINT32 Length );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Offset</i>	Width-aligned offset to Reflective Memory at which to begin the read (I). Valid offset values are 0x0 to 0x3FFFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFFF for 128 Mbyte cards.
<i>Buffer</i>	Pointer to where data is copied from Reflective Memory (O).
<i>Length</i>	Number of bytes to transfer (I). Valid values are 0 to ([<i>RFM Size</i>] - <i>rfmOffset</i>).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_3 — <i>Buffer</i> is NULL.
	RFM2G_BAD_PARAMETER_4 — <i>Length</i> is not aligned to data width.
	RFM2G_OUT_OF_RANGE — <i>Offset</i> and <i>Length</i> is beyond the end of memory on RFM2G device.
	RFM2G_DMA_FAILED — DMA failed.
	RFM2G_UNALIGNED_OFFSET — <i>Offset</i> not aligned with data size.
	RFM2G_READ_ERROR — Data not aligned and/or system error.
RFM2G_LSEEK_ERROR — Failure of lseek(2) command.	
RFM2G_UNALIGNED_ADDRESS — <i>Buffer</i> is not aligned to data width.	

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT8 Buffer[128];
RFM2G_UINT32 rfmBytes = sizeof(Buffer);
RFM2G_UINT32 Offset = 0;

result = RFM2gRead(Handle, Offset, Buffer, rfmBytes);
```

Related Commands

- RFM2gWrite()
- RFM2gSetDMAThreshold()
- RFM2gSetDMAByteSwap()
- RFM2gSetPIOByteSwap()

RFM2gWrite()

The `RFM2gWrite()` function transfers one or more I/O data buffers from the application program to the RFM2g node beginning at the specified aligned memory offset.

If the RFM2g interface supports the bus master DMA feature and the I/O request meets certain constraints, the RFM2g device driver will use DMA to perform the I/O transfer. See the discussion for the `RFM2gRead()` function for a description of the constraints for the DMA transfer support.

The `RFM2gWrite()` writes one or more bytes starting at an offset in Reflective Memory (i.e. allows an application program to output (write) arbitrary-sized I/O buffers).

Operation

DMA will be used if *Length* is equal to or greater than the DMA threshold.

NOTES:

See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

The DMA feature can be used as an alternative method for transferring data. See "Using Direct Memory Access (DMA)" on page 49 for more information.

If byte swapping is enabled on the RFM2g device, the *Offset* and *Length* must be width aligned.

Syntax

```
STDRFM2GCALL RFM2gWrite( RFM2GHANDLE rh,
                          RFM2G_UINT32 Offset,
                          void *Buffer,
                          RFM2G_UINT32 Length );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Offset</i>	Width-aligned offset in Reflective Memory at which to begin the write (I). Valid offset values are 0x0 to 0x3FFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFF for 128 Mbyte cards.
<i>Buffer</i>	Pointer to where data is copied to Reflective Memory (I).
<i>Length</i>	Number of bytes units to write (I). Valid values are 0 to ([<i>RFM Size</i>] - <i>rfmOffset</i>).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_3 — <i>Buffer</i> is NULL.
	RFM2G_BAD_PARAMETER_4 — <i>Length</i> is not aligned to data width.
	RFM2G_OUT_OF_RANGE — <i>Offset</i> and <i>Length</i> is beyond the end of memory on RFM2G device.
	RFM2G_DMA_FAILED — DMA failed.
	RFM2G_UNALIGNED_OFFSET — <i>Offset</i> not aligned with data size.
	RFM2G_WRITE_ERROR — Data not aligned and/or system error.
	RFM2G_READ_ERROR — Data not aligned and/or system error.
	RFM2G_LSEEK_ERROR — Failure of lseek(2) command.
	RFM2G_UNALIGNED_ADDRESS — <i>Buffer</i> is not aligned to data width.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT8 Buffer[4];
RFM2G_UINT32 rfmBytes = sizeof(Buffer);
RFM2G_UINT32 Offset = 0;
Buffer[0] = 0;
Buffer[1] = 1;
Buffer[2] = 2;
Buffer[3] = 3;

result = RFM2gWrite(Handle, Offset, (void*)Buffer, rfmBytes);
```

Related Commands

- RFM2gRead()
- RFM2gSetDMAThreshold()
- RFM2gSetDMAByteSwap()
- RFM2gSetPIOByteSwap()

RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()

The `RFM2gPeek()` functions return the contents of the specified RFM2g offset. The specified memory offset is accessed as either an 8-bit byte, a 16-bit word, a 32-bit longword or a 64-bit longword.

NOTE: See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

Operation

The `RFM2gPeek()` functions return the contents of the indicated RFM2g memory location and make no attempt to lock the RFM2g during the access.

Refer to "Data Transfer Considerations" on page 48 for information on byte swapping.

Syntax

```
STDRFM2GCALL RFM2gPeek8( RFM2GHANDLE rh,
                          RFM2G_UINT32 Offset,
                          RFM2G_UINT8 *Value );
```

```
STDRFM2GCALL RFM2gPeek16( RFM2GHANDLE rh,
                            RFM2G_UINT32 Offset,
                            RFM2G_UINT16 *Value );
```

```
STDRFM2GCALL RFM2gPeek32( RFM2GHANDLE rh,
                            RFM2G_UINT32 Offset,
                            RFM2G_UINT32 *Value );
```

```
STDRFM2GCALL RFM2gPeek64( RFM2GHANDLE rh,
                            RFM2G_UINT32 Offset,
                            RFM2G_UINT64 *Value );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Offset</i>	Offset in Reflective Memory from which to read (I).
<i>Value</i>	Pointer to where the value is read from <i>Offset</i> (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_3 — <i>Value</i> is NULL.
	RFM2G_OUT_OF_RANGE — <i>Offset</i> is beyond the end of RFM2G memory.
	RFM2G_UNALIGNED_OFFSET — <i>Offset</i> not aligned with data size.

Example (RFM2gPeek8())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT8 Value;
RFM2G_UINT32 Offset = 0;

result = RFM2gPeek8(Handle, Offset, &Value);
```

Example (RFM2gPeek16())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT16 Value;
RFM2G_UINT32 Offset = 0;

result = RFM2gPeek16(Handle, Offset, &Value);
```

Example (RFM2gPeek32())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 Value;
RFM2G_UINT32 Offset = 0;

result = RFM2gPeek32(Handle, Offset, &Value);
```

Example (RFM2gPeek64())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT64 Value;  
RFM2G_UINT32 Offset = 0;  
  
result = RFM2gPeek64(Handle, Offset, &Value);
```

Related Commands

- RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()
- RFM2gSetPIOByteSwap()

RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()

The `RFM2gPoke()` functions are used to update a value in the RFM2g using either an 8-bit byte, a 16-bit word, a 32-bit longword or a 64-bit longword access. No attempt at RFM2g shared memory locking is performed.

NOTE: See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

Syntax

```
STDRFM2GCALL RFM2gPoke8( RFM2GHANDLE rh,
                          RFM2G_UINT32 Offset,
                          RFM2G_UINT8 Value );

STDRFM2GCALL RFM2gPoke16( RFM2GHANDLE rh,
                           RFM2G_UINT32 Offset,
                           RFM2G_UINT16 Value );

STDRFM2GCALL RFM2gPoke32( RFM2GHANDLE rh,
                           RFM2G_UINT32 Offset,
                           RFM2G_UINT32 Value );

STDRFM2GCALL RFM2gPoke64( RFM2GHANDLE rh,
                           RFM2G_UINT32 Offset,
                           RFM2G_UINT64 Value );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Offset</i>	Offset in Reflective Memory from which to read (I).
<i>Value</i>	Value written to <i>Offset</i> (I).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_UNALIGNED_OFFSET — <i>Offset</i> not aligned with data size.

Example (RFM2gPoke8())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT8 Value = 0;
RFM2G_UINT32 Offset = 0;

result = RFM2gPoke8(Handle, Offset, Value);
```

Example (RFM2gPoke16())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT16 Value = 0;
RFM2G_UINT32 Offset = 0;

result = RFM2gPoke16(Handle, Offset, Value);
```

Example (RFM2gPoke32())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 Value = 0;
RFM2G_UINT32 Offset = 0;

result = RFM2gPoke32(Handle, Offset, Value);
```

Example (RFM2gPoke64())

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT64 Value = 0;
RFM2G_UINT32 Offset = 0;

result = RFM2gPoke64(Handle, Offset, Value);
```

Related Commands

- RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()
- RFM2gSetPIOByteSwap()

RFM2g Interrupt Event API Functions

The following API functions in the `rfm2g_api.h` file can be used to perform event-related operations with the RFM2g driver.

Table 1-6 RFM2g Interrupt Event API Functions

API Function	Description
RFM2gEnableEvent()	Enables reception of an RFM2g interrupt event.
RFM2gDisableEvent()	Disables the reception of an RFM2g event.
RFM2gSendEvent()	Transmits the specified RFM2g interrupt event to one or all other RFM2g node IDs.
RFM2gWaitForEvent()	Blocks the calling process until an occurrence of the specified RFM2g interrupt event is received or a timeout (if enabled) expires.
RFM2gEnableEventCallback()	Enables the interrupt notification for one event on one board.
RFM2gDisableEventCallback()	Disables the interrupt notification for one event on one board.
RFM2gClearEvent()	Flushes all pending events for a specified event type.
RFM2gCancelWaitForEvent()	Cancels any pending RFM2gWaitForEvent() calls for the specified event type.

RFM2gEnableEvent()

RFM2g event interrupts are not enabled by default. The `RFM2gEnableEvent()` function enables an event so an interrupt can be generated on the receiving node. Only RFM2g interrupt events listed in the *EventType* parameter description (see the **Parameter** section below) may be controlled in this way. User applications are notified of received events by using the `RFM2gWaitForEvent()` or `RFM2gEnableEventCallback()` function.

The behavior of `RFM2gEnableEvent()` varies, depending on the following scenarios regarding `RFM2gEnableEvent()`:

Existing Condition(s)	RFM2gEnableEvent() Behavior
The event is disabled.	RFM2G_SUCCESS
The event is enabled.	RFM2G_SUCCESS

Syntax

```
STDRFM2GCALL RFM2gEnableEvent( RFM2GHANDLE rh,
                               RFM2GEVENTTYPE EventType );
```

Parameters

rh Handle to open RFM2g device (I).

EventType Specifies which interrupt event to enable (I).
Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver. RFM2G_BAD_PARAMETER_2 — <i>EventType</i> is invalid. RFM2G_DRIVER_ERROR — Internal driver error. RFM2G_ALREADY_ENABLED — The specified event is already enabled.

Example

The following example code enables user interrupt event 1. Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gEnableEvent(Handle, RFM2GEVENT_INTR1);
```

Related Commands

- RFM2gDisableEvent()
- RFM2gClearEvent()
- RFM2gSendEvent()
- RFM2gWaitForEvent()
- RFM2gCancelWaitForEvent()
- RFM2gEnableEventCallback()
- RFM2gDisableEventCallback()

RFM2gDisableEvent()

The `RFM2gDisableEvent()` function disables the generation of a CPU interrupt when an RFM2g event is received on the current node.

