

# Operator's Reference Manual

## Common RFM2g\* Application Program Interface (API) and Command Line Interpreter for RFM2g\* Drivers

Publication No. 523-000447-000 Rev. K.0



## Document History

Revision	Date	Description
K.0	September 2016	Reformatting

## Waste Electrical and Electronic Equipment (WEEE) Returns



Abaco Systems is registered with an approved Producer Compliance Scheme (PCS) and, subject to suitable contractual arrangements being in place, will ensure WEEE is processed in accordance with the requirements of the WEEE Directive.

Abaco Systems will evaluate requests to take back products purchased by our customers before August 13, 2005 on a case-by-case basis. A WEEE management fee may apply.

# About This Manual

## Conventions

### Notices

This manual may use the following types of notice:



#### **WARNING**

Warnings alert you to the risk of severe personal injury.



#### **CAUTION**

Cautions alert you to system danger or loss of data.



#### **NOTE**

Notes call attention to important features or instructions.



#### **TIP**

Tips give guidance on procedures that may be tackled in a number of ways.



#### **LINK**

Links take you to other documents or websites.

### Numbers

All numbers are expressed in decimal, except addresses and memory or register data, which are expressed in hexadecimal. Where confusion may occur, decimal numbers have a “D” subscript and binary numbers have a “b” subscript. The prefix “0x” shows a hexadecimal number, following the ‘C’ programming language convention. Thus:

$$\text{One dozen} = 12_D = 0x0C = 1100_b$$

The multipliers “k”, “M” and “G” have their conventional scientific and engineering meanings of  $\times 10^3$ ,  $\times 10^6$  and  $\times 10^9$  respectively. The only exception to this is in the description of the size of memory areas, when “k”, “M” and “G” mean  $\times 2^{10}$ ,  $\times 2^{20}$  and  $\times 2^{30}$  respectively.



#### **NOTE**

When describing transfer rates, “k”, “M” and “G” mean  $\times 10^3$ ,  $\times 10^6$  and  $\times 10^9$  not  $\times 2^{10}$ ,  $\times 2^{20}$  and  $\times 2^{30}$ .

In PowerPC terminology, multiple bit fields are numbered from 0 to n where 0 is the MSB and n is the LSB. PCI terminology follows the more familiar convention that bit 0 is the LSB and n is the MSB.

### Text

Signal names ending with a tilde (“~”) denote active low signals; all other signals are active high. “N” and “P” denote the low and high components of a differential signal respectively.

## Further Information

### Abaco Website

You can find information regarding Abaco products on the following website:



### Abaco Documents

This document is distributed via the Abaco website. You may register for access to manuals via the website.



## Technical Support Contact Information

You can find technical assistance contact details on the website Embedded Support page.



Abaco will log your query in the Technical Support database and allocate it a unique Case number for use in any future correspondence.

Alternatively, you may also contact Abaco's Technical Support via:



## Returns

If you need to return a product, there is a Return Materials Authorization (RMA) form available via the website Embedded Support page.



Do not return products without first contacting the Abaco Repairs facility.

## Safety Summary

The following general safety precautions must be observed during all phases of the operation, service, and repair of this product. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of this product.

Abaco assumes no liability for the customer's failure to comply with these requirements.

### Ground the System

To minimize shock hazard, the chassis and system cabinet must be connected to an electrical ground. A three-conductor AC power cable should be used. The power cable must either be plugged into an approved three-contact electrical outlet or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power outlet.

### Do Not Operate in an Explosive Atmosphere

Do not operate the system in the presence of flammable gases or fumes. Operation of any electrical system in such an environment constitutes a definite safety hazard.

### Keep Away from Live Circuits

Operating personnel must not remove product covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

### Do Not Service or Adjust Alone

Do not attempt internal service or adjustment unless another person capable of rendering first aid and resuscitation is present.

### Do Not Substitute Parts or Modify System

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification to the product. Return the product to Abaco for service and repair to ensure that safety features are maintained.

# Table of Contents

1 • Application Program Interface (API) Library	10
1.1 Introduction.	10
1.2 Using the Application Program Interface.	10
1.2.1 Opening the RFM2g Driver	11
1.2.2 Routine Code for Use with API Function Examples	11
1.3 RFM2g Error Codes.	12
1.4 RFM2g API Functions.	14
1.5 RFM2g Opening and Closing API Functions.	16
1.5.1 RFM2gOpen()	16
1.5.2 RFM2gClose()	18
1.6 RFM2g Configuration API Functions.	18
1.6.1 RFM2gGetConfig()	20
1.6.2 RFM2gUserMemory()	21
1.6.3 RFM2gUnMapUserMemory()	23
1.6.4 RFM2gUserMemoryBytes()	25
1.6.5 RFM2gUnMapUserMemoryBytes()	27
1.6.6 RFM2gNodeID()	29
1.6.7 RFM2gBoardID()	30
1.6.8 RFM2gSize()	31
1.6.9 RFM2gFirst()	32
1.6.10 RFM2gDeviceName()	33
1.6.11 RFM2gDLLVersion()	34
1.6.12 RFM2gDriverVersion()	35
1.6.13 RFM2gGetDMAThreshold()	36
1.6.14 RFM2gSetDMAThreshold()	37
1.6.15 RFM2gGetDMAByteSwap()	39
1.6.16 RFM2gSetDMAByteSwap()	40
1.6.17 RFM2gGetPIOByteSwap()	41
1.6.18 RFM2gSetPIOByteSwap()	42
1.7 RFM2g Data Transfer API Functions.	43
1.7.1 Data Transfer Considerations	43
1.7.2 RFM2gRead()	45
1.7.3 RFM2gWrite()	47
1.7.4 RFM2gPeek8(), RFM2gPeek16(), RFM2gPeek32() and RFM2gPeek64()	49
1.7.5 RFM2gPoke8(), RFM2gPoke16(), RFM2gPoke32() and RFM2gPoke64()	51
1.8 RFM2g Interrupt Event API Functions.	53
1.8.1 RFM2gEnableEvent()	54
1.8.2 RFM2gDisableEvent()	56
1.8.3 RFM2gSendEvent()	58
1.8.4 RFM2gWaitForEvent()	60
1.8.5 RFM2gEnableEventCallback()	63
1.8.6 RFM2gDisableEventCallback()	65
1.8.7 RFM2gClearEvent()	67
1.8.8 RFM2gCancelWaitForEvent()	69
1.8.9 RFM2gClearEventCount()	71
1.8.10 RFM2gGetEventCount()	73
1.9 RFM2g Utility API Functions.	75

1.9.1 RFM2gErrorMsg()	76
1.9.2 RFM2gGetLed()	77
1.9.3 RFM2gSetLed()	78
1.9.4 RFM2gCheckRingCont()	79
1.9.5 RFM2gGetDebugFlags()	80
1.9.6 RFM2g SetDebugFlags()	82
1.9.7 RFM2gGetDarkOnDark()	84
1.9.8 RFM2gSetDarkOnDark()	85
1.9.9 RFM2gClearOwnData()	86
1.9.10 RFM2gGetTransmit()	87
1.9.11 RFM2gSetTransmit()	88
1.9.12 RFM2gGetLoopback()	89
1.9.13 RFM2gSetLoopback()	90
1.9.14 RFM2gGetParityEnable()	91
1.9.15 RFM2gSetParityEnable()	92
1.9.16 RFM2gGetMemoryOffset()	93
1.9.17 RFM2gSetMemoryOffset()	94
1.9.18 RFM2gGetSlidingWindow()	95
1.9.19 RFM2gSetSlidingWindow()	96
<b>2 • rfm2g_util.c Utility Program</b>	<b>97</b>
2.1 Introduction.	97
2.2 RFM2g Command Line Interpreter.	97
2.2.1 Using the Command Line Interpreter	97
2.2.2 Command Line Interpreter Example	99
2.3 Utility Commands.	100
2.3.1 boardid	103
2.3.2 cancelwait	103
2.3.3 checkring	104
2.3.4 clearevent	104
2.3.5 cleareventcount	105
2.3.6 clearowndata	105
2.3.7 config	106
2.3.8 devname	106
2.3.9 disableevent	107
2.3.10 disablecallback	108
2.3.11 dllversion	108
2.3.12 driverversion	109
2.3.13 drvspecific	109
2.3.14 dump	110
2.3.15 enableevent	111
2.3.16 enablecallback	112
2.3.17 errormsg	113
2.3.18 exit	113
2.3.19 first	113
2.3.20 getdarkondark	114
2.3.21 getdebug	114
2.3.22 getdmabyteswap	114
2.3.23 geteventcount	115
2.3.24 getled	115
2.3.25 getmemoryoffset	116
2.3.26 getloopback	116
2.3.27 getparityenable	116
2.3.28 getpiobyteswap	116