The behavior of `RFM2gDisableEvent()` varies, depending on the following scenarios regarding `RFM2gEnableEvent()`, `RFM2gEnableEventCallback()` and `RFM2gWaitForEvent()`:

Existing Condition(s)	RFM2gDisableEvent() Behavior
The event is not enabled.	RFM2G_SUCCESS
The event is enabled without a pending callback or RFM2gWaitForEvent() .	RFM2G_SUCCESS
The event is enabled and a callback is registered.	RFM2G_SUCCESS — The callback is not affected. The callback will not occur until the event is enabled and received.
The event is enabled with a pending RFM2gWaitForEvent() .	RFM2G_SUCCESS — RFM2gWaitForEvent() remains pending.

NOTE: `RFM2gDisableEvent()` will disable an event *only* if it was enabled using the same handle.

Even if disabled, the RFM2g device continues storing received events in an onboard FIFO until enabled or cleared.

Syntax

```
STDRFM2GCALL RFM2gDisableEvent( RFM2GHANDLE rh,
                                RFM2GEVENTTYPE EventType );
```

Parameters

rh Handle to open RFM2g device (I).

EventType Specifies which interrupt event to disable (I). Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>EventType</i> is invalid.
	RFM2G_DRIVER_ERROR — Internal driver error.

Example

The following example code disables user interrupt 1. Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gDisableEvent(Handle, RFM2GEVENT_INTR1);
```

Related Commands

- RFM2gEnableEvent()
- RFM2gClearEvent()
- RFM2gSendEvent()
- RFM2gWaitForEvent()
- RFM2gCancelWaitForEvent()
- RFM2gEnableEventCallback()
- RFM2gDisableEventCallback()

RFM2gSendEvent()

The `RFM2gSendEvent` function sends an interrupt event and a 32-bit longword value to another node. Five RFM2g interrupt event types are available for an application program to use in signaling events to other RFM2g nodes.

Syntax

```
RFM2G_STATUS RFM2gSendEvent( RFM2GHANDLE rh,
                              RFM2G_NODE ToNode,
                              RFM2GEVENTTYPE EventType,
                              RFM2G_UINT32 ExtendedData );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>ToNode</i>	Who will receive the interrupt event (I) (RFM2G_NODE_ALL or -1 sends the event to all nodes). NOTE: A node cannot send an event to itself.
<i>EventType</i>	The type of interrupt event to send (I). Interrupts correlate to the following event IDs:

Interrupt

Reset Interrupt
Network Interrupt 1
Network Interrupt 2
Network Interrupt 3
Network Interrupt 4

Event ID

RFM2GEVENT_RESET
RFM2GEVENT_INTR1
RFM2GEVENT_INTR2
RFM2GEVENT_INTR3
RFM2GEVENT_INTR4

<i>ExtendedData</i>	User-defined data (I).
---------------------	------------------------

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — Invalid <i>ToNode</i> .
	RFM2G_BAD_PARAMETER_3 — Invalid <i>EventType</i> .
	RFM2G_DRIVER_ERROR — Internal driver error.
	RFM2G_NOT_OPEN — Device is not open.

Example

The following example code sends user interrupt 1. Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
/* Send interrupt event 1 to node 0 with extended data value 0 */  
result = RFM2gSendEvent(Handle, 0x0, RFM2GEVENT_INTR1, 0x0);
```

Related Commands

- RFM2gEnableEvent()
- RFM2gDisableEvent()
- RFM2gClearEvent()
- RFM2gWaitForEvent()
- RFM2gCancelWaitForEvent()
- RFM2gEnableEventCallback()
- RFM2gDisableEventCallback()

RFM2gWaitForEvent()

The `RFM2gWaitForEvent()` function allows an application program to determine that an RFM2g interrupt event has been received. The program blocks until the next RFM2g interrupt event of the requested type has been received, then returns.

The specified event is disabled if one of the following error events occurs, and the `RFM2gEnableEvent()` must be called to re-enable the interrupt:

Error Event ID	Description
<code>RFM2GEVENT_BAD_DATA</code>	Bad Data Interrupt
<code>RFM2GEVENT_RXFIFO_FULL</code>	RX FIFO Full Interrupt
<code>RFM2GEVENT_ROGUE_PKT</code>	Rogue packet detected and removed

NOTE: Ensure that events do not interrupt continuously if they are re-enabled.

The behavior of `RFM2gWaitForEvent()` varies, depending on the following scenarios regarding `RFM2gEnableEvent()`, `RFM2gEnableEventCallback()`, `RFM2gDisableEvent()` and `RFM2gClose()`:

Existing Condition(s)	RFM2gWaitForEvent() Behavior
The event is enabled.	<code>RFM2G_SUCCESS</code> — Event received <code>RFM2G_TIMED_OUT</code> — Event not received before timeout <code>RFM2G_WAIT_EVENT_CANCELLED</code> — User called RFM2gCancelWaitForEvent()
The event is not enabled.	<code>RFM2G_SUCCESS</code> — Event received <code>RFM2G_TIMED_OUT</code> — Event not received before timeout <code>RFM2G_WAIT_EVENT_CANCELLED</code> — User called RFM2gCancelWaitForEvent()
RFM2gDisableEvent() is called for the event.	<code>RFM2G_SUCCESS</code> — Event received <code>RFM2G_TIMED_OUT</code> — Event not received before timeout <code>RFM2G_WAIT_EVENT_CANCELLED</code> — User called RFM2gCancelWaitForEvent()
The event is enabled and a callback is registered.	<code>RFM2G_EVENT_IN_USE</code>
Another thread is pending on RFM2gWaitForEvent() .	<code>RFM2G_EVENT_IN_USE</code>

Syntax

```
RFM2G_STATUS RFM2gWaitForEvent( RFM2GHANDLE rh,
                                RFM2GEVENTINFO *EventInfo );
```

Parameters

rh Handle to open RFM2g device (I).

EventInfo Pointer to RFM2GEVENTINFO structure containing the event type and time, in milliseconds, to wait for the event before returning.

The following is the `rfm2gEventInfo` structure used by the *EventInfo* parameter of `RFM2gWaitForEvent()`:

```
typedef struct rfm2gEventInfo
{
    RFM2G_UINT32   ExtendedInfo; /* Data passed with event          */
    RFM2G_NODE     NodeId;      /* Source Node                    */
    RFM2GEVENTTYPE Event;      /* Event type                     */
    RFM2G_UINT32   Timeout;     /* timeout value to wait for event mSec */
    void *         pDrvSpec;    /* Driver specific                */
} RFM2GEVENTINFO;
```

The `rfm2gEventInfo` structure parameters are:

<i>ExtendedInfo</i>	User data that accompanies an event (O).																		
<i>NodeId</i>	RFM node that sent the event (O).																		
<i>Event</i>	Specifies which interrupt event to wait upon (I). Interrupts correlate to the following event IDs:																		
	<table> <thead> <tr> <th>Interrupt</th> <th>Event ID</th> </tr> </thead> <tbody> <tr> <td>Reset Interrupt</td> <td>RFM2GEVENT_RESET</td> </tr> <tr> <td>Network Interrupt 1</td> <td>RFM2GEVENT_INTR1</td> </tr> <tr> <td>Network Interrupt 2</td> <td>RFM2GEVENT_INTR2</td> </tr> <tr> <td>Network Interrupt 3</td> <td>RFM2GEVENT_INTR3</td> </tr> <tr> <td>Network Interrupt 4 (Init Interrupt)</td> <td>RFM2GEVENT_INTR4</td> </tr> <tr> <td>Bad Data Interrupt</td> <td>RFM2GEVENT_BAD_DATA</td> </tr> <tr> <td>RX FIFO Full Interrupt</td> <td>RFM2GEVENT_RXFIFO_FULL</td> </tr> <tr> <td>Rogue packet detected and removed</td> <td>RFM2GEVENT_ROGUE_PKT</td> </tr> </tbody> </table>	Interrupt	Event ID	Reset Interrupt	RFM2GEVENT_RESET	Network Interrupt 1	RFM2GEVENT_INTR1	Network Interrupt 2	RFM2GEVENT_INTR2	Network Interrupt 3	RFM2GEVENT_INTR3	Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4	Bad Data Interrupt	RFM2GEVENT_BAD_DATA	RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL	Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT
Interrupt	Event ID																		
Reset Interrupt	RFM2GEVENT_RESET																		
Network Interrupt 1	RFM2GEVENT_INTR1																		
Network Interrupt 2	RFM2GEVENT_INTR2																		
Network Interrupt 3	RFM2GEVENT_INTR3																		
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4																		
Bad Data Interrupt	RFM2GEVENT_BAD_DATA																		
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL																		
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT																		
<i>Timeout</i>	Indicates the timeout, in milliseconds, to wait for the event before returning (I). Non-zero values use a timeout, as determined by the following criteria:																		
	<table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>RFM2G_INFINITE_TIMEOUT</td> <td>Wait forever for event to occur.</td> </tr> <tr> <td>RFM2G_NOWAIT</td> <td>Do not wait for event to occur.</td> </tr> <tr> <td>[value]</td> <td>Number of milliseconds to wait for event to occur.</td> </tr> </tbody> </table>	Value	Description	RFM2G_INFINITE_TIMEOUT	Wait forever for event to occur.	RFM2G_NOWAIT	Do not wait for event to occur.	[value]	Number of milliseconds to wait for event to occur.										
Value	Description																		
RFM2G_INFINITE_TIMEOUT	Wait forever for event to occur.																		
RFM2G_NOWAIT	Do not wait for event to occur.																		
[value]	Number of milliseconds to wait for event to occur.																		
<i>pDrvSpec</i>	Driver-specific data that accompanies and event (I). Refer to your driver-specific manual for more information.																		

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver. RFM2G_BAD_PARAMETER_2 — <i>EventInfo</i> is NULL or <i>EventInfo</i> event has invalid event type. RFM2G_EVENT_IN_USE — Event is already being waited on. RFM2G_TIMED_OUT — Timed out waiting for event. RFM2G_WAIT_EVENT_CANCELLED — <code>RFM2gCancelWaitForEvent()</code> called for this event. RFM2G_DRIVER_ERROR — Internal driver error.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2GEVENTINFO info;

info.EventType = RFM2GEVENT_INT1;
info.mSecToWait = 10000; /* Wait 10 seconds */

result = RFM2gWaitForEvent(Handle, &info);
```

Related Commands

- `RFM2gEnableEvent()`
- `RFM2gDisableEvent()`
- `RFM2gClearEvent()`
- `RFM2gSendEvent()`
- `RFM2gCancelWaitForEvent()`
- `RFM2gEnableEventCallback()`
- `RFM2gDisableEventCallback()`

RFM2gEnableEventCallback()

The `RFM2gEnableEventCallback()` function enables the interrupt notification for one event on one board.

The specified event is disabled if one of the following error events occurs, and the `RFM2gEnableEvent()` must be called to re-enable the interrupt:

Error Event ID	Description
<code>RFM2GEVENT_BAD_DATA</code>	Bad Data Interrupt
<code>RFM2GEVENT_RXFIFO_FULL</code>	RX FIFO Full Interrupt
<code>RFM2GEVENT_ROGUE_PKT</code>	Rogue packet detected and removed

NOTE: Ensure that events do not interrupt continuously if they are re-enabled.

The behavior of `RFM2gEnableEventCallback()` varies, depending on the following scenarios regarding `RFM2gEnableEvent()` and `RFM2gWaitForEvent()`:

Existing Condition(s)	RFM2gDisableEvent() Behavior
The event is enabled and a callback is registered.	<code>RFM2G_EVENT_IN_USE</code>
The event is enabled without an enabled callback.	<code>RFM2G_SUCCESS</code>
The event is enabled with a pending RFM2gWaitForEvent() .	<code>RFM2G_EVENT_IN_USE</code>

Syntax

```

STDRFM2GCALL RFM2gEnableEventCallback( RFM2GHANDLE rh,
                                        RFM2GEVENTTYPE EventType,
                                        RFM2G_EVENT_FUNCPTR pEventFunc );

void rfmSampleFunc( RFM2GHANDLE rh,
                   RFM2GEVENTINFO EventInfo )

```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).																		
<i>EventType</i>	Specifies which interrupt event to disable (I). Interrupts correlate to the following event IDs:																		
	<table> <thead> <tr> <th>Interrupt</th> <th>Event ID</th> </tr> </thead> <tbody> <tr> <td>Reset Interrupt</td> <td>RFM2GEVENT_RESET</td> </tr> <tr> <td>Network Interrupt 1</td> <td>RFM2GEVENT_INTR1</td> </tr> <tr> <td>Network Interrupt 2</td> <td>RFM2GEVENT_INTR2</td> </tr> <tr> <td>Network Interrupt 3</td> <td>RFM2GEVENT_INTR3</td> </tr> <tr> <td>Network Interrupt 4 (Init Interrupt)</td> <td>RFM2GEVENT_INTR4</td> </tr> <tr> <td>Bad Data Interrupt</td> <td>RFM2GEVENT_BAD_DATA</td> </tr> <tr> <td>RX FIFO Full Interrupt</td> <td>RFM2GEVENT_RXFIFO_FULL</td> </tr> <tr> <td>Rogue packet detected and removed</td> <td>RFM2GEVENT_ROGUE_PKT</td> </tr> </tbody> </table>	Interrupt	Event ID	Reset Interrupt	RFM2GEVENT_RESET	Network Interrupt 1	RFM2GEVENT_INTR1	Network Interrupt 2	RFM2GEVENT_INTR2	Network Interrupt 3	RFM2GEVENT_INTR3	Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4	Bad Data Interrupt	RFM2GEVENT_BAD_DATA	RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL	Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT
Interrupt	Event ID																		
Reset Interrupt	RFM2GEVENT_RESET																		
Network Interrupt 1	RFM2GEVENT_INTR1																		
Network Interrupt 2	RFM2GEVENT_INTR2																		
Network Interrupt 3	RFM2GEVENT_INTR3																		
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4																		
Bad Data Interrupt	RFM2GEVENT_BAD_DATA																		
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL																		
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT																		
<i>pEventFunc</i>	The address of the function to be called when the event occurs (I).																		

Return Values

Success	RFM2G_SUCCESS
Failure	<p>RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.</p> <p>RFM2G_OS_ERROR — Operating system (OS) returned an error.</p> <p>RFM2G_NOT_OPEN — Device is not open.</p> <p>RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.</p> <p>RFM2G_BAD_PARAMETER_2 — Invalid <i>EventType</i>.</p> <p>RFM2G_BAD_PARAMETER_3 — <i>pEventFunc</i> is NULL.</p> <p>RFM2G_EVENT_IN_USE — Event is already being waited on.</p> <p>RFM2G_DRIVER_ERROR — Internal driver error.</p> <p>RFM2G_WAIT_EVENT_CANCELLED — Wait for event cancelled.</p>

Example

The following example code registers the function `MyEventCallback`, which is called when the hardware returns the `RFM2GEVENT_INTR1` interrupt. Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gEnableEventCallback(Handle, RFM2GEVENT_INTR1, MyEventCallback);
```

Related Commands

- `RFM2gEnableEvent()`
- `RFM2gDisableEvent()`
- `RFM2gClearEvent()`
- `RFM2gSendEvent()`
- `RFM2gWaitForEvent()`
- `RFM2gCancelWaitForEvent()`
- `RFM2gDisableEventCallback()`

RFM2gDisableEventCallback()

The `RFM2gDisableEventCallback()` function disables interrupt notification for one event by this handle.

The specified event is disabled if one of the following error events occurs, and the `RFM2gEnableEvent()` must be called to re-enable the interrupt:

Error Event ID	Description
RFM2GEVENT_BAD_DATA	Bad Data Interrupt
RFM2GEVENT_RXFIFO_FULL	RX FIFO Full Interrupt
RFM2GEVENT_ROGUE_PKT	Rogue packet detected and removed

NOTE: Ensure that events do not interrupt continuously if they are re-enabled.

The behavior of `RFM2gDisableEventCallback()` varies, depending on the following scenarios regarding `RFM2gEnableEvent()` and `RFM2gEnableEventCallback()`:

Existing Condition(s)	RFM2gDisableEvent() Behavior
The event is enabled and a callback is registered.	RFM2G_SUCCESS — The callback is terminated without calling the user function.
The event is enabled without a registered callback.	RFM2G_SUCCESS

Syntax

```
STDRFM2GCALL RFM2gDisableEventCallback( RFM2GHANDLE rh,
                                          RFM2GEVENTTYPE EventType );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>EventType</i>	Specifies which interrupt event to disable (I). Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — Invalid <i>EventType</i> .
	RFM2G_EVENT_NOT_IN_USE — Event not registered for callback.
	RFM2G_EVENT_NOT_IN_USE — No callback was set up for the specified event.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gDisableEventCallback(Handle, RFM2GEVENT_INTR1);
```

Related Commands

- RFM2gEnableEvent()
- RFM2gDisableEvent()
- RFM2gClearEvent()
- RFM2gSendEvent()
- RFM2gWaitForEvent()
- RFM2gCancelWaitForEvent()
- RFM2gEnableEventCallback()

RFM2gClearEvent()

The `RFM2gClearEvent()` function clears any or all pending interrupt events from a specified event FIFO.

The pending event is cleared and disabled if one of the following error events occurs, and the `RFM2gEnableEvent()` must be called to re-enable the interrupt.

Error Event ID	Description
RFM2GEVENT_BAD_DATA	Bad Data Interrupt
RFM2GEVENT_RXFIFO_FULL	RX FIFO Full Interrupt
RFM2GEVENT_ROGUE_PKT	Rogue packet detected and removed

Syntax

```
STDRFM2GCALL RFM2gClearEvent( RFM2GHANDLE rh,
                               RFM2GEVENTTYPE EventType );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>EventType</i>	The event FIFO to clear (I). Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT
All interrupts	RFM2GEVENT_LAST

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — Invalid <i>EventType</i> .
	RFM2G_DRIVER_ERROR — Internal driver error.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gClearEvent(Handle, RFM2GEVENT_INTR1);
```

Related Commands

- RFM2gEnableEvent()
- RFM2gDisableEvent()
- RFM2gSendEvent()
- RFM2gWaitForEvent()
- RFM2gCancelWaitForEvent()
- RFM2gEnableEventCallback()
- RFM2gDisableEventCallback()

RFM2gCancelWaitForEvent()

The `RFM2gCancelWaitForEvent()` function cancels any pending `RFM2gWaitForEvent()` calls on an event by this handle. If a callback has been registered to the specified event, the callback will be cancelled.

NOTE: A canceled `RFM2gWaitForEvent()` call returns a value of `RFM2G_WAIT_EVENT_CANCELLED`.

Syntax

```
STDRFM2GCALL RFM2gCancelWaitForEvent( RFM2GHANDLE rh,
                                       RFM2GEVENTTYPE EventType );
```

Parameters

rh Handle to open RFM2g device (I).

EventType Specifies which interrupt event to cancel (I).
Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt 1	RFM2GEVENT_INTR1
Network Interrupt 2	RFM2GEVENT_INTR2
Network Interrupt 3	RFM2GEVENT_INTR3
Network Interrupt 4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue packet detected and removed	RFM2GEVENT_ROGUE_PKT

Return Values

Success `RFM2G_SUCCESS`

Failure `RFM2G_NULL_DESCRIPTOR` — *rh* is NULL.

`RFM2G_OS_ERROR` — Operating system (OS) returned an error.

`RFM2G_NOT_OPEN` — Device is not open.

`RFM2G_NOT_IMPLEMENTED` — API function is not implemented in the driver.

`RFM2G_EVENT_NOT_IN_USE` — No wait is pending for the event.

`RFM2G_BAD_PARAMETER_2` — Invalid *EventType*.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gCancelWaitForEvent(Handle, RFM2GEVENT_INTR1);
```

Related Commands

- RFM2gEnableEvent()
- RFM2gDisableEvent()
- RFM2gClearEvent()
- RFM2gSendEvent()
- RFM2gWaitForEvent()
- RFM2gEnableEventCallback()
- RFM2gDisableEventCallback()

RFM2g Utility API Functions

The following API functions in the rfm2g_api.h file are utility functions that are provided by the RFM2g driver.

Table 1-7 RFM2g Utility API Functions

API Function	Description
RFM2gErrorMsg()	Returns a pointer to a text string describing an error code.
RFM2gGetLed()	Retrieves the current on/off state of the Reflective Memory board's STATUS LED.
RFM2gSetLed()	Sets the on/off state of the Reflective Memory board's STATUS LED.
RFM2gCheckRingCont()	Returns the fiber ring continuity through nodes.
RFM2gGetDebugFlags()	Retrieves a copy of all RFM2g device driver debug control flags.
RFM2gSetDebugFlags()	Sets or clears the device driver debug control flags.

RFM2gErrorMsg()

The `RFM2gErrorMsg()` function returns a pointer to a text string describing a runtime error.

Runtime errors are detected by the API. A text description of the error is output to the screen if debug mode is enabled and returns the string through the return pointer.

Syntax

```
char* RFM2gErrorMsg( RFM2G_STATUS ErrorCode );
```

Parameters

ErrorCode Return code from an API function (I).

Return Values

`char*` The address pointing to a character string that describes the error parameter. The following string is returned if an invalid *ErrorCode* is passed:

```
"UNKNOWN RFM2G ERROR [%d]"  
where [%d] is the ErrorCode value.
```

Failure A NULL pointer.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2G_SUCCESS;  
printf("RFM2g Error Message : %s\n", RFM2gErrorMsg(result));
```

RFM2gGetLed()

The `RFM2gGetLed()` function retrieves the current on/off state of the Reflective Memory board's STATUS LED.

Syntax

```
STDRFM2GCALL RFM2gGetLed( RFM2GHANDLE rh,  
                           RFM2G_BOOL *Led );
```

Parameters

<i>rh</i>	Handle to open RFM2g device (I).
<i>Led</i>	Pointer to where the state of the STATUS LED is written (RFM2G_ON when the LED is on, or RFM2G_OFF when the LED is off) (O).

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL. RFM2G_OS_ERROR — Operating system (OS) returned an error. RFM2G_NOT_OPEN — Device is not open. RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver. RFM2G_BAD_PARAMETER_2 — <i>Led</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_BOOL Led;  
result = RFM2gGetLed(Handle, &Led);
```

Related Commands

- RFM2gSetLed()

RFM2gSetLed()

The `RFM2gSetLed()` function sets the current on/off state of the Reflective Memory board's STATUS LED.