2.3.29	getslidingwindow	117
2.3.30	getthreshold	117
2.3.31	gettransmit	117
2.3.32	help	118
2.3.33	mapuser	120
2.3.34	mapuserbytes	121
2.3.35	memop	122
2.3.36	nodeid	123
2.3.37	peek8, peek16, peek32 and peek64	124
2.3.38	performancetest	125
2.3.39	poke8, poke16, poke32 and poke64	126
2.3.40	quit	127
2.3.41	read	127
2.3.42	repeat	128
2.3.43	return	129
2.3.44	send	130
2.3.45	setdarkondark	131
2.3.46	setdebug	132
2.3.47	setdmabyteswap	133
2.3.48	setled	133
2.3.49	setloopback	134
2.3.50	setmemoryoffset	134
2.3.51	setparityenable	135
2.3.52	setpiobyteswap	135
2.3.53	setslidingwindow	136
2.3.54	setthreshold	136
2.3.55	settransmit	137
2.3.56	size	137
2.3.57	unmapuser	138
2.3.58	unmapuserbytes	138
2.3.59	wait	139
2.3.60	write	140
2.4	Troubleshooting the rfm2g_util.c Command Line Interpreter.	141
2.4.1	Errors	141
3	• RFM2g Sample Applications	142
3.1	rfm2g_sender.c.	142
3.2	rfm2g_receiver.c.	142
3.3	rfm2g_map.c.	143
3.4	rfm2g_sender.c and rfm2g_receiver.c Example Workflow.	143
3.5	rfm2g_map.c Example Workflow.	145



# List of Tables

Table 1-1 Common RFM2g Error Code .....	13
Table 1-2 RFM2g API Functions .....	14
Table 1-3 RFM2g Opening and Closing API Functions .....	16
Table 1-4 RFM2g Configuration API Functions .....	18
Table 1-5 RFM2g Data Transfer API Functions .....	43
Table 1-6 RFM2g Interrupt Event API Functions .....	53
Table 1-7 RFM2gEnableEvent() .....	54
Table 1-8 RFM2gDisableEvent() .....	56
Table 1-9 RFM2gWaitForEvent() .....	60
Table 1-10 RFM2gEnableEventCallback() .....	63
Table 1-11 RFM2gDisableEventCallback() .....	65
Table 1-12 RFM2g Utility API Functions .....	75
Table 2-1 RFM2g Driver Commands .....	100

# 1 • Application Program Interface (API) Library

## 1.1 Introduction

The 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.

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.

## 1.2 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.

### 1.2.1 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 in “Routine Code for Use with API Function Examples” below.

### 1.2.2 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](#) for more information.

## 1.3 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.



#### NOTE

Error code values are driver-specific.  
Use the error code name instead of the value in user applications.

Table 1-1 Common RFM2g Error Code

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
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_CANCELED	Wait for event canceled

Table 1-1 Common RFM2g Error Code (Continued)

Error Code	Description
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_SIGNALED	Signal from OS
RFM2G_NODE_ID_SELF	Cannot send event to self
RFM2G_MAX_ERROR_CODE	Invalid error code

## 1.4 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

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()
RFM2gUserMemoryBytes()	Maps RFM2g memory to the user space.
RFM2gUnMapUserMemoryBytes()	Unmaps RFM2g memory from the user space that was mapped using RFM2gUserMemoryBytes()
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 (Direct Memory Access) threshold value.
RFM2gSetDMAThreshold()	Sets the transfer size at which reads and writes will use DMA.
RFM2gGetDMAByteSwap() *	Returns the state of DMA byte swapping specified by RFM2gGetDMAThreshold().
RFM2gSetDMAByteSwap() *	Sets the current ON/OFF state of DMA byte swapping.

Table 1-2 RFM2g API Functions (Continued)

RFM2g Configuration API Functions (Continued)	
API Function	Description
RFM2gGetPIOByteSwap() *	Returns the state of PIO 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.
RFM2gClearEventCount()	Clears the event counter for an interrupt event type.
RFM2gGetEventCount()	Gets the number of interrupt events for a given event type.
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.
RFM2gGetDarkOnDark() *	Retrieves the current ON/OFF state of the Reflective Memory boards Dark on Dark feature.

Table 1-2 RFM2g API Functions (Continued)

RFM2g Utility API Functions (Continued)	
API Function	Description
RFM2gSetDarkOnDark() *	Sets the ON/OFF state of the Reflective Memory board's Dark on Dark feature.
RFM2gClearOwnData()	Returns the state of the Own Data bit and resets the state if set, calling this function will turn OFF the Own Data LED if ON.
RFM2gGetTransmit()	Retrieves the current ON/OFF state of the Reflective Memory board's transmitter.
RFM2gSetTransmit()	Sets the ON/OFF state of the Reflective Memory board's transmitter.
RFM2gGetLoopback()	Retrieves the current ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
RFM2gSetLoopback()	Sets the ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
RFM2gGetParityEnable()	Retrieves the current ON/OFF state of the Reflective Memory boards' parity checking on all onboard memory accesses.
RFM2gSetParityEnable()	Sets the ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.
RFM2gGetMemoryOffset()	Gets the memory offset of the Reflective Memory board.
RFM2gSetMemoryOffset()	Sets the memory offset of the Reflective Memory board.
RFM2gGetSlidingWindow() *	Retrieves the base Reflective Memory offset and size of the current sliding window.
RFM2gSetSlidingWindow() *	Sets the base Reflective Memory offset of the sliding window.

\* **NOTE:** These APIs are not supported on VME RFM2g hardware.

† **NOTE:** These APIs are not supported on VxWorks OS.

## 1.5 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.

### 1.5.1 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

**DevicePath** Path to special device file (I). Refer to your driver-specific manual for the format of DevicePath.

**rh** Pointer to an RFM2GHANDLE structure (IO).

## Return Values

**Success** RFM2G\_SUCCESS

**Failure** RFM2G\_BAD\_PARAMETER\_1 – NULL or invalid DevicePath.

RFM2G\_BAD\_PARAMETER\_2 – rh is NULL

RFM2G\_HANDLE\_NOT\_NULL – \*rh 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\_RFM2G\_BOARD – No RFM2g device found.

RFM2G\_BAD\_RFM2G\_BOARD\_ID – RFM2g device has bad board ID.

## Example

See [Section 1.2.2 Routine Code for Use with API Function Examples](#) on page 11 for an example of the RFM2gOpen() command.

## Related Commands

- RFM2gClose()

## 1.5.2 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 — OS returned an error.

RFM2G\_NOT\_IMPLEMENTED — API function is not implemented in the driver.

RFM2G\_NOT\_OPEN — Device is not open.

### Example

See [Section 1.2.2 Routine Code for Use with API Function Examples](#) on page 11 for an example of the RFM2gClose() command.

### Related Commands

- RFM2gOpen()

## 1.6 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().
RFM2gUnMapUserMemoryBytes()	Maps RFM2g memory to the user space.
RFM2gUnmpaUserBytes	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.

Table 1-4 RFM2g Configuration API Functions (Continued)

API Function	Description
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 byte swapping specified by RFM2gSetDMAThreshold().
RFM2gSetDMAByteSwap()	Sets the current ON/OFF state of DMA byte swapping.
RFM2gGetPIOByteSwap()	Returns the state of PIO byte swapping specified by RFM2gSetPIOByteSwap().
RFM2gSetPIOByteSwap()	Sets the current ON/OFF state of PIO byte swapping.

## 1.6.1 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 opened 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 — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11 :

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

### Related Commands

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

## 1.6.2 RFM2gUserMemory()

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

### Syntax

```
STDRFM2GCALL RFM2gUserMemory( RFM2GHANDLE rh,
                                volatile void **UserMemoryPtr,
                                RFM2G_UINT64 Offset,
                                RFM2G_UINT32 Pages );
```

### Parameters

**rh** Handle to opened RFM2g device (I).

**UserMemoryPtr** Where to put the pointer to mapped RFM2g space (IO).



#### NOTE

The volatile keyword must be used to force an implementation to suppress optimization.

**Offset** Base byte offset of RFM2g memory to map (I).

**Pages** Number of pages to map (I).



#### NOTE

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 – rh is NULL.
- RFM2G\_OS\_ERROR – 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 – UserMemoryPtr is NULL.
- RFM2G\_BAD\_PARAMETER\_4 – Pages is 0.
- RFM2G\_MEM\_NOT\_MAPPED – System memory map call failed.
- 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 the [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

- RFM2gUnMapUserMemory()

### 1.6.3 RFM2gUnMapUserMemory()

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

#### Syntax

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

#### Parameters

**rh** Handle to opened RFM2g device (I).

**UserMemoryPtr** Pointer to mapped RFM2g space (IO).



#### NOTE

The volatile keyword must be used to force an implementation to suppress optimization.

**Pages** The number of pages originally mapped to UserMemoryPtr (I). Refer to your driver-specific manual for information on using this parameter.

#### Return Values

**Success** RFM2G\_SUCCESS

**Failure** RFM2G\_NULL\_DESCRIPTOR — rh is NULL.

RFM2G\_OS\_ERROR — 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 — UserMemoryPtr is NULL.