Syntax

```
STDRFM2GCALL RFM2gSetLed( RFM2GHANDLE rh,  
                           RFM2G_BOOL Led );
```

Parameters

rh Handle to open RFM2g device (I).
Led The state of the Fail LED: RFM2G_FALSE=>OFF, RFM2G_TRUE=>ON (I).

Return Values

Success RFM2G_SUCCESS
Failure RFM2G_NULL_DESCRIPTOR — *rh* is NULL.
 RFM2G_OS_ERROR — Operating system (OS) returned an error.
 RFM2G_NOT_OPEN — Device is not open.
 RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gSetLed(Handle, RFM2G_TRUE);
```

Related Commands

- RFM2gGetLed()

RFM2gCheckRingCont()

The `RFM2gCheckRingCont()` function is a diagnostic aid that shows whether or not the fiber ring is continuous through all nodes in the ring. No data is written to the Reflective Memory locations.

Syntax

```
STDRFM2GCALL RFM2gCheckRingCont( RFM2GHANDLE rh );
```

Parameters

rh Handle to currently opened RFM2g device (I).

Return Values

Success	RFM2G_SUCCESS — Link is closed and intact.
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_LINK_TEST_FAIL — Link is open.
	RFM2G_DRIVER_ERROR — Internal driver error.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
result = RFM2gCheckRingCont(Handle);
if(result == RFM2G_SUCCESS)
{
    printf("Ring Intact");
}
```

RFM2gGetDebugFlags()

NOTE: Application programs should not use this function unless directed to do so by VMIC support personnel.

The `RFM2gGetDebugFlags()` function returns a copy of the current setting of the debug flags of the device driver. The RFM2g device driver can generate debug messages by checking a bit in the driver's debug flags variable.

A maximum of 32 debug message classes are possible. Each debug message class is assigned to an individual bit within this 32-bit control word. A nonzero bit implies that the corresponding debug message class can be generated by the RFM2g device driver.

Syntax

```
STDRFM2GCALL RFM2gGetDebugFlags( RFM2GHANDLE rh,  
                                  RFM2G_UNIT32 *Flags )
```

Parameters

rh Handle to currently opened RFM2g device (I).

Flags Pointer to where debug flags are written (O). The following are possible debug flags:

Debug Flag	Description
RFM2G_DBERROR	Report critical errors
RFM2G_DBINIT	Trace device probing and search
RFM2G_DBINTR	Trace interrupt service
RFM2G_DBIOCTL	Trace ioctl(2) system calls
RFM2G_DBMMAP	Trace mmap(2) system calls
RFM2G_DBOPEN	Trace open(2) system calls
RFM2G_DBCLOSE	Trace close(2) system calls
RFM2G_DBREAD	Trace read(2) system calls
RFM2G_DBWRITE	Trace write(2) system calls
RFM2G_DBPEEK	Trace peeks
RFM2G_DBPOKE	Trace pokes
RFM2G_DBSTRAT	Trace read/write strategy
RFM2G_DBTIMER	Trace interval timer
RFM2G_DBTRACE	Trace subroutine entry/exit
RFM2G_DBMUTEX	Trace synchronization and locking
RFM2G_DBINTR_NOT	Trace non-RFM interrupt service
RFM2G_DBSLOW	Let syslogd get the message
RFM2G_DBMINPHYS	Trace minphys limits

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.
	RFM2G_BAD_PARAMETER_2 — <i>Flags</i> is NULL.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UINT32 lFlag;  
  
result = RFM2gGetDebugFlags(Handle, &lFlag);
```

Related Commands

- RFM2gSetDebugFlags()

RFM2gSetDebugFlags()

NOTE: Application programs should not use this function unless directed to do so by VMIC support personnel.

Each possible RFM2g device driver debug output message is assigned to a debug message class. The device driver will generate messages of that class if the corresponding flag bit is set in the control word. The `RFM2gSetDebugFlags()` function allows an application program to set that control word (i.e. this command sets the driver's debug flags).

Application programs do not normally need to alter the setting of the debug message output control word. In some cases, enabling debug flags can severely impact the performance of the host system.

Operation

The `RFM2gSetDebugFlags()` function specifies the new debug message control word. The change is effective immediately.

Syntax

```
STDRFM2GCALL RFM2gSetDebugFlags( RFM2GHANDLE rh,  
                                  RFM2G_UNIT32 Flags );
```

Parameters

rh Handle to currently opened RFM2g device (I).

Flags Debug flags (I). The following are possible debug flags to set:

Debug Flag	Description
RFM2G_DBERROR	Report critical errors
RFM2G_DBINIT	Trace device probing and search
RFM2G_DBINTR	Trace interrupt service
RFM2G_DBIOCTL	Trace ioctl(2) system calls
RFM2G_DBMMAP	Trace mmap(2) system calls
RFM2G_DBOPEN	Trace open(2) system calls
RFM2G_DBCLOSE	Trace close(2) system calls
RFM2G_DBREAD	Trace read(2) system calls
RFM2G_DBWRITE	Trace write(2) system calls
RFM2G_DBPEEK	Trace peeks
RFM2G_DBPOKE	Trace pokes
RFM2G_DBSTRAT	Trace read/write strategy
RFM2G_DBTIMER	Trace interval timer
RFM2G_DBTRACE	Trace subroutine entry/exit
RFM2G_DBMUTEX	Trace synchronization and locking
RFM2G_DBINTR_NOT	Trace non-RFM interrupt service
RFM2G_DBSLOW	Let syslogd get the message
RFM2G_DBMINPHYS	Trace minphys limits

Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <i>rh</i> is NULL.
	RFM2G_OS_ERROR — Operating system (OS) returned an error.
	RFM2G_NOT_OPEN — Device is not open.
	RFM2G_NOT_IMPLEMENTED — API function is not implemented in the driver.

Example

Use the following code by inserting it into the example routine in "Routine Code for Use with API Function Examples" on page 15:

```
RFM2G_UNIT32 lFlag = RFM2G_DBERROR;  
  
result = RFM2gSetDebugFlags(Handle, lFlag);
```

Related Commands

- RFM2gGetDebugFlags()

rfm2g_util.c *Utility Program*

Contents

RFM2g Command Line Interpreter	92
Utility Commands	95
Troubleshooting the rfm2g_util.c Command Line Interpreter	146

Introduction

The RFM2g driver is delivered with a command line interpreter (**rfm2g_util.c**) that enables you to exercise various RFM2g commands by entering commands at the standard input (usually the console keyboard). The utility provides a convenient method of accessing most of the services provided by the driver.

RFM2g Command Line Interpreter

The `rfm2g_util.c` command line interpreter program is a utility that enables you to view or change the contents of a RFM2g board and provides an easy method of operating the device driver.

No programming is required to use the command line interpreter program. Instead, simple ASCII text commands are used. A single command may be given on a command line when the command line interpreter program is running, or multiple commands may be read from the standard input file.

Reflective Memory can be displayed or changed. RFM2g interrupt events may be sent or received. The program also allows asynchronous notification of RFM interrupt events.

The command line interpreter program is coded in the ANSI dialect of the C language. The source code for the program is provided to serve as an example of how to use the language bindings provided by the driver and the DLL or library.

Using the Command Line Interpreter

The command line interpreter program is not case-sensitive, so uppercase and lowercase characters may be intermixed at will and will not affect execution. In addition, the command line interpreter program attempts to reduce the amount of typing that may be necessary. Whenever a keyword is required (such as a command name), only the first few characters need to be typed to uniquely identify the command. If you do not type enough characters for the command line interpreter to select a single command, all of the possible commands will be listed, along with another command prompt.

For example, five commands in the command line interpreter program begin with the character *d* (the **devname**, **disableevent**, **disablecallback**, **dllversion** and **driverversion** commands). However, the first two letters of the **devname** command will specify it as a unique command to the command line interpreter. So, instead of having to type:

```
devname
```

only the first two letters needs to be typed:

```
de
```

If the typed characters aren't enough to uniquely identify the command, the command line interpreter outputs an error message and shows a table of the possible names. Since there are five commands that begin with the letter *d*, the typed input:

```
d
```

produces this message:

```
d
Ambiguous command 'd'. This could be
devname
disableevent
disablecallback
dllversion
driverversion
```

Notes On Entering Numbers

Whenever the command line interpreter expects a number, any C-language style number may be used. If it begins with `0x`, a hexadecimal value follows; if it begins with a `0`, an octal value follows; otherwise, the number is assumed to be decimal.

Notes On Device Numbers

When the `rfm2g_util.c` utility is started, you will be prompted for a device number. Refer to your driver-specific manual for the correct RFM2g device number to use.

Once a device number has been entered, it displays next to the utility prompt. In the following example, board number 0 has been entered:

```
UTIL0 >
```

Command Line Interpreter Example

The following is an example workflow illustrating how the command line interpreter program can be used. Examples are also provided with the descriptions of individual commands within the command line interpreter.

1. Start the utility program by following the directions in your driver-specific manual.

The following is displayed in the console window:

```
PCI RFM2g Commandline Diagnostic Utility
```

```
Please enter device number:
```

2. Type an RFM2g device number and press <ENTER>.

NOTES: Refer to your driver-specific manual for the correct RFM2g device number to use.

A prompt displays that includes the device number. For example, if you entered 0:

```
UTIL0 >
```

3. View the contents of RFM2g memory at offset 0x0 by entering:

```
peek8 0x0
```

4. Observe the output.
5. Exit the command line interpreter program by entering:

```
quit
```

The following prompt displays:

```
Exit? (y/n):
```

6. Enter **y** to confirm.

Utility Commands

The commands which are implemented in the command line interpreter program are described and demonstrated in this section.

The table below lists each command included in the command line interpreter and a short description of each.

Table 2-1 RFM2g Driver Commands

Command	Description
boardid	Returns the board ID of the currently opened RFM2g device.
cancelwait	Cancels any pending calls for a specified event type.
checkring	Returns the fiber ring continuity through all nodes in a ring.
clearevent	Clear any pending events for a specified event.
config	Display RFM2g board configuration information.
devname	Returns the device name associated with a RFM2g handle.
disableevent	Disables the reception of a specified RFM2g interrupt event.
disablecallback	Disables the interrupt notification for a specified event notification.
dllversion	Returns the DLL or library version.
driverversion	Returns the RFM2g device driver version.
drvspecific	Enter a driver-specific menu.
dump	Peek and display an area of Reflective Memory
enableevent	Enable the receiving of a specified RFM2g event.
enablecallback	Enables the interrupt notification for a specified event.
errmsg	Prints a text string describing an error code.
exit	Terminate the command line interpreter program.
first	Returns the first available RFM2g offset.
getdebug	Retrieves a copy of all RFM2g device driver debug message class control flags.
getdmabyteswap	Returns the state of DMA (Direct Memory Access) byte swapping hardware.
getled	Retrieves the current on/off state of the Reflective Memory board's STATUS LED.
getpiobyteswap	Returns the state of PIO (Programmed IO) byte swapping hardware.
getthreshold	Returns the current DMA threshold value.

Table 2-1 RFM2g Driver Commands (Continued)

Command	Description
help	Display command help.
mapuser	Retrieves RFM2g memory information or maps RFM2g memory to the user space.
memop	Fill or verify an area of Reflective Memory.
nodeid	Returns the RFM2g device node ID.
peek8, peek16, peek32 and peek64	Display RFM2g contents.
performancetest	Display the speed of system reads and writes.
poke8, poke16, poke32 and poke64	Set RFM2g contents.
quit	Terminate the command line interpreter program.
read	Reads memory contents starting at the specified offset in Reflective Memory and dumps data read to output.
repeat	Repeat a command line interpreter command.
return	Exit a driver-specific sub-menu.
send	Sends an RFM2g interrupt event to another node.
setdebug	Sets the driver's debug display word (i.e. debug flags).
setdmabyteswap	Sets the on/off state of DMA (Direct Memory Access) byte swapping hardware.
setled	Sets the current on/off state of the Reflective Memory board's STATUS LED.
setpiobyteswap	Sets the on/off state of PIO (Programmed IO) byte swapping hardware.
setthreshold	Sets the transfer size at which reads and writes will use DMA.
size	Returns the total amount of virtual memory space available on the RFM2g device.
unmapuser	Unmaps RFM2g buffer memory from the user space.

Table 2-1 RFM2g Driver Commands (Continued)

Command	Description
wait	Blocks the calling process until an occurrence of a RFM2g interrupt event is received or a timeout expires.
write	Writes a value starting at the specified offset in Reflective Memory.

boardid

The **boardid** command returns the RFM2g interface model type. Each RFM2g interface model type is uniquely identified by a numeric value assigned by VMIC and recorded as a fixed constant in a RFM2g hardware register. The driver and support library read this value when the device is opened. The utility calls `RFM2gBoardID()` to obtain the RFM2g board ID.

Syntax

```
boardid
```

Example

```
UTIL0 > boardid
Board ID           0x65      (VMIPCI-5565)
UTIL0 >
```

cancelwait

The **cancelwait** command cancels any pending calls on an event. If a callback has been registered to the specified event, the callback will be cancelled. The utility calls `RFM2gCancelWaitForEvent()` to cancel the pending calls.

Syntax

```
cancelwait event
```

Parameters

event Specifies which interrupt event to cancel (I).
Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	0
Network Interrupt 1	1
Network Interrupt 2	2
Network Interrupt 3	3
Network Interrupt 4 (Init Interrupt)	4
Bad Data Interrupt	5
RX FIFO Full Interrupt	6
Rogue Packet Detected and Removed	7

Example

```
UTIL0 > cancelwait 1  
RFM2gWaitForEvent has been canceled for the  
"NETWORK INT 1" event.  
UTIL0 >
```

checking

The **checking** command is a diagnostic aid that shows whether or not the fiber ring is continuous through all nodes in the ring. No data is written to the Reflective Memory locations. The utility calls `RFM2gCheckRingCont()` to obtain the RFM2g ring status.

Syntax

```
checking
```

Example

```
UTIL0 > checking  
The Reflective Memory link is intact  
UTIL0 >
```

clearevent

The **clearevent** command clears all pending interrupt events for a specified event. The utility calls `RFM2gClearEvent()` function.

Syntax

```
clearevent event
```

Parameters

event The event FIFO to clear (I), which is one of the following:

Number	Event to Clear
0	RESET
1	NETWORK INT 1
2	NETWORK INT 2
3	NETWORK INT 3
4	NETWORK INT 4
5	BAD DATA
6	FIFO FULL
7	ROGUE PACKET
8	ALL EVENTS

Example

```
UTIL0 > clearevent 0
The "RESET" interrupt event was flushed.
UTIL0 >
```

config

The **config** command will display the values of members in the RFM2GCONFIG structure. The utility calls `RFM2gGetConfig()` to obtain the RFM2GCONFIG structure.

Syntax

```
config
```

Example

```
UTIL0 > config
  Driver Part Number      "VMISFT-RFM2G-ABC-037"
  Driver Version          "RELEASE 2.00"
  Device Name             "RFM2G_0"
  Board Instance          0
  Board ID                0x65
  Node ID                 0x01
  Installed Memory        134217728d (0x08000000)
  Board Revision          0x04
  PLX Revision            0xAD
UTIL0 >
```

devname

The **devname** command displays a string containing the first 64 characters of the device name associated with a RFM2g file handle. The utility calls `RFM2gDeviceName()` to obtain the RFM2g device name.

Syntax

```
devname
```

Example

```
UTIL0 > devname
Device Name:          "rfm2g_0"
UTIL0 >
```

disableevent

The **disableevent** command disables the reception of an RFM2g event. The utility calls `RFM2gDisableEvent()` to disable the event.

Syntax

```
disableevent event
```

Parameters

event Specifies which interrupt event to disable (I). Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	0
Network Interrupt 1	1
Network Interrupt 2	2
Network Interrupt 3	3
Network Interrupt 4 (Init Interrupt)	4
Bad Data Interrupt	5
RX FIFO Full Interrupt	6
Rogue Packet Detected and Removed	7

Example

```
UTIL0 > disableevent 0
Interrupt event "RESET" is disabled.
UTIL0 >
```

disablecallback

The **disablecallback** command disables event notification for a specified event. The utility calls `RFM2gDisableEventCallback()` to disable event notification.

Syntax

```
disablecallback event
```

Parameters

event Specifies which event notification to disable (I). Events correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	0
Network Interrupt 1	1
Network Interrupt 2	2
Network Interrupt 3	3
Network Interrupt 4 (Init Interrupt)	4
Bad Data Interrupt	5
RX FIFO Full Interrupt	6
Rogue Packet Detected and Removed	7

Example

```
UTIL0 > disableevent 1  
Interrupt event "NETWORK INT 1" is disabled.  
UTIL0 >
```

dllversion

The **dllversion** command displays an ASCII string showing the version of the DLL or API library. This string contains the production release level of the library and is unique between different versions of the API library. The utility calls `RFM2gDllVersion()` to return the library version.

Syntax

```
dllversion
```

Example

```
UTIL0 > dllversion
      Dll Version:           "R01.00"
UTIL0 >
```

driverversion

The **driverversion** command displays an ASCII string showing the VMIC production release version of the underlying RFM2g device driver. The utility calls `RFM2gDriverVersion()` to return the driver version.

Syntax

```
driverversion
```

Example

```
UTIL0 > driverversion
Driver Version:          "R01.00"
UTIL0 >
```

drvspecific

The **drvspecific** command enables the use of the driver-specific sub-menu commands provided in addition to the common commands discussed in this chapter. Refer to your driver-specific manual for information on commands specific to your RFM2g driver.

Syntax

```
drvspecific
```

Examples

To access driver-specific commands:

```
UTIL0 > drvspecific
Welcome to the driver specific menu
UTILDRVSPEC0 >
```

To display a list of driver-specific commands:

```
UTILDRVSPEC0 > help

COMMAND          PARAMETERS
-----
help             [command]
repeat          [-p] count cmd [arg...]
return
UTILDRVSPEC0 >
```

To exit the driver-specific commands:

```
UTILDRVSPEC0 > return
Welcome to the main menu
UTIL0 >
```

dump

The **dump** command enables the user to peek and display an area of Reflective Memory. This utility calls `RFM2gPeek8()`, `RFM2gPeek16()`, `RFM2gPeek32()` or `RFM2gPeek64()`.

NOTE: See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

Syntax

```
dump offset width length
```

Parameters

offset Width-aligned offset in Reflective Memory at which to begin the peek and display (I). Valid offset values are 0x0 to 0x3FFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFF for 128 Mbyte cards.

width Indicates access width in bits, which is one of the following (I):

Value	Description
1	8-bit byte
2	16-bit word
4	32-bit longword
8	64-bit longword

length Number of *width* units to peek and display (I), which is determined using the formula $[buffer\ size] / width$. For example, the *length* of a buffer size of 1024 in 32-bit longwords is 256 ($1024 / 4 = 256$).

Width Bit	Maximum Length (Dec/Hex) for 128 Mbyte Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit long	16777216 (0x1000000)

Example

```
UTIL0 > dump 0 8 4

                                0                               1
0x00000000: 0123456789ABCDEF 0123456789ABCDEF |. #Eg. #Eg .....|
0x00000010: 0123456789ABCDEF 0123456789ABCDEF |. #Eg. #Eg .....|

UTIL0 >
```

enableevent

RFM2g event interrupts are not enabled by default. The **enableevent** command enables a specific RFM event so a system interrupt can be generated on the receiving node. The utility calls `RFM2gEnableEvent()` to enable the RFM event.

Syntax

```
enableevent event
```

Parameters

event The interrupt event to enable. Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	0
Network Interrupt 1	1
Network Interrupt 2	2
Network Interrupt 3	3
Network Interrupt 4 (Init Interrupt)	4
Bad Data Interrupt	5
RX FIFO Full Interrupt	6
Rogue Packet Detected and Removed	7

Example

```
UTIL0 > enableevent 0
Interrupt event "RESET" is enabled.
UTIL0 >
```

enablecallback

The **enablecallback** command enables the interrupt notification for one event on one board.

A message is returned to the console window each time an event call successfully occurs using this command. For example, if four callbacks have been previously performed and a new callback is made from RFM2GEVENT_INTR3, the following displays in the console window:

```
EventCallback: Counter = 5
node 2 Received "RFM2GEVENT_INTR3" interrupt from node 0
```

Extended information for a value can also be displayed. For example:

```
Asynchronous Event Notification has been enabled for the
"NETWORK INT 1" event.
```

The utility calls `RFM2gEnableEventCallback()` to enable interrupt notification.

Syntax

```
enablecallback event
```

Parameters

event Specifies which interrupt notification to enable (I). Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	0
Network Interrupt 1	1
Network Interrupt 2	2
Network Interrupt 3	3
Network Interrupt 4 (Init Interrupt)	4
Bad Data Interrupt	5
RX FIFO Full Interrupt	6
Rogue Packet Detected and Removed	7

Example

```
UTIL0 > enablecallback 1
Asynchronous Event Notification has been enabled for the
"NETWORK INT 1" event.
UTIL0 >
```

errormsg

The **errormsg** command prints a text string describing a runtime error.

Runtime errors are returned by the API functions. The utility calls `RFM2gErrorMsg()` to obtain the error code pointer.

Syntax

```
errormsg ErrorCode
```

Parameters

ErrorCode Return code from an API function (I).

Example

```
UTIL0 > errormsg 0  
ErrorCode = 0, Msg = No current error  
UTIL0 >
```

exit

The **exit** command terminates the command line interpreter program.

Syntax