RFM2G\_BAD\_PARAMETER\_3 — Pages 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UINT32 Offset = 0;
RFM2G_UINT32 Pages = 1;
volatile RFM2G_UINT32 *pUser; /* Must be volatile */

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

## Related Commands

- RFM2gUserMemory()



## 1.6.4 RFM2gUserMemoryBytes()

The RFM2gUserMemoryBytes() function maps the Reflective Memory address space to a user-level pointer, allowing direct access to RFM memory via pointer de-referencing. All transfers using this pointer will use PIO and will not use DMA. This function is similar to the RFM2gUserMemory() except that it takes the number for bytes instead of the number for pages. User application should choose this function over RFM2gUserMemory() for portability across platforms with different page sizes.

### Syntax

```
STDRFM2GCALL RFM2gMapUserBytes( RFM2GHANDLE rh,  
                                volatile void **UserBytePtr,  
                                RFM2G_UINT64 Offset, RFM2G_UINT32 Bytes );
```

### Parameters

**rh** Handle to opened RFM2g device (I).

**UserMemoryPtr** Where to put the pointer to mapped RFM2g space (IO).



### NOTE

The volatile keyword must be used to force an implementation to suppress optimization.

**Offset** Base byte offset of RFM2g memory to map (I).

**Bytes** Number of bytes to map (I).

### Return Values

**Success** RFM2G\_SUCCESS

**Failure**

- RFM2G\_NULL\_DESCRIPTOR — rh is NULL.
- RFM2G\_OS\_ERROR — 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 — UserBytePtr is NULL.
- RFM2G\_BAD\_PARAMETER\_4 — Bytes is 0.
- RFM2G\_MAP\_NOT\_ALLOWED — Invalid map Offset and Bytes (Bytes 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UINT32 Offset = 0;
RFM2G_UINT32 Bytes = 4096;
volatile char *pUser; /* Must be volatile */

result = RFM2gUserMemoryBytes(Handle,
                               (volatile void **)&pUser, Offset, Bytes);
```

## Related Commands

- RFM2gUnMapUserMemoryBytes()

## 1.6.5 RFM2gUnMapUserMemoryBytes()

The RFM2gUnMapUserMemoryBytes() function unmaps a memory byte space mapped by RFM2gUserMemoryBytes().

### Syntax

```
STDRFM2GCALL RFM2gUnMapUserMemoryBytes( RFM2GHANDLE rh,  
                                           volatile void **UserMemoryPtr,  
                                           RFM2G_UINT32 Bytes );
```

### Parameters

**rh** Handle to opened RFM2g device (I).

**UserMemoryPtr** Pointer to mapped RFM2g space (IO).



### NOTE

The volatile keyword must be used to force an implementation to suppress optimization.

**Bytes** The number of bytes originally mapped to UserMemoryPtr (I). Refer to your driver-specific manual for information on using this parameter.

### Return Values

**Success** RFM2G\_SUCCESS

**Failure**

- RFM2G\_NULL\_DESCRIPTOR — rh is NULL.
- RFM2G\_OS\_ERROR — 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 — UserMemoryPtr is NULL.
- RFM2G\_BAD\_PARAMETER\_3 — Bytes 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UINT32 Offset = 0;
RFM2G_UINT32 Bytes = 4096;
volatile char *pUser = NULL; /* Must be volatile */

result = RFM2gUserMemoryBytes(Handle,
    (volatile void **)&pUser, Offset, Bytes);
if (result == RFM2G_SUCCESS)
{
    result = RFM2gUnMapUserMemoryBytes(Handle,
        (volatile void **)&pUser, Bytes);
}
```

## Related Commands

- RFM2gUserMemoryBytes()

## 1.6.6 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 — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

- RFM2gBoardID()
- RFM2gGetConfig()

## 1.6.7 RFM2gBoardID()

The RFM2gBoardID() function returns the RFM2g interface model type. Each RFM2g model type is uniquely identified by a numeric value assigned by Abaco 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 the RFM2gBoardID() number to obtain that value.

### Syntax

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

### Parameters

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

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — BoardIdPtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

- RFM2gNodeID()
- RFM2gGetConfig()

## 1.6.8 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()-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 RFM2gSize( RFM2GHANDLE rh, RFM2g_UINT32 *SizePtr );
```

### Parameters

rh	Handle to opened RFM2g device (I).
SizePtr	Pointer to variable that is filled with the RFM2g size value (O).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — SizePtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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()

## 1.6.9 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

rh	Handle to opened RFM2g device (I).
FirstPtr	Pointer to the variable filled with the offset of the first location of RFM memory (O).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — FirstPtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

- RFM2gGetConfig()
- RFM2gSize()



## 1.6.10 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

rh	Initialized previously with a call to RFM2gOpen() (I).
NamePtr	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — NamePtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_CHAR name[64];  
name[0] = 0;  
  
result = RFM2gDeviceName(Handle, name);  
if(result == RFM2G_SUCCESS)  
{  
    printf("RFM2gDeviceName : %s\n", name);  
}
```

### Related Commands

- RFM2gGetConfig()

## 1.6.11 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

rh	Handle to opened RFM2g device (I).
VersionPtr	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — VersionPtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

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

### Related Commands

- RFM2gDriverVersion()
- RFM2gGetConfig()

## 1.6.12 RFM2gDriverVersion()

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

### Syntax

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

### Parameters

rh	Handle to opened RFM2g device (I).
VersionPtr	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — VersionPtr is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_CHAR drvVersion[64];  
drvVersion[0] = 0;  
  
result = RFM2gDriverVersion(Handle, drvVersion);  
if(result == RFM2G_SUCCESS)  
{  
    printf("RFM2gDriverVersion : %s\n", drvVersion);  
}
```

### Related Commands

- RFM2gDllVersion()
- RFM2gGetConfig()

### 1.6.13 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 RFM2gGetDMAThreshold() function.

#### Syntax

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

#### Parameters

rh	Handle to opened RFM2g device (I).
Threshold	Pointer to the variable where the current DMA threshold value will be copied (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — Threshold is NULL.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

#### Related Commands

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

### 1.6.14 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 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 five 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 six to ten 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

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

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL. RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.6.15 RFM2gGetDMAByteSwap()

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

### Syntax

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

### Parameters

rh	Handle to currently opened RFM2g device (I).
byteSwap	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — byteSwap is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

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

## 1.6.16 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

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

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL byteSwap = RFM2G_TRUE; /* Turn byte swap on */  
  
result = RFM2gSetDMAByteSwap(Handle, byteSwap);
```

### Related Commands

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



## 1.6.17 RFM2gGetPIOByteSwap()

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

Refer to [section 1.7.1 Data Transfer Considerations](#), on page 43 for information on byte swapping and PIO.

### Syntax

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

### Parameters

rh	Handle to currently opened RFM2g device (I).
byteSwap	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — byteSwap is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL byteSwap;  
  
result = RFM2gGetPIOByteSwap(Handle, &byteSwap);
```

### Related Commands

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

## 1.6.18 RFM2gSetPIOByteSwap()

The RFM2gSetPIOByteSwap() function enables or disables byte swapping of PIO 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 [section 1.7.1 Data Transfer Considerations](#), on page 43 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

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

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11.

```
RFM2G_BOOL byteSwap = RFM2G_TRUE /* Turn byte swap on */  
  
result = RFM2gSetPIOByteSwap(Handle, byteSwap);
```

### Related Commands

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

## 1.7 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
<code>RFM2gRead()</code>	Reads one or more bytes starting at an offset in Reflective Memory.
<code>RFM2gWrite()</code>	Writes one or more bytes starting at an offset in Reflective Memory.
<code>RFM2gPeek8()</code> , <code>RFM2gPeek16()</code> , <code>RFM2gPeek32()</code> and <code>RFM2gPeek64()</code>	Reads a single byte, word or longword from an offset in Reflective Memory.
<code>RFM2gPoke8()</code> , <code>RFM2gPoke16()</code> , <code>RFM2gPoke32()</code> and <code>RFM2gPoke64()</code>	Writes a single byte, word or longword to an offset in Reflective Memory.

### 1.7.1 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 [Chapter 2 • \*rfm2g\\_util.c\* Utility Program](#) 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 PowerPC, 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 [section 1.6.16 \*RFM2gSetDMAByteSwap\(\)\*](#), on page 40 and [section 1.6.18 \*RFM2gSetPIOByteSwap\(\)\*](#), on page 42 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 programmed I/O.

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 [section 1.6.15 RFM2gGetDMAByteSwap\(\)](#), on page 39 for more information.

## 1.7.2 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.



### NOTE

See [section Big Endian and Little Endian Data Conversions](#), on page 43 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 [section Using Direct Memory Access \(DMA\)](#), on page 44 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 PIO.

Refer to [section 1.7.1 Data Transfer Considerations](#), on page 43 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

rh	Handle to opened RFM2g device (I).
Offset	Width-aligned offset to Reflective Memory at which to begin the read (I). Valid offset values are 0x0 to 0x3FFFFFFF for 64MB cards, 0x0 to 0x7FFFFFFF for 128MB cards and 0x0 to 0xFFFFFFFF for 256MB cards.
Buffer	Pointer to where data is copied from Reflective Memory (O).
Length	Number of bytes to transfer (I). Valid values are 0 to ([RFM Size] - rfmOffset).

## Return Values

Success	RFM2G_SUCCESS
Failure	<p>RFM2G_NULL_DESCRIPTOR — rh is NULL.</p> <p>RFM2G_OS_ERROR — 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_3 — Buffer is NULL.</p> <p>RFM2G_BAD_PARAMETER_4 — Length is not aligned to data width.</p> <p>RFM2G_OUT_OF_RANGE — Offset and Length is beyond the end of memory on RFM2G device.</p> <p>RFM2G_DMA_FAILED — DMA failed.</p> <p>RFM2G_UNALIGNED_OFFSET — Offset not aligned with data size.</p> <p>RFM2G_READ_ERROR — Data not aligned and/or system error.</p> <p>RFM2G_LSEEK_ERROR — Failure of lseek(2) command.</p> <p>RFM2G_UNALIGNED_ADDRESS — Buffer is not aligned to data width.</p>

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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()

### 1.7.3 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). 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.



#### NOTE

See [section Big Endian and Little Endian Data Conversions](#), on page 43 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 [section Using Direct Memory Access \(DMA\)](#), on page 44 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

rh	Handle to opened RFM2g device (I).
Offset	Width-aligned offset in Reflective Memory at which to begin the write (I). Valid offset values are 0x0 to 0x3FFFFFFF for 64MB cards, 0x0 to 0x7FFFFFFF for 128MB cards and 0x0 to 0xFFFFFFF for 256MB cards.
Buffer	Pointer to where data is copied to Reflective Memory (I).
Length	Number of bytes units to write (I). Valid values are 0 to ([RFM Size] - rfmOffset).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — Buffer is NULL.

RFM2G\_BAD\_PARAMETER\_4 — Length is not aligned to data width.

RFM2G\_OUT\_OF\_RANGE — Offset and Length is beyond the end of memory on RFM2G device.

RFM2G\_DMA\_FAILED — DMA failed.

RFM2G\_UNALIGNED\_OFFSET — Offset 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 — Buffer is not aligned to data width.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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()
- RFM2gGetDMAByteSwap()
- RFM2gSetPIOByteSwap()



## 1.7.4 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 [section Big Endian and Little Endian Data Conversions](#), on page 43 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 [section 1.7.1 Data Transfer Considerations](#), on page 43 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

rh	Handle to opened RFM2g device (I).
Offset	Offset in Reflective Memory from which to read (I).
Value	Pointer to where the value is read from Offset (O).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL. RFM2G_OS_ERROR — 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 — Value is NULL.

RFM2G\_OUT\_OF\_RANGE — Offset is beyond the end of RFM2G memory.

RFM2G\_UNALIGNED\_OFFSET — Offset not aligned with data size.

### Example (RFM2gPeek8())

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11 :

```
RFM2G_UINT64 Value;
RFM2G_UINT32 Offset = 0;

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

### Related Commands

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

## 1.7.5 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 [section Big Endian and Little Endian Data Conversions](#), on page 43 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

rh	Handle to opened RFM2g device (I).
Offset	Offset in Reflective Memory from which to read (I).
Value	Value written to Offset (I).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL. RFM2G_OS_ERROR — 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 — Offset not aligned with data size.

### Example (RFM2gPoke8())

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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 [Section 1.2.2 Routine Code for Use with API Function Examples](#) on page 11:

```
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 [Section 1.2.2 Routine Code for Use with API Function Examples](#) on page 11:

```
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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

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

### Related Commands

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

## 1.8 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
<code>RFM2gEnableEvent()</code>	Enables reception of an RFM2g interrupt event.
<code>RFM2gDisableEvent()</code>	Disables the reception of an RFM2g event.
<code>RFM2gSendEvent()</code>	Transmits the specified RFM2g interrupt event to one or all other RFM2g node IDs.
<code>RFM2gWaitForEvent()</code>	Blocks the calling process until an occurrence of the specified RFM2g interrupt event is received or a timeout (if enabled) expires.
<code>RFM2gEnableEventCallback()</code>	Enables the interrupt notification for one event on one board.
<code>RFM2gDisableEventCallback()</code>	Disables the interrupt notification for one event on one board.
<code>RFM2gClearEvent()</code>	Flushes all pending events for a specified event type.
<code>RFM2gCancelWaitForEvent()</code>	Cancels any pending <b>RFM2gWaitForEvent()</b> calls for the specified event type.
<code>RFM2gClearEventCount()</code>	Clears the event counter for an interrupt event type.
<code>RFM2gGetEventCount()</code>	Gets the number of interrupt events for a given event type.

## 1.8.1 RFM2gEnableEvent()

RFM2g network 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():

Table 1-7 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 opened RFM2g device (I).

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

Event	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 Interrupt	RFM2GEVENT_ROGUE_PKT
	RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
	Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
	Memory Write Inhibited	RFM2GEVENT_MEM_ WRITE_INHIBITED
	Memory Parity Error	RFM2GEVENT_LOCAL_ MEM_PARITY_ERR

## Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — EventType 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.8.2 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():

Table 1-8 RFM2gDisableEvent()

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 opened RFM2g device (I).

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

#### Interrupt

Reset Interrupt  
Network Interrupt 1  
Network Interrupt 2  
Network Interrupt 3  
Network Interrupt 4  
    (Init Interrupt)  
Bad Data Interrupt  
RX FIFO Full Interrupt  
Rogue Packet Detected  
    and Removed Interrupt  
RX FIFO Almost Full Interrupt  
Sync Loss Occurred Interrupt  
Memory Write Inhibited  
  
Memory Parity Error

#### Event ID

RFM2GEVENT\_RESET  
RFM2GEVENT\_INTR1  
RFM2GEVENT\_INTR2  
RFM2GEVENT\_INTR3  
  
RFM2GEVENT\_INTR4  
RFM2GEVENT\_BAD\_DATA  
RFM2GEVENT\_RXFIFO\_FULL  
  
RFM2GEVENT\_ROGUE\_PKT  
RFM2GEVENT\_RXFIFO\_AFULL  
RFM2GEVENT\_SYNC\_LOSS  
RFM2GEVENT\_MEM  
    \_WRITE\_INHIBITED  
RFM2GEVENT\_LOCAL  
    \_MEM\_PARITY\_ERR

### Return Values



Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.
	RFM2G_OS_ERROR — 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 — EventType 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

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

### 1.8.3 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

**rh** Handle to opened RFM2g device (I).

**ToNode** Who will receive the interrupt event (I) (RFM2G\_NODE\_ALL sends the event to all nodes).



#### NOTE

A node cannot send an event to itself.

**EventType** The type of interrupt event to send (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	RFM2GEVENT_INTR4

**ExtendedData** User-defined data (I).

#### Return Values

**Success** RFM2G\_SUCCESS

**Failure**

- RFM2G\_NULL\_DESCRIPTOR — rh is NULL.
- RFM2G\_OS\_ERROR — 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 ToNode.
- RFM2G\_BAD\_PARAMETER\_3 — Invalid EventType.
- RFM2G\_DRIVER\_ERROR — Internal driver error.
- RFM2G\_NOT\_OPEN — Device is not open.
- RFM2G\_NODE\_ID\_SELF — Cannot send event to self.

## Example

The following example code sends user interrupt 1. Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
/* 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()

## 1.8.4 RFM2gWaitForEvent()

The RFM2gWaitForEvent() function allows an application program to determine that an RFM2g interrupt event has been received. The **RFM2gWaitForEvent()** function 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
RFM2GEVENT_BAD_DATA	Bad Data Interrupt
RFM2GEVENT_SYNC_LOSS	Sync Loss Occurred Interrupt



### NOTE

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

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

Table 1-9 RFM2gWaitForEvent()

Existing Condition(s)	RFM2gWaitForEvent() Behavior
The event is enabled.	RFM2G_SUCCESS – Event received RFM2G_TIMED_OUT – Event not received before timeout RFM2G_WAIT_EVENT_CANCELED – User called RFM2gCancelWaitForEvent()
The event is not enabled.	RFM2G_SUCCESS – Event received RFM2G_TIMED-OUT – Event not received before timeout RFM2G_WAIT_EVENT_CANCELED – User called RFM2gCancelWaitForEvent()
RFM2gCancelWaitForEvent() is called for the event.	RFM2G_SUCCESS – Event received RFM2G_TIMED_OUT – Event not received before timeout RFM2G_WAIT_EVENT_CANCELED – User called RFM2gCancelWaitForEvent()
The event is enabled and a callback is registered.	RFM2G_EVENT_IN_USE
Another thread is pending on RFM2gWaitForEvent().	RFM2G_EVENT_IN_USE

### Syntax

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

### Parameters

rh                      Handle to opened 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:

**ExtendedInfo** User data that accompanies an event (O).

**NodeId** RFM node that sent the event (O).

**Event** Specifies which interrupt event to wait upon (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 Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full	
Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred	
Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR

**Timeout** 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:

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.

**pDrvSpec** Driver-specific data that accompanies an event (I). Refer to your driver-specific manual for more information.