```
exit
```

Example

```
UTIL0 > exit  
Exit? (y/n):y
```

first

The **first** command displays the first RFM2g offset available for use by an application program. The utility calls `RFM2gFirst()` to return the first available RFM2g offset.

Syntax

```
first
```

Example

```
UTIL0 > first
First                               0x00000000
```

getdebug

NOTE: Users should not use this command unless directed to do so by VMIC support personnel.

The **getdebug** command displays a copy of the current setting of the debug flags of the device driver. The RFM2g device driver can generate debug messages by checking a bit in the driver's debug flags variable.

A maximum of 32 debug message classes are possible. Each debug message class is assigned to an individual bit within this 32-bit control word. A nonzero (0) bit implies that the corresponding debug message class can be generated by the RFM2g device driver. The utility calls `RFM2gGetDebugFlags()` to retrieve debug control flags.

Syntax

```
getdebug
```

Example

```
UTIL0 > getdebug
Current Debug Flags: 0x00000000
UTIL0 >
```

getdmabyteswap

The **getdmabyteswap** command returns the state of the DMA (Direct Memory Access) byte swapping hardware. The utility calls `RFM2gGetDMAByteSwap()` to return the DMA byte swapping state.

Syntax

```
getdmabyteswap
```

Example

```
UTIL0 > getdmabyteswap  
The Reflective Memory board's DMA Byte Swap is ON.  
UTIL0 >
```

getled

Every RFM2g interface board has a STATUS LED which is turned on whenever the RFM2g device is reset and turned off by the RFM2g device driver during initialization. When the RFM2g device driver is unloaded, the STATUS LED is turned on again.

The **getled** command displays the current on/off state of the Reflective Memory board's STATUS LED. The utility calls `RFM2gGetLed()` to retrieve the STATUS LED state.

Syntax

```
getled
```

Example

```
UTIL0 > getled
The Reflective Memory board's Status LED is OFF.
UTIL0 >
```

getpiobyteswap

The **getpiobyteswap** command displays the state of PIO (Programmed IO) byte swapping hardware. The utility calls `RFM2gGetPIOByteSwap()` to return the PIO byte swapping state.

Syntax

```
getpiobyteswap
```

Example

```
UTIL0 > getpiobyteswap  
The Reflective Memory board's PIO Byte Swap is ON.  
UTIL0 >
```

getthreshold

The **getthreshold** command displays the value of the current DMA threshold. The RFM2g device driver will use the bus master DMA feature present on some RFM2g devices if an I/O request qualifies (i.e. if the size is larger than or equal to the *Threshold*). One of the criteria for performing the DMA is that the I/O transfer be long enough that the time saved by performing the DMA offsets the overhead processing involved with the DMA itself. The default DMA threshold is driver-dependent. Refer to your driver-specific manual for the default DMA threshold value.

This command is useful since the amount of this overhead can vary between host computer configurations. The user can set a new threshold using the **setthreshold** command. The utility calls `RFM2gGetDMATHreshold()` to return the current DMA threshold value.

Syntax

```
getthreshold
```

Example

```
UTIL0 > getthreshold  
Current DMA Threshold: "32"  
UTIL0 >
```

help

The **help** command lists the name of each defined command and a short description of it. This command can also be used to show detailed usage information for a specific **rfm2g_util.c** command.

Syntax

```
help command
```

Parameters

<i>command</i>	The command help to display (I). Entering help displays a list of all commands for which help is available. Entering help followed by the command displays help information for the command if any is available.
----------------	--

Examples

```
UTIL0 > help settled

settled          : Set the current on/off state of the Reflective Memory board's
                  Status LED

Usage: settled          state
```

```
"state" is one of the following (0-1):
  0 for OFF
  1 for ON
UTIL0 > help
```

COMMAND	PARAMETERS
boardid	
cancelwait	event
checking	
clearevent	event
config	
devname	
disableevent	event
disablecallback	event
dllversion	
driverversion	
drvspecific	
dump	offset width length
enableevent	event
enablecallback	event
errmsg	ErrorCode
exit	

Press ENTER for more commands ...

COMMAND	PARAMETERS
first	
getdebug	
getdmabyteswap	
getled	
getpiobyteswap	
getthreshold	
help	[command]
mapuser	offset pages
memop	pattern offset width length verify float patterntype
nodeid	
peek8	offset
peek16	offset
peek32	offset
peek64	offset
performancetest	
poke8	value offset

Press ENTER for more commands ...

COMMAND	PARAMETERS
poke16	value offset
poke32	value offset
poke64	value offset
quit	
read	offset width length display
repeat	[-p] count cmd [arg...]
send	event tonode [ext_data]
setdebug	flag
setdmabyteswap	state
setled	state
setpiobyteswap	state
setthreshold	value
size	
unmapuser	
wait	event timeout
write	value offset width length

Press ENTER for more commands ...

UTIL0 >

mapuser

NOTE: Users should not use this command unless directed to do so by VMIC support personnel.

The **mapuser** command allows the user to get RFM Memory offset and page information, or to map RFM memory pages to the user space.

Using this command with no parameters displays which area, if any, has been mapped. The utility calls `RFM2gUserMemory()` to map RFM2g memory.

Syntax

```
mapuser offset pages
```

Parameters

<i>offset</i>	Offset in Reflective Memory at which to begin the mapping (I). Valid offset values are 0x0 to [<i>size of Reflective Memory on device - system memory page size</i>].
<i>pages</i>	Number of memory pages to map (I).

Examples

The following example displays the values of the mapped region:

```
UTIL0 > mapuser
UTIL0 >
```

The following example maps a buffer that begins at offset 0 and is 100 system passes long:

```
UTIL0 > mapuser 0 100
UTIL0 > RFM2gUserMemory assigned UserMemoryPtr = 0x50000000
UTIL0 >
```

memop

The **memop** command allows the user to fill or verify an area of Reflective Memory. This utility calls `RFM2gPoke8()`, `RFM2gPoke16()`, `RFM2gPoke32()` or `RFM2gPoke64()` to fill the memory.

Syntax

```
memop pattern offset width length verify float patterntype
```

Parameters

<i>pattern</i>	The pattern to write or verify (I).
<i>offset</i>	Width-aligned offset in Reflective Memory at which to begin the read or verify (I). Valid offset values are 0x0 to 0x3FFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFF for 128 Mbyte cards.
<i>width</i>	Indicates access width in bits, which is one of the following (I):

Value	Description
1	8-bit byte
2	16-bit word
4	32-bit longword
8	64-bit longword

<i>length</i>	Number of <i>width</i> units to write or verify (I), which is determined using the formula $[buffer\ size] / width$. For example, the <i>length</i> of a buffer size of 1024 in 32-bit longwords is 256 ($1024 / 4 = 256$).
---------------	--

Width Bit	Maximum Length (Dec/Hex) for 128 Mbyte Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit long	16777216 (0x1000000)

<i>verify</i>	Writes (0) or verifies (1) the <i>pattern</i> in Reflective Memory.
<i>float</i>	Specifies whether the <i>pattern</i> is (1) or is not (0) a floating point value.
<i>patterntype</i>	Specifies the <i>pattern</i> type, which is one of the following (I):

Type	Description
0	Pattern for fixed data
1	Pattern for incrementing address
2	Pattern for incrementing transfers count
3	Pattern for inverted incrementing address

Example

The following example writes the value 0x123456789ABCDEF to Reflective Memory, starting at offset 0. RFM2gPoke64 () is called 128 times, incrementing offset 8 each time it is called:

```
UTIL0 > memop 0x123456789ABCDEF 0 8 128 0 0
```

nodeid

The **nodeid** command displays the value of the RFM2g device node ID. Each RFM2g device on a RFM2g network is uniquely identified by its node ID, which is manually set by switches on the device when the RFM2g network is installed. The utility calls `RFM2gNodeID()` to return the RFM2g device node ID.

Syntax

```
nodeid
```

Example

```
UTIL0 > nodeId  
Node ID           0x01  
UTIL0 >
```

peek8, peek16, peek32 and peek64

The **peek** commands display the contents of the specified RFM2g offset. The specified memory offset is accessed as either an 8-bit byte, a 16-bit word, a 32-bit longword or a 64-bit longword and is displayed as a hexadecimal version of the RFM2g contents.

These commands make no attempt to lock the RFM2g during the access. These utilities call `RFM2gPeek8()`, `RFM2gPeek16()`, `RFM2gPeek32()` and `RFM2gPeek64()` to read from an RFM2g offset.

NOTE: See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

Syntax

`peek8 offset`

`peek16 offset`

`peek32 offset`

`peek64 offset`

Parameters

offset Offset in Reflective Memory from which to read (I).

Example (peek8)

```
UTIL0 > peek8 0
Data: 0x78            Read from Offset: 0x00000000
UTIL0 >
```

Example (peek16)

```
UTIL0 > peek16 0
Data: 0x5678        Read from Offset: 0x00000000
UTIL0 >
```

Example (peek32)

```
UTIL0 > peek32 0
Data: 0x12345678            Read from Offset: 0x00000000
UTIL0 >
```

Example (peek64)

```
UTIL0 > peek64 0
Data: 0x123456789ABCDEF        Read from Offset: 0x00000000
UTIL0 >
```

performancetest

The **performancetest** command uses the `RFM2gRead()` and `RFM2gWrite()` API functions to display the speed of reads and writes performed on your system.

Syntax

```
performancetest
```

Example

NOTE: The numbers in the following example are for illustration purposes only. Your actual system performance will vary.

```
UTIL0 > performancetest

VMIC RFM2g Performance Test (DMA Threshold is 32)
-----
  Bytes      Read IOps   Read MBps   Write IOps   Write MBps
    4         277760     1.1         900823       3.4
    8         456448     3.5        1254411       9.6
   12         343536     3.9        1197772      13.7
   16         275421     4.2         900820      13.7
   20         229826     4.4         724569      13.8
[...]
```

1048576	245	245.5	142	142.0
1310720	196	245.6	113	141.9
1572864	163	245.3	94	141.8
1835008	140	245.9	81	141.8
2097152	122	245.0	71	142.0

```
UTIL0 >
```

poke8, poke16, poke32 and poke64

The **poke** commands may be used to set or update consecutive RFM2g locations. The specified memory offset is written as either an 8-bit byte, a 16-bit word, a 32-bit longword or a 64-bit longword and must be entered in hexadecimal format.

These commands make no attempt to lock the RFM2g shared memory during the access. The utility calls `RFM2gPoke8()`, `RFM2gPoke16()`, `RFM2gPoke32()` and `RFM2gPoke64()` to write to an RFM2g offset.

NOTE: See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

Syntax

`poke8 value offset`

`poke16 value offset`

`poke32 value offset`

`poke64 value offset`

Parameters

`value offset` Value written to offset (I).

Example (poke8)

```
UTIL0 > poke8 255 0
Data: 0xFF      Written to Offset: 0x00000000
UTIL0 >
```

Example (poke16)

```
UTIL0 > poke16 65535 0
Data: 0xFFFF    Written to Offset: 0x00000000
UTIL0 >
```

Example (poke32)