## Return Values

Success	RFM2G_SUCCESS
Failure	<p>RFM2G_NULL_DESCRIPTOR — <code>rh</code> is NULL.</p> <p>RFM2G_OS_ERROR — 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 — <code>EventInfo</code> is NULL or <code>EventInfo</code> event has invalid event type.</p> <p>RFM2G_EVENT_IN_USE — Event is already being waited on.</p> <p>RFM2G_TIMED_OUT — Timed out waiting for event.</p> <p>RFM2G_WAIT_EVENT_CANCELED RFM2gCancelWaitForEvent() called for this event.</p> <p>RFM2G_DRIVER_ERROR — Internal driver error.</p>

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
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()

## 1.8.5 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
RFM2GEVENT_BAD_DATA	Bad Data Interrupt
RFM2GEVENT_SYNC_LOSS	Sync Loss Occurred Interrupt



### 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():

Table 1-10 RFM2gEnableEventCallback()

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

### Syntax

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

void rfmSampleFunc( RFM2GHANDLE rh,
                    RFM2GEVENTINFO EventInfo )
```

### Parameters

**rh** Handle to opened 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 Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO _AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM _WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL _MEM_PARITY_ERR

pEventFunc      The address of the function to be called when the event occurs (I).

### Return Values

Success            RFM2G\_SUCCESS

Failure            RFM2G\_NULL\_DESCRIPTOR — rh is NULL.

RFM2G\_OS\_ERROR — 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 EventType.

RFM2G\_BAD\_PARAMETER\_3 — pEventFunc is NULL.

RFM2G\_EVENT\_IN\_USE — Event is already being waited on.

RFM2G\_DRIVER\_ERROR — Internal driver error.

RFM2G\_WAIT\_EVENT\_CANCELED — Wait for event canceled.

### 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

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



## 1.8.6 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_SYNC_LOSS	Sync Loss Occurred Interrupt



### 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():

Table 1-11 RFM2gDisableEventCallback()

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

rh                      Handle to opened 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 Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM_WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL_MEM_PARITY_ERR

## Return Values

Success	RFM2G_SUCCESS
Failure	<p>RFM2G_NULL_DESCRIPTOR — rh is NULL.</p> <p>RFM2G_OS_ERROR — 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 EventType.</p> <p>RFM2G_EVENT_NOT_IN_USE — Event not registered for callback.</p> <p>RFM2G_EVENT_NOT_IN_USE — No callback was set up for the specified event.</p>

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.8.7 RFM2gClearEvent()

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

### Syntax

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

### Parameters

rh                      Handle to opened RFM2g device (I).

EventType              The event FIFO to clear (I).  
Interrupts correlate to the following event IDs:

Interrupt	Event ID
Reset Interrupt	RFM2GEVENT_RESET
Network Interrupt1	RFM2GEVENT_INTR1
Network Interrupt2	RFM2GEVENT_INTR2
Network Interrupt3	RFM2GEVENT_INTR3
Network Interrupt4 (Init Interrupt)	RFM2GEVENT_INTR4
Bad Data Interrupt	RFM2GEVENT_BAD_DATA
RX FIFO Full Interrupt	RFM2GEVENT_RXFIFO_FULL
Rogue Packet Detected and Removed Interrupt	RFM2GEVENT_ROGUE_PKT
RX FIFO Almost Full Interrupt	RFM2GEVENT_RXFIFO_AFULL
Sync Loss Occurred Interrupt	RFM2GEVENT_SYNC_LOSS
Memory Write Inhibited	RFM2GEVENT_MEM _WRITE_INHIBITED
Memory Parity Error	RFM2GEVENT_LOCAL _MEM_PARITY_ERR
All interrupts	RFM2GEVENT_LAST

### Return Values

Success                RFM2G\_SUCCESS

Failure                RFM2G\_NULL\_DESCRIPTOR — rh is NULL.

RFM2G\_OS\_ERROR — 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 EventType.

RFM2G\_DRIVER\_ERROR — Internal driver error.

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.8.8 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 canceled.



### NOTE

A canceled RFM2gWaitForEvent() call returns a value of RFM2G\_WAIT\_EVENT\_CANCELED.

### Syntax

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

### Parameters

rh	Handle to opened RFM2g device (I).
EventType	Specifies which interrupt event to cancel (I). Interrupts correlate to the following event IDs:

#### Interrupt

Reset Interrupt  
Network Interrupt1  
Network Interrupt2  
Network Interrupt3  
Network Interrupt4  
(Init Interrupt)  
Bad Data Interrupt  
RX FIFO Full  
Interrupt  
Rogue Packet  
Detected and Removed  
Interrupt  
RX FIFO Almost Full  
Interrupt  
Sync Loss Occurred  
Interrupt  
Memory Write Inhibited  
  
Memory Parity Error

#### Event ID

RFM2GEVENT\_RESET  
RFM2GEVENT\_INTR1  
RFM2GEVENT\_INTR2  
RFM2GEVENT\_INTR3  
RFM2GEVENT\_INTR4  
  
RFM2GEVENT\_BAD\_DATA  
RFM2GEVENT\_RXFIFO\_FULL  
  
RFM2GEVENT\_ROGUE\_PKT  
  
RFM2GEVENT\_RXFIFO\_AFULL  
  
RFM2GEVENT\_SYNC\_LOSS  
  
RFM2GEVENT\_MEM  
\_WRITE\_INHIBITED  
RFM2GEVENT\_LOCAL  
\_MEM\_PARITY\_ERR

## Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.8.9 RFM2gClearEventCount()

The RFM2gClearEventCount() function clears event counts for a specified event or all events.

### Syntax

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

### Parameters

**rh** Handle to opened RFM2g device (I).

**EventType** The event count to clear (I).  
Interrupts correlate to the following event IDs:

#### Interrupt

Reset Interrupt  
Network Interrupt1  
Network Interrupt2  
Network Interrupt3  
Network Interrupt4  
(Init Interrupt)  
Bad Data Interrupt  
RX FIFO Full  
Interrupt  
Rogue Packet  
Detected and Removed  
Interrupt  
RX FIFO Almost Full  
Interrupt  
Sync Loss Occurred  
Interrupt  
Memory Write Inhibited  
Memory Parity Error  
All interrupts

#### Event ID

RFM2GEVENT\_RESET  
RFM2GEVENT\_INTR1  
RFM2GEVENT\_INTR2  
RFM2GEVENT\_INTR3  
RFM2GEVENT\_INTR4  
  
RFM2GEVENT\_BAD\_DATA  
RFM2GEVENT\_RXFIFO\_FULL  
  
RFM2GEVENT\_ROGUE\_PKT  
  
RFM2GEVENT\_RXFIFO\_AFULL  
  
RFM2GEVENT\_SYNC\_LOSS  
  
RFM2GEVENT\_MEM  
\_WRITE\_INHIBITED  
RFM2GEVENT\_LOCAL  
\_MEM\_PARITY\_ERR  
RFM2GEVENT\_LAST

### Return Values

**Success** RFM2G\_SUCCESS

**Failure** RFM2G\_NULL\_DESCRIPTOR — rh is NULL.

RFM2G\_OS\_ERROR — 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 EventType.

RFM2G\_DRIVER\_ERROR — Internal driver error.

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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



## 1.8.10 RFM2gGetEventCount()

The RFM2gGetEventCount() function returns the event count for a specified event.

### Syntax

```
STDRFM2GCALL RFM2gGetEventCount(RFM2GHANDLE rh,  
                                  RFM2GEVENTTYPE EventType),  
                                  RFM2G_UINT32 *Count);
```

### Example:

```
RFM2G_UINT32 Count;  
result = RFM2gClearEvent(Handle, RFM2GEVENT_INTR1,  
&Count);
```

### Parameters:

rh                      Handle to opened RFM2g device (I).

EventType              The event FIFO to clear (I).  
Interrupts correlate to the following event IDs:

#### Interrupt

Reset Interrupt  
Network Interrupt1  
Network Interrupt2  
Network Interrupt3  
Network Interrupt4  
(Init Interrupt)  
Bad Data Interrupt  
RX FIFO Full  
  Interrupt  
Rogue Packet  
  Detected and Removed  
  Interrupt  
RX FIFO Almost Full  
  Interrupt  
Sync Loss Occurred  
  Interrupt  
Memory Write Inhibited  
Memory Parity Error

#### Event ID

RFM2GEVENT\_RESET  
RFM2GEVENT\_INTR1  
RFM2GEVENT\_INTR2  
RFM2GEVENT\_INTR3  
RFM2GEVENT\_INTR4  
  
RFM2GEVENT\_BAD\_DATA  
RFM2GEVENT\_RXFIFO\_FULL  
  
RFM2GEVENT\_ROGUE\_PKT  
  
RFM2GEVENT\_RXFIFO\_AFULL  
  
RFM2GEVENT\_SYNC\_LOSS  
  
RFM2GEVENT\_MEM  
  \_WRITE\_INHIBITED  
RFM2GEVENT\_LOCAL  
  \_MEM\_PARITY\_ERR

Count                  Pointer to where the event count of the specified event is  
written (O).

## Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 EventType.  RFM2G_DRIVER_ERROR — Internal driver error.

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

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

## 1.9 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 1-12 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.
RFM2gGetDarkOnDark()	Retrieves the current ON/OFF state of the Reflective Memory board's Dark on Dark feature.
RFM2gSetDarkOnDark()	Sets the ON/OFF state of the Reflective Memory board's Dark on Dark feature.
RFM2gClearOwnData()	Returns the state of the Own Data bit and resets the state if set. Calling this function will turn OFF the Own Data LED if ON.
RFM2gGetTransmit()	Retrieves the current ON/OFF state of the Reflective Memory board's transmitter.
RFM2gSetTransmit()	Sets the ON/OFF state of the Reflective Memory board's transmitter.
RFM2gGetLoopback()	Retrieves the current ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
RFM2gSetLoopback()	Sets the ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
RFM2gGetParityEnable()	Retrieves the current ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.
RFM2gSetParityEnable()	Sets the ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.
RFM2gGetMemoryOffset()	Gets the memory offset of the Reflective Memory board.
RFM2gSetMemoryOffset()	Sets the memory offset of the Reflective Memory board.
RFM2gGetSlidingWindow()	Retrieves the base Reflective Memory offset and size of the current sliding window.
RFM2gSetSlidingWindow()	Sets the base Reflective Memory offset of the sliding window.

### 1.9.1 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: <b>"UNKNOWN RFM2G ERROR [%d]"</b> where [%d] is the ErrorCode value.
Failure	A NULL pointer.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## 1.9.2 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

rh	Handle to opened RFM2g device (I).
Led	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 — rh is NULL.  RFM2G_OS_ERROR — 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 — Led is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

### Related Commands

- RFM2gSetLed()

### 1.9.3 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 opened 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 — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

#### Related Commands

- RFM2gGetLed()

## 1.9.4 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 — rh is NULL.

RFM2G\_OS\_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## 1.9.5 RFM2gGetDebugFlags()



### NOTE

Application programs should not use this function unless directed to do so by Abaco 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 <b>ioctl(2)</b> system calls
RFM2G_DBMMAP	Trace <b>mmap(2)</b> system calls
RFM2G_DBOPEN	Trace <b>open(2)</b> system calls
RFM2G_DBCLOSE	Trace <b>close(2)</b> system calls
RFM2G_DBREAD	Trace <b>read(2)</b> system calls
RFM2G_DBWRITE	Trace <b>write(2)</b> 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 <b>syslogd</b> get the message
RFM2G_DBMINPHYS	Trace minphys limits



## Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — <code>rh</code> is NULL.  RFM2G_OS_ERROR — 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 — <code>Flags</code> is NULL.

## Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

## Related Commands

- RFM2gSetDebugFlags()

## 1.9.6 RFM2g SetDebugFlags()



### NOTE

Application programs should not use this function unless directed to do so by Abaco 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 <b>ioctl(2)</b> system calls
RFM2G_DBMMAP	Trace <b>mmap(2)</b> system calls
RFM2G_DBOPEN	Trace <b>open(2)</b> system calls
RFM2G_DBCLOSE	Trace <b>close(2)</b> system calls
RFM2G_DBREAD	Trace <b>read(2)</b> system calls
RFM2G_DBWRITE	Trace <b>write(2)</b> 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 <b>syslogd</b> get the message
RFM2G_DBMINPHYS	Trace minphys limits
RFM2G_DBDMA	Trace DMA calls

### Return Values

Success	RFM2G_SUCCESS
Failure	<p>RFM2G_NULL_DESCRIPTOR — <code>rh</code> is NULL.</p> <p>RFM2G_OS_ERROR — 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>

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UNIT32 lFlag = RFM2G_DBERROR;

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

### Related Commands

- RFM2gGetDebugFlags()

## 1.9.7 RFM2gGetDarkOnDark()

The RFM2gGetDarkOnDark() function retrieves the current ON/OFF state of the Reflective Memory board's Dark On Dark feature.

### Syntax

```
STDRFM2GCALL RFM2gGetDarkOnDark( RFM2GHANDLE rh,  
                                   RFM2G_BOOL *state );
```

### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Dark on Dark feature is written (RFM2G_ON when the Dark On Dark is ON, or RFM2G_OFF when the Dark On Dark is OFF) (O).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — State is NULL.

### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL DarkOnDark;  
result = RFM2gGetDarkOnDark(Handle, &DarkOnDark);
```

### Related Commands

- RFM2gSetDarkOnDark()

## 1.9.8 RFM2gSetDarkOnDark()

The RFM2gSetDarkOnDark() function sets the current ON/OFF state of the Reflective Memory board's Dark On Dark feature.

### Syntax

```
STDRFM2GCALL RFM2gSetDarkOnDark( RFM2GHANDLE rh,  
                                   RFM2G_BOOL state );
```

### Parameters

rh	Handle to opened RFM2g device (I).
state	The state of the Dark On Dark: RFM2G_FALSE=>OFF, RFM2G_TRUE=>ON (I).

### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
result = RFM2gSetDarkOnDark(Handle, RFM2G_TRUE);
```

### Related Commands

- RFM2gGetDarkOnDark()

### 1.9.9 RFM2gClearOwnData()

The RFM2gClearOwnData() function retrieves the current ON/OFF state of the Reflective Memory board's Own Data bit and resets the state if set. Calling this function will turn OFF the Own Data LED if ON.

#### Syntax

```
STDRFM2GCALL RFM2gClearOwnData( RFM2GHANDLE rh,  
                                  RFM2G_BOOL *state );
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Own Data LED feature is written (RFM2G_ON when the Own Data LED is ON, or RFM2G_OFF when the Own Data LED is OFF) (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NOT_OPEN — Device is not open.  RFM2G_BAD_PARAMETER_1 — if the handle is NULL  RFM2G_BAD_PARAMETER_2 — if the state is NULL  RFM2G_OS_ERROR — if ioctl (2) fails

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL OwnData;  
  
result = RFM2gClearOwnData(Handle, OwnData);
```

### 1.9.10 RFM2gGetTransmit()

The RFM2gGetTransmit() function retrieves the current ON/OFF state of the Reflective Memory board's Transmitter.

#### Syntax

```
STDRFM2GCALL RFM2gGetTranmit( RFM2GHANDLE rh,  
                                RFM2G_BOOL *state);
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Transmitter is written (RFM2G_ON when the Transmitter is ON, or RFM2G_OFF when the Transmitter is OFF) (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — State is NULL.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL Transmit;  
  
result = RFM2gGetTransmit(Handle, &Transmit);
```

#### Related Commands

- RFM2gSetTransmit()

### 1.9.11 RFM2gSetTransmit()

The RFM2gSetTransmit() function sets the current ON/OFF state of the Reflective Memory board's Transmitter.

#### Syntax

```
STDRFM2GCALL RFM2gSetTransmit( RFM2GHANDLE rh,  
                                RFM2G_BOOL state);
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	The state of the Transmitter: RFM2G_FALSE=>OFF, RFM2G_TRUE=>ON (I).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
result = RFM2gSetTransmit(Handle, RFM2G_TRUE);
```

#### Related Commands

- RFM2gGetTransmit()



### 1.9.12 RFM2gGetLoopback()

The RFM2gGetLoopback() function retrieves the current ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.

#### Syntax

```
STDRFM2GCALL RFM2gGetLoopback( RFM2GHANDLE rh,  
                                RFM2G_BOOL *state );
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Loopback is written (RFM2G_ON when the Loopback is ON, or RFM2G_OFF when the Loopback is OFF) (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — State is NULL.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL Loopback;  
  
result = RFM2gGetLoopback(Handle, &Loopback);
```

#### Related Commands

- RFM2gSetLoopback()

### 1.9.13 RFM2gSetLoopback()

The RFM2gSetLoopback() function sets the current ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.

#### Syntax

```
STDRFM2GCALL RFM2gSetLoopback( RFM2GHANDLE rh,  
                                RFM2G_BOOL state );
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	The state of the Loopback: RFM2G_FALSE=>OFF, RFM2G_TRUE=>ON (I).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
result = RFM2gSetLoopback(Handle, RFM2G_TRUE);
```

#### Related Commands

- RFM2gSetLoopback()

### 1.9.14 RFM2gGetParityEnable()

The RFM2gGetParityEnable() function retrieves the current ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.

#### Syntax

```
STDRFM2GCALL RFM2gGetParityEnable( RFM2GHANDLE rh,  
                                     RFM2G_BOOL *state );
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Parity Enable is written (RFM2G_ON when the ParityEnable is ON, or RFM2G_OFF when the Parity Enable is OFF) (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 — State is NULL.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_BOOL ParityEnable;  
  
result = RFM2gGetParityEnable(Handle, &ParityEnable);
```

#### Related Commands

- RFM2gSetParityEnable()

### 1.9.15 RFM2gSetParityEnable()

The RFM2gSetParityEnable() function sets the current ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.

#### Syntax

```
STDRFM2GCALL RFM2gSetParityEnable( RFM2GHANDLE rh,  
                                     RFM2G_BOOL state );
```

#### Parameters

rh	Handle to opened RFM2g device (I).
state	Pointer to where the state of the Set Parity Enable feature is written (RFM2G_ON when the Set Parity Enable is ON, or RFM2G_OFF when the Set Parity Enable is OFF) (I).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NULL_DESCRIPTOR — rh is NULL.  RFM2G_OS_ERROR — 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 [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
result = RFM2gSetParityEnable(Handle, RFM2G_TRUE);
```

#### Related Commands

- RFM2gGetParityEnable()

### 1.9.16 RFM2gGetMemoryOffset()

The RFM2gGetMemoryOffset() function retrieves the Reflective Memory board's memory window.

#### Syntax

```
STDRFM2GCALL RFM2gGetMemoryOffset( RFM2GHANDLE rh,  
                                     RFM2G_MEM_OFFSETTYPE *Offset);
```

#### Parameters

rh	Handle to opened RFM2g device (I).
Offset	Pointer to where the current offset to the network address is written (O).

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NOT_OPEN — if the device has not yet been opened.  RFM2G_BAD_PARAMETER_1 — if the handle is NULL  RFM2G_BAD_PARAMETER_2 — if the offset is NULL  RFM2G_OS_ERROR — if IOCTL(2) fails

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_MEM_OFFSETTYPE MemoryOffset;  
  
result = RFM2gGetMemoryOffset(Handle, &MemoryOffset);
```

#### Related Commands

- RFM2gSetMemoryOffset()

### 1.9.17 RFM2gSetMemoryOffset()

The RFM2gSetMemoryOffset() function sets the Reflective Memory board's memory window.

#### Syntax

```
STDRFM2GCALL RFM2gSetMemoryOffset( RFM2GHANDLE rh,  
                                     RFM2G_MEM_OFFSETTYPE offset);
```

#### Parameters

rh                      Handle to opened RFM2g device (I)

Offset                 The offset to the network address (I)

#### Valid Offsets

#### Description

RFM2G_MEM_OFFSET0	1st 64 Megabyte Memory Offset
RFM2G_MEM_OFFSET1	2nd 64 Megabyte Memory Offset
RFM2G_MEM_OFFSET2	3rd 64 Megabyte Memory Offset
RFM2G_MEM_OFFSET3	4th 64 Megabyte Memory Offset

#### Return Values

Success                RFM2G\_SUCCESS

Failure                RFM2G\_NOT\_OPEN — if the device has not yet been opened.

RFM2G\_BAD\_PARAMETER\_1 — if the handle is NULL

RFM2G\_BAD\_PARAMETER\_2 — if the offset is NULL

RFM2G\_OS\_ERROR — if IOCTL(2) fails

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

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

#### Related Commands

- RFM2gGetMemoryOffset()

### 1.9.18 RFM2gGetSlidingWindow()

The RFM2gGetSlidingWindow() function retrieves the base Reflective Memory offset and size of the current sliding memory window.

#### Syntax

```
STDRFM2GCALL RFM2gGetSlidingWindow(RFM2GHANDLE rh,
                                     RFM2G_UINT32 *offset,
                                     RFM2G_UINT32 *size);
```

#### Parameters

rh	Handle to opened RFM2g device (I)
offset	Offset of the current sliding memory window (O)
size	Size of the current sliding memory window in bytes. May be set to NULL if the size is not requested. (O)

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NOT_OPEN - if the device has not yet been opened.  RFM2G_NOT_SUPPORTED - if the Sliding Memory Window feature is not supported by the Reflective Memory board's firmware.  RFM2G_BAD_PARAMETER_1 - if the handle is NULL  RFM2G_BAD_PARAMETER_2 - if the offset is NULL  RFM2G_OS_ERROR - if register accesses fail

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UINT32  offset, size;

Result = RFM2gGetSlidingWindow(Handle, &offset,
                                &size);
```

#### Related Commands

- RFM2gSetSlidingWindow()

### 1.9.19 RFM2gSetSlidingWindow()

The RFM2gSetSlidingWindow() function sets the base Reflective Memory offset of the sliding memory window. The size of the sliding window is set with switches or jumpers on the Reflective Memory board. The available window sizes are 2 MByte, 16 MByte, 64 MByte, and the total memory size. The offset of the sliding window must be a multiple of the size of the sliding window.

#### Syntax

```
STDRFM2GCALL RFM2gSetSlidingWindow(RFM2GHANDLE rh,  
                                     RFM2G_UINT32 offset);
```

#### Parameters

rh	Handle to opened RFM2g device (I)
offset	Offset of the sliding memory window (I)

#### Return Values

Success	RFM2G_SUCCESS
Failure	RFM2G_NOT_OPEN - if the device has not yet been opened.  RFM2G_NOT_SUPPORTED - if the Sliding Memory Window feature is not supported by the Reflective Memory board's firmware.  RFM2G_BAD_PARAMETER_1 - if the handle is NULL  RFM2G_BAD_PARAMETER_2 - if the offset is not a multiple of the window size.  RFM2G_OUT_OF_RANGE - if the offset and window size exceed the Reflective Memory board's memory size.  RFM2G_OS_ERROR - if register accesses fail  RFM2G_DRIVER_ERROR - if the driver returns an invalid window size.

#### Example

Use the following code by inserting it into the example routine in [section 1.2.2 Routine Code for Use with API Function Examples](#), on page 11:

```
RFM2G_UINT32 offset = 0x00400000; /* A valid offset  
for a 2MB window */  
  
Result = RFM2gSetSlidingWindow(Handle, offset);
```

#### Related Commands

- RFM2gGetSlidingWindow()



## 2 • rfm2g\_util.c Utility Program

### 2.1 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.

### 2.2 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.

#### 2.2.1 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 are not 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 >
```

## 2.2.2 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>.



### NOTE

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.

## 2.3 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.
checking	Returns the fiber ring continuity through all nodes in a ring.
clear event	Clear any pending events for a specified event.
clear event count	Clear one or all interrupt event counts.
clear own data	The Reflective Memory board's Own Data LED was OFF.
config	Display RFM2g board configuration information.
devname	Returns the device name associated with a RFM2g handle.
disable event	Disables the reception of a specified RFM2g interrupt event.
disable callback	Disables the interrupt notification for a specified event notification.
dllversion	Returns the DLL or library version.
driver version	Returns the RFM2g device driver version.
drv specific	Enter a driver-specific menu.
dump	Peek and display an area of Reflective Memory.
enable event	Enable the receiving of a specified RFM2g event.
enable callback	Enables the interrupt notification for a specified event.
error msg	Prints a text string describing an error code.
exit	Terminate the command line interpreter program.
first	Returns the first available RFM2g offset.
get dark on dark	Returns the state of the RFM2g Dark on Dark feature.
get debug	Retrieves a copy of all RFM2g device driver debug message class control flags.
get dma byteswap	Returns the state of DMA byte swapping hardware.
get event count	Retrieves the count of received interrupt events.
get led	Retrieves the current ON/OFF state of the Reflective Memory board's STATUS LED.
get memory offset	Gets the memory offset of the Reflective Memory board.

Table 2-1 RFM2g Driver Commands (Continued)

Command	Description
getloopback	Retrieves the current ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
getparityenable	Retrieves the current ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.
getpiobyteswap	Returns the state of PIO byte swapping hardware.
getslidingwindow	Get the offset and size of the current Sliding Window.
getthreshold	Returns the current DMA threshold value.
gettransmit	Retrieves the current ON/OFF state of the Reflective Memory board's transmitter.
help	Display command help.
mapuser	Retrieves RFM2g memory information or maps RFM2g memory to the user space.
mapuserbytes	Retrieves RFM2g memory byte information or maps RFM2g memory bytes 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	Peek an element from RFM2g memory.
performancetest	Display the speed of system reads and writes.
poke8, poke16, poke32 and poke64	Poke an element to RFM2g memory.
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.
setdarkondark	Sets the ON/OFF state of the Reflective Memory board's Dark on Dark feature.
setdebug	Sets the driver's debug display word (i.e. debug flags).
setdmabyteswap	Sets the ON/OFF state of DMA byte swapping hardware.
setled	Sets the current ON/OFF state of the Reflective Memory board's STATUS LED.
setloopback	Sets the ON/OFF state of the Reflective Memory board's loopback of the transmit signal to the receiver circuit internally.
setmemoryoffset	Sets the memory offset of the Reflective Memory board.
setparityenable	Sets the ON/OFF state of the Reflective Memory board's parity checking on all onboard memory accesses.
setpiobyteswap	Sets the ON/OFF state of PIO byte swapping hardware.
setslidingwindow	Set the offset of the Sliding Window.

Table 2-1 RFM2g Driver Commands (Continued)

Command	Description
setthreshold	Sets the transfer size at which reads and writes will use DMA.
settransmit	Sets the ON/OFF state of the Reflective Memory board's transmitter.
size	Returns the total amount of virtual memory space available on the RFM2g device.
unmapuser	Unmaps RFM2g buffer memory from the user space.
unmapuserbytes	Unmaps RFM2g buffer memory from the user space.
wait	Blocks the calling process until an occurrence of an RFM2g interrupt event is received or a timeout expires.
write	Writes a value starting at the specified offset in Reflective Memory.

### 2.3.1 boardid

The **boardid** command returns the RFM2g interface model type. Each RFM2g interface model type is uniquely identified by a numeric value assigned by Abaco Systems 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 utility calls `RFM2gBoardID()` to obtain the RFM2g board ID.

#### Syntax