```
UTIL0 > poke32 4294967295 0
Data: 0xFFFFFFFF Written to Offset: 0x00000000
UTIL0 >
```

Example (poke64)

```
UTIL0 > poke64 0x123456789ABCDEF 0
Data: 0x123456789ABCDEF      Written to Offset: 0x00000000
UTIL0 >
```

quit

The **quit** command terminates the command line interpreter.

Syntax

```
quit
```

Example

```
UTIL0 > quit  
Exit? (y/n):  
UTIL0 > y  
C: >
```

read

The **read** command reads data from the RFM2g node to system memory. Once transferred, the data is displayed. The utility calls `RFM2gRead()` to read data buffers. If DMA threshold and other conditions are met, DMA will be used; otherwise, PIO will be used.

NOTES:

See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

If byte swapping is enabled on the RFM2g device, *offset* and *length* must be width aligned.

Syntax

```
read offset width length display
```

Parameters

offset Offset in Reflective Memory at which to begin the read (I). Valid offset values are 0x0 to 0x3FFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFF for 128 Mbyte cards.

width Indicates access width in bits, which is one of the following (I):

Value	Description
1	8-bit byte
2	16-bit word
4	32-bit longword
8	64-bit longword

length Number of *width* units to display (I), which is determined using the formula $[buffer\ size] / width$. For example, the *length* of a buffer size of 1024 in 32-bit longwords is 256 ($1024 / 4 = 256$).

Width Bit	Maximum Length (Dec/Hex) for 128 Mbyte Cards
-----------	---

bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit long	16777216 (0x1000000)

display Display read data to the output device (0 = do not display; 1 = display information (default)).

Example

```
UTIL0 >read 0 1 0x40
```

```
          0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
0x00000000: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
0x00000010: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
0x00000020: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
0x00000030: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
```

```
UTIL0 >read 0 1 0x40 0
```

```
UTIL0 >
```

repeat

The **repeat** command is used to execute another utility command a specified number of times, as rapidly as possible. The command to be executed is supplied as an argument to the **repeat** command.

The `-p` switch may be useful to mark the progress of commands with large repetition counts. If the switch is used, the current pass number is output to the screen, followed by a `TAB` character. If the switch is omitted, no indication of the **repeat** command's progress is given.

The **repeat** command immediately stops if an error is reported while the command is executing.

Syntax

```
repeat [-p] count cmd [arg...]
```

Parameters

<code>-p</code>	Displays the number of times the specified utility command has repeated and updates this number to the screen.
<code>count</code>	The number of times to repeat the specified utility command.
<code>cmd</code>	The utility command to repeat.
<code>arg...</code>	Any arguments required by the specified utility command to repeat.

Example

```
UTIL0 > repeat 4 send 1 0xFF
Network Int 1 interrupt event was sent to node 255.
Network Int 1 interrupt event was sent to node 255.
Network Int 1 interrupt event was sent to node 255.
Network Int 1 interrupt event was sent to node 255.
UTIL0 >
```

return

The **return** command is used to exit a driver-specific sub-menu of commands (accessed using the **drvspecific** command) so that you can use common **rfm2g_util.c** commands.

Refer to your driver-specific manual for information on commands specific to your RFM2g driver.

Syntax

```
return
```

Examples

To access driver-specific commands:

```
UTIL0 > drvspecific
Welcome to the driver specific menu
UTILDRVSPEC >
```

To display a list of driver-specific commands:

```
UTILDRVSPEC > help

COMMAND          PARAMETERS
-----
help             [command]
repeat           [-p] count cmd [arg...]
return
UTILDRVSPEC >
```

To exit the driver-specific commands:

```
UTILDRVSPEC > return
Welcome to the main menu
UTIL0 >
```

send

Use the **send** command to transmit an interrupt event and a binary value to another node. RFM2g interrupt event types are available for an application program to use in signaling events to other RFM2g nodes.

If the destination RFM2g node number is given as `-1`, the event will be broadcast to all other RFM2g nodes on the network. The `ext_data` parameter is a user-defined, 32-bit value to send with the interrupt event. The utility calls `RFM2gSendEvent()` to send the RFM2g interrupt event.

Syntax

```
send event tonode [ext_data]
```

Parameters

<i>event</i>	The type of interrupt event to send (I). Interrupts correlate to the following event IDs:												
	<table> <thead> <tr> <th>Interrupt</th> <th>Event ID</th> </tr> </thead> <tbody> <tr> <td>Reset Interrupt</td> <td>0</td> </tr> <tr> <td>Network Interrupt 1</td> <td>1</td> </tr> <tr> <td>Network Interrupt 2</td> <td>2</td> </tr> <tr> <td>Network Interrupt 3</td> <td>3</td> </tr> <tr> <td>Network Interrupt 4</td> <td>4</td> </tr> </tbody> </table>	Interrupt	Event ID	Reset Interrupt	0	Network Interrupt 1	1	Network Interrupt 2	2	Network Interrupt 3	3	Network Interrupt 4	4
Interrupt	Event ID												
Reset Interrupt	0												
Network Interrupt 1	1												
Network Interrupt 2	2												
Network Interrupt 3	3												
Network Interrupt 4	4												
<i>tonode</i>	Who will receive the interrupt event (I) (<code>-1</code> sends the event to all nodes). NOTE: A node cannot send an event to itself.												
<i>ext_data</i>	User-defined 32-bit extended data to send (I).												

Example

```
UTIL0 > send 0 0
"RESET" interrupt event was sent to node 0.
UTIL0 >
```

setdebug

NOTE: Application programs should not use this command unless directed to do so by VMIC support personnel.

Each possible RFM2g device driver debug output message is assigned to a debug message class. The device driver will generate messages of that class if the corresponding flag bit is set in the control word. The **setdebug** command allows an application program to set that control word (i.e. this command sets the driver's debug flags). The change is effective immediately.

Application programs do not normally need to alter the setting of the debug message output control word. The utility calls `RFM2gSetDebugFlags()` to turn debug flags on or off.

Syntax

```
setdebug [-] flag
```

Parameters

-	Clears instead of setting the flag.
<i>flag</i>	New debug flags (I). Valid strings are:
String	Description
allflags	Turns all debug flags on
close	Trace close(2) system calls
error	Report critical errors
init	Trace device probing and search
intr	Trace interrupt service
ioctl	Trace ioctl(2) system calls
minphys	Trace minphys limits
mmap	Trace mmap(2) system calls
mutex	Trace synchronization and locking
not_intr	Trace non-RFM interrupt service
open	Trace open(2) system calls
peek	Trace peeks
poke	Trace pokes
read	Trace read(2) system calls
slow	Let syslogd get the message
strat	Trace read/write strategy
timer	Trace interval timer
trace	Trace subroutine entry/exit
write	Trace write(2) system calls

Example

```
UTIL0 > setdebug error
Debug Flag "error" was set.
UTIL0 >
```

setdmabyteswap

The **setdmabyteswap** command enables or disables byte swapping DMA transfers to or from an RFM2g device. This command provides 4-byte swaps only (i.e. byte swapping based on size is not performed by the RFM2g device). The utility calls `RFM2gSetDMAByteSwap()` to turn DMA byte swapping on or off.

NOTE: DMA byte swapping may be enabled by default when the driver has been built for use on big endian systems. Refer to your driver-specific manual for the default setting of DMA byte swapping.

Syntax

```
setdmabyteswap state
```

Parameters

<i>state</i>	Sets the state of DMA byte swapping, which is one of the following (I):
State	Description
0	Turns DMA byte swapping off
1	Turns DMA byte swapping on

Example

```
UTIL0 > setdmabyteswap 1
The Reflective Memory board's DMA Byte Swap is ON.
UTIL0 >
```

setled

The **setled** command sets the current on/off state of the Reflective Memory board's STATUS LED. The utility calls `RFM2gSetLed()` to turn the STATUS LED on or off.

Syntax

```
setled state
```

Parameters

state The state of the STATUS LED: 0=>OFF, 1=>ON (O).

Example

```
UTIL0 > setled 1  
The Reflective Memory board's status LED is ON  
UTIL0 >
```

setpiobyteswap

The **setpiobyteswap** command enables or disables byte swapping PIO (Programmed IO) transfers to or from an RFM2g device. This function provides 4-byte swaps only (i.e. byte swapping based on size is not performed by the RFM2g device). The utility calls `RFM2gSetPIOByteSwap()` to turn PIO byte swapping on or off.

NOTE: PIO byte swapping may be enabled by default when the driver has been built for use on big endian systems. Refer to your driver-specific manual for the default setting of PIO byte swapping.

Syntax

```
setpiobyteswap state
```

Parameters

state Sets the state of PIO byte swapping, which is one of the following (I):

State	Description
0	Turns PIO byte swapping off
1	Turns PIO byte swapping on

Example

```
UTIL0 > setpiobyteswap 1
The Reflective Memory board's PIO Byte Swap is ON.
UTIL0 >
```

setthreshold

The **setthreshold** command sets the transfer size at which reads and writes will use DMA to transfer data. If the **read** or **write** command is used, DMA will be used if the size of the data is larger than or equal to the threshold *value*. A threshold can be set for each handle created by a call to `RFM2gOpen()`.

The amount of cycles taken to set up a DMA transfer can increase the transfer time for small transfer sizes. The transfer size for which DMAs are more efficient than standard transfers varies, depending on the system.

DMA is generally preferred over the Programmed IO (PIO) method for transferring data. PIO operations require the usage of the CPU to process the transfer, while DMA enables the Reflective Memory controller to access system memory while leaving the CPU's resources unaffected. However, the best value to use (i.e. PIO vs. DMA) is system-dependent. The RFM2g driver performs approximately 5 PCI accesses to set up and process a DMA request and generates an interrupt on completion of the DMA operation. In general, DMA is the preferred method if a PIO transfer requires more than 6 to 10 PCI cycles to complete.

A *value* of `0xFFFFFFFF` specifies that DMAs will never be used for data transfer. The utility calls `RFM2gSetDMAThreshold()` to set the DMA threshold size.

NOTE: The default DMA threshold *value* is driver-dependent and should be changed *only* if recommended by the driver's documentation. Refer to your driver-specific manual for more information, including the default value.

Syntax

```
setthreshold value
```

Parameters

value New DMA threshold value (I).

Example

```
UTIL0 > setthreshold 128  
UTIL0 >
```

size

The **size** command displays the value of the total amount of virtual memory space available on the RFM2g device. The user may access RFM2g space between offset 0 and `RFM2gSize(rh)-1`.

RFM2g boards may be configured with a variety of memory sizes. The device driver and API library determine the amount of memory contained on a RFM2g device as it is initialized. A user may then use **size** to obtain the number of bytes on the board. The utility calls `RFM2gSize()` to return the RFM2g device's total available memory space.

Syntax

```
size
```

Example

```
UTIL0 > size
      Size                134217728 (0x08000000)
UTIL0 >
```

unmapuser

NOTE: Users should not use this command unless directed to do so by VMIC support personnel.

The **unmapuser** command unmaps the RFM2g memory buffer from user memory space. The utility calls `RFM2gUnMapUserMemory()` to unmap the RFM2g memory buffer.

Syntax

```
unmapuser UserMemoryPtr pages
```

Parameters

UserMemoryPtr Pointer returned by the **mapuser** command (O).
Pages The number of pages mapped by the **mapuser** command (O).

Example

```
UTIL0 > unmapuser 0x50000000 100  
UTIL0 >
```

wait

The **wait** command allows the user to wait for an RFM2g interrupt event. The utility program blocks until the next RFM2g interrupt event of the requested type has been received, or the timeout period expires. The event must be enabled by this application before it can be received; otherwise, a timeout will occur. The utility calls `RFM2gWaitForEvent()` to wait for the RFM2g event.

Syntax

```
wait event timeout
```

Parameters

<i>event</i>	The type of interrupt event on which to wait (I). Interrupts correlate to the following event IDs:																		
	<table> <thead> <tr> <th>Interrupt</th> <th>Event ID</th> </tr> </thead> <tbody> <tr> <td>Reset Interrupt</td> <td>0</td> </tr> <tr> <td>Network Interrupt 1</td> <td>1</td> </tr> <tr> <td>Network Interrupt 2</td> <td>2</td> </tr> <tr> <td>Network Interrupt 3</td> <td>3</td> </tr> <tr> <td>Network Interrupt 4 (Init Interrupt)</td> <td>4</td> </tr> <tr> <td>Bad Data Interrupt</td> <td>5</td> </tr> <tr> <td>RX FIFO Full Interrupt</td> <td>6</td> </tr> <tr> <td>Rogue Packet Detected and Removed</td> <td>7</td> </tr> </tbody> </table>	Interrupt	Event ID	Reset Interrupt	0	Network Interrupt 1	1	Network Interrupt 2	2	Network Interrupt 3	3	Network Interrupt 4 (Init Interrupt)	4	Bad Data Interrupt	5	RX FIFO Full Interrupt	6	Rogue Packet Detected and Removed	7
Interrupt	Event ID																		
Reset Interrupt	0																		
Network Interrupt 1	1																		
Network Interrupt 2	2																		
Network Interrupt 3	3																		
Network Interrupt 4 (Init Interrupt)	4																		
Bad Data Interrupt	5																		
RX FIFO Full Interrupt	6																		
Rogue Packet Detected and Removed	7																		
<i>timeout</i>	Indicates the time, in milliseconds, to wait for the event before returning.																		

Example

```
UTIL0 > wait 1 1000
Waiting for event...
Received Network INT 1 event from node 5.
This events extended data is 0X12345678.
UTIL0 > wait 0 0
Waiting for event ... Notification for this event has already been requested.
UTIL0 > wait 0 10000
Waiting for event ... Notification for this event has already been requested.
UTIL0 > wait 1 0
Waiting for event ... Timed out.
UTIL0 > wait 1 10000
Waiting for event ... Timed out.
UTIL0 >
```

write

The **write** command writes one or more bytes starting at an offset in Reflective Memory (i.e. allows the user to fill memory area with a byte, word or longword). The utility calls `RFM2gWrite()` to write data buffers. If DMA threshold and other conditions are met, DMA will be used; otherwise, PIO will be used.

NOTES:

See "Big Endian and Little Endian Data Conversions" on page 49 for information on the big endian/little endian byte-reordering process used by the RFM2g driver when accessing multibyte data.

If byte swapping is enabled on the RFM2g device, *offset* and *length* must be width aligned.

Syntax

```
write value offset width length
```

Parameters

<i>value</i>	Byte, word or longword value to write to the range specified by <i>offset</i> , <i>length</i> and <i>width</i> (I).										
<i>offset</i>	Width-aligned offset in Reflective Memory at which to begin the write (I). Valid offset values are 0x0 to 0x3FFFFFF for 64 Mbyte cards, and 0x0 to 0x7FFFFFF for 128 Mbyte cards.										
<i>width</i>	Indicates access width in bits, which is one of the following (I):										
	<table> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>1</td> <td>8-bit byte</td> </tr> <tr> <td>2</td> <td>16-bit word</td> </tr> <tr> <td>4</td> <td>32-bit longword</td> </tr> <tr> <td>8</td> <td>64-bit longword</td> </tr> </tbody> </table>	Value	Description	1	8-bit byte	2	16-bit word	4	32-bit longword	8	64-bit longword
Value	Description										
1	8-bit byte										
2	16-bit word										
4	32-bit longword										
8	64-bit longword										
<i>length</i>	Number of <i>width</i> units to write (I), which is determined using the formula $[buffer\ size] / width$. For example, the <i>length</i> of a buffer size of 1024 in 32-bit longwords is 256 ($1024 / 4 = 256$).										

Width Bit	Maximum Length (Dec/Hex) for 128 Mbyte Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit long	16777216 (0x1000000)

Example

```
UTIL0 > write 0 0 1 16 1000  
Write used DMA.  
Write completed.  
UTIL0 >
```

Troubleshooting the rfm2g_util.c Command Line Interpreter

If you encounter problems building or exercising the RFM2g driver, this section contains possible solutions and discusses the most common sources of errors and how to reduce error possibilities.

Errors

Use the following method to perform driver build troubleshooting.

If the compiler outputs the following error, the operating system for which the file is to be compiled has not been defined in the build specification.

```
C:\RFM2g\PCI\VxWorks\rfm2g_util.c:59: #error OS not defined, define LINUX or
RFM2G_VXWORKS
```

To resolve this error, define the operating system in the build options as follows:

Operating System	Build Option Definition
VxWorks	-DRFM2G_VXWORKS
Solaris	-DRFM2G_SOLARIS

RFM2g Sample Applications

Contents

rfm2g_sender.c	148
rfm2g_receiver.c	149
rfm2g_map.c	150
rfm2g_map.c	150
rfm2g_sender.c and rfm2g_receiver.c Example Workflow	151
rfm2g_map.c Example Workflow	153

Introduction

This chapter contains information on the three sample application programs delivered with the RFM2g driver in the rfm2g/samples folder. These programs provide examples on how to use the driver and API with your application and are intended to work together to demonstrate basic data transfer and interrupt handshaking:

- **rfm2g_sender.c**
- **rfm2g_receiver.c**
- **rfm2g_map.c**

To use the programs together, it is assumed that:

- Two systems are present
- Each system contains a Reflective Memory card
- The Reflective Memory cards in the systems are connected to each other
- Each system has the RFM2g device driver installed

See your driver-specific documentation for the location of these files and information on how to build the executable programs.

rfm2g_sender.c

The **rfm2g_sender.c** program runs on system 1 and does the following:

1. Writes a small buffer of data to Reflective Memory
2. Sends an interrupt event to system number 2
3. Waits to receive an interrupt event from system number 2
4. Reads a buffer of data (written by system number 2) from a different Reflective Memory location
5. Closes the RFM2g driver.

rfm2g_receiver.c

The **rfm2g_receiver.c** program runs on system 2 and does the following:

1. Opens the RFM2g driver
2. Waits to receive an interrupt event from system number 1
3. Reads the buffer of data (written by system number 1) from Reflective Memory
4. Writes the buffer of data to a different Reflective Memory location
5. Sends an interrupt event to system number 1
6. Closes the RFM2g driver.

rfm2g_map.c

The **rfm2g_map.c** program demonstrates the usage of the `RFM2gUserMemory()` function, which enables you to obtain a pointer for directly accessing the memory of the RFM2g device.

rfm2g_sender.c and rfm2g_receiver.c Example Workflow

The following is an example workflow using the **rfm2g_sender.c** and **rfm2g_receiver.c** programs.

In this example:

- Verbose mode is *not* enabled for **rfm2g_sender.c** or **rfm2g_receiver.c**.
 - Continuous mode is enabled for **rfm2g_receiver.c**.
 - The device number of the host computer running the **rfm2g_sender.c** program is 0.
 - The device number of the target computer running the **rfm2g_receiver.c** program is 3.
1. Start the **rfm2g_sender.c** program on the host system by following the directions in your driver-specific manual.

The following is displayed in the console window:

```
PCI RFM2g Sender
```

```
Please enter device number:
```

2. Type the RFM2g host's device number (0, 1, etc.) and press `<ENTER>`.

The following is displayed in the host's console window:

```
Do you wish for sender to loop continuously? (Y/N):
```

3. Enter `y` to use the **rfm2g_sender.c** command in continuous mode so that it will run continuously.

-or-

Enter `n` if you do not want to use **rfm2g_sender.c** in continuous mode.

The following is displayed in the host's console window:

```
Do you wish for sender to be verbose? (Y/N):
```

4. Enter `y` to use the **rfm2g_sender.c** command in verbose mode so that the buffer contents are dumped to the screen while it is running.

-or-

Enter `n` if you do not want to use **rfm2g_sender.c** in verbose mode.

The following is displayed in the host's console window:

```
What is the Reflective Memory Node ID of the computer running  
the "RFM2G_receiver" program?
```

5. Type the RFM2g target's device number (0, 1, etc.) and press `<ENTER>`.

The following is displayed in the host's console window:

```
Start the "RFM2G_receiver" program on the other computer.  
Press RETURN to continue ...
```

6. Start the **rfm2g_receiver.c** program on the target system by following the directions in your driver-specific manual.

The following is displayed in the console window:

```
PCI RFM2g Receiver
Please enter device number:
```

7. Type the RFM2g target's device number (0, 1, etc.) and press <ENTER>.

The following is displayed in the target's console window:

```
Do you wish for receiver to loop continuously? (Y/N):
```

8. Enter **y** to use the **rfmwg_receiver.c** command in continuous mode so that it will run continuously.

-or-

Enter **n** if you do not want to use **rfm2g_receiver.c** in continuous mode.

The following is displayed in the target's console window:

```
Do you wish for receiver to be verbose? (Y/N):
```

9. Enter **y** to use the **rfm2g_receiver.c** command in verbose mode so that the buffer contents are dumped to the screen while it is running.

-or-

Enter **n** if you do not want to use **rfm2g_receiver.c** in verbose mode.

The following is displayed in the target's console window:

```
Waiting 60 seconds to receive an interrupt from the other Node ...
Received the interrupt from Node 3.

Data was read from Reflective Memory.

The data was written to Reflective Memory starting at offset 0x2000.

An interrupt was sent to Node 3.

Success!
```

10. Return to the host system. The following is displayed in the host's console window:

```
The data was written to Reflective Memory. An interrupt was sent to Node 3.

Waiting 60 seconds for an interrupt from Node 3 ... Received the interrupt from
Node 3.

Success!
```

rfm2g_map.c Example Workflow

The following is an example workflow using the **rfm2g_map.c** program.

In this example, the RFM2g device's number is 0.

1. Start the **rfm2g_map.c** program by following the directions in your driver-specific manual.

The following is displayed in the console window:

```
PCI RFM2g Map
Please enter device number:
```

2. Type the RFM2g device's number (0, 1, etc.) and press <ENTER>.

The following is displayed in the host's console window:

```
Wrote: A5A50000, Read: A5A50000
Wrote: A5A50001, Read: A5A50001
Wrote: A5A50002, Read: A5A50002
Wrote: A5A50003, Read: A5A50003
```

```
Success!
```