```
boardid
```

#### Example

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

### 2.3.2 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 canceled. 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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11
All Events	12

#### Example

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

### 2.3.3 checkring

The **checkring** 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

```
checkring
```

#### Example

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

### 2.3.4 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	RX FIFO ALMOST FULL
9	SYNC LOSS
10	MEM WRITE INHIBITED
11	LOCAL MEM PARITY ERROR
12	ALL EVENTS

#### Example

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



### 2.3.5 cleareventcount

The **cleareventcount** command clears event counts for a specified event or all events. The utility calls `RFM2gClearEventCount()` function.

#### Syntax

```
cleareventcount event
```

#### Parameters

event                      Where “event” is one of the following interrupts events [0-12]:

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	RX FIFO FULL
7	ROGUE PACKET
8	RX FIFO ALMOST FULL
9	SYNC LOSS
10	MEM WRITE INHIBITED
11	LOCAL MEM PARITY ERROR
12	ALL EVENTS

#### Example

```
Please enter device Number: 0
UTIL0 > cleareventcount 0
The "RESET" interrupt event count cleared.
UTIL0 >
```

### 2.3.6 clearowndata

The **clearowndata** command turns the Reflective Memory board’s Own Data LED OFF. The utility calls `RFM2gClearOwnData()` function.

#### Syntax

```
clearowndata
```

#### Example

```
UTIL0 > cleareventcount
The Reflective Memory board’s Own Data LED was turned
off.
UTIL0 >
```

### 2.3.7 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 >
```

### 2.3.8 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 >
```

### 2.3.9 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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11

#### Example

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

## 2.3.10 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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11

### Example

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

## 2.3.11 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 >
```

### 2.3.12 driverversion

The **driverversion** command displays an ASCII string showing the Abaco 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 >
```

### 2.3.13 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 >
```

## 2.3.14 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 [section Big Endian and Little Endian Data Conversions](#), on page 43 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 64MB cards, 0x0 to 0x7FFFFFFF for 128MB cards or 0x0 to 0x0FFFFFFF for 256MB 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 128MB Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit longword	16777216 (0x1000000)

### Example

```
UTIL0 > dump 0 8 4
                                0                                1
0x00000000:0123456789ABCDEF 0123456789ABCDEF | .#Eg.#Eg ..... |
0x00000010:0123456789ABCDEF 0123456789ABCDEF | .#Eg.#Eg ..... |

UTIL0 >
```

### 2.3.15 enableevent

RFM2g network event interrupts are not enabled by default. However, RFM2g error event interrupts are enabled by default so they will be counted. The user can choose to disable the error events, in which case they will stay that way until the user enables them or the driver is unloaded and reloaded. 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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11

#### Example

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

## 2.3.16 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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11

### Example

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



### 2.3.17 errmsg

The **errmsg** 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

```
errmsg ErrorCode
```

## Parameters

ErrorCode	Return code from an API function (I).
-----------	---------------------------------------

### Example

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

### 2.3.18 exit

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

## Syntax

```
exit
```

### Example

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

### 2.3.19 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
```

### 2.3.20 getdarkondark

The **getdarkondark** command returns the state of the RFM2g Dark on Dark feature. The utility calls `RFM2gGetDarkOnDark()` to retrieve the state of the Dark on Dark feature.

#### Syntax

```
getdarkondark
```

#### Example

```
Please enter device number: 0
UTIL0 > getdarkondark
    The Reflective Memory board's Dark On Dark feature is
    turned Off.
UTIL0 >
```

### 2.3.21 getdebug



#### NOTE

Users should not use this command unless directed to do so by Abaco 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 >
```

### 2.3.22 getdmabyteswap

The **getdmabyteswap** command returns the state of the DMA 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 >
```

### 2.3.23 geteventcount

The **geteventcount** command displays the current event count for the specified event or all events. The utility calls `RFM2gGetEventCount()` to retrieve event count for the specified event or all events.

#### Syntax

```
geteventcount
```

#### Parameters

event                      Where “event” is one of the following interrupt events [0-12]:

Interrupt	Event ID
Reset	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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11
All Events	12

#### Example

```
UTIL0 > geteventcount 1
The event count for Network INT 1 is 0
UTIL0 >
```

### 2.3.24 getled

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.

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.

#### Syntax

```
getled
```

#### Example

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

### 2.3.25 getmemoryoffset

The **getmemoryoffset** command displays the network memory offset of the Reflective Memory board. The utility calls `RFM2gGetMemoryOffset()` to retrieve the network memory offset.

#### Syntax

```
getmemoryoffset
```

#### Example

```
UTIL0 > getmemoryoffset
The Reflective Memory board's memory offset is
0x00000000.
UTIL0 >
```

### 2.3.26 getloopback

The **getloopback** command displays the state of the RFM2g transmit loopback hardware. The utility calls `RFM2gGetLoopback()` to return the transmit loopback state.

#### Syntax

```
getloopback
```

#### Example

```
UTIL0 > getloopback
The Reflective Memory board's transmit loopback is Off.
UTIL0 >
```

### 2.3.27 getparityenable

The **getparityenable** command displays the state of the RFM2g's parity enable. The utility calls `RFM2gGetParityEnable()` to display the parity enable state.

#### Syntax

```
getparityenable
```

#### Example

```
UTIL0 > getparityenable
The Reflective Memory board's parity enable is Off.
UTIL0 >
```

### 2.3.28 getpiobyteswap

The **getpiobyteswap** command displays the state of PIO 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 >
```

### 2.3.29 getslidingwindow

The **getslidingwindow** command displays the offset and size of the current sliding memory window. The utility calls `RFM2gGetSlidingWindow()` to retrieve the information.

#### Syntax

```
getslidingwindow
```

#### Example

```
UTIL0 > getslidingwindow
The 2 MByte Sliding Window begins at offset 0x00400000.
UTIL0 >
```

### 2.3.30 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 >
```

### 2.3.31 gettransmit

The **gettransmit** command displays status of the Reflective Memory board's transmitter. The utility calls `RFM2gGetTransmit()` to display the state of the Reflective Memory board's transmitter.

#### Syntax

```
gettransmit
```

#### Example

```
UTIL0 > gettransmit
The Reflective Memory board's Transmitter is On.
UTIL0 >
```

### 2.3.32 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

command	The command help to display (I). Entering <b>help</b> displays a list of all commands for which help is available. Entering <b>help</b> 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
checkring	
clearevent	event
cleareventcount	event
clearowndata	
config	
devname	
disableevent	event
disablecallback	event
dllversion	
driverversion	
drvspecific	
dump	offset width length
enableevent	event
enablecallback	event

Press ENTER for more commands ...

COMMAND	PARAMETERS
-----	
errormsg	ErrorCode
exit	
first	
getdarkondark	
getdebug	

getdmabyteswap	
geteventcount	event
getled	
getmemoryoffset	
getloopback	
getparityenable	
getpiobyteswap	
getthreshold	
gettransmit	
help	[command]
mapuser	offset pages

Press ENTER for more commands ...

COMMAND	PARAMETERS
-----	
mapuserbytes	offset bytes
memop	pattern offset width length verify
	float patterntype
nodeid	
peek8	offset
peek16	offset
peek32	offset
peek64	offset
performancetest	
poke8	value offset
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]

Press ENTER for more commands ...

COMMAND	PARAMETERS
-----	
setdarkondark	state
setdebug	flag
setdmabyteswap	state
setled	state
setloopback	state
setmemoryoffset	offset
setparityenable	state
setpiobyteswap	state
setthreshold	value
settransmit	state
size	
unmapuser	UserMemoryPtr pages
unmapuserbytes	UserMemoryPtr bytes
wait	event timeout
write	value offset width length

UTIL0 >

### 2.3.33 mapuser

The **mapuser** command allows the user to map RFM memory pages to the user space.

The utility calls `RFM2gUserMemory()` to map RFM2g memory. Refer to driver specific manual for additional functionality.



#### NOTE

When using the RFM2g-DRV-LNX Linux version of the driver, the mapped area must be reserved by booting Linux with the Linux mem command.

#### Syntax

```
mapuser offset pages
```

#### Parameters

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

#### Examples

The following example displays the values of the mapped region:

```
UTIL0 > mapuser
mapuser: Get Set the User buffer offset and pages
Usage:  mapuser          offset pages
        "offset" is the beginning offset to map.
        "pages" is the number of pages of memory to map.
UTIL0 >
```

The following example maps a buffer that begins at offset 0 and is 100 system pages long and Linux was booted with the mem setting mem=63M:

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



### 2.3.34 mapuserbytes

The **mapuserbytes** command allows the user to map RFM memory to the user space.

The utility calls `RFM2gMapUserBytes()` to map RFM2g memory bytes. Refer to driver specific manual for additional functionality.

#### Syntax

```
mapuserbytes offset bytes
```

#### Parameters

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

#### Examples

The following example displays the values of the mapped region:

```
UTIL0 > mapuserbytes
mapuserbytes: Get Set the User buffer offset and pages
Usage: mapuserbytes          offset bytes
      "offset" is the beginning offset to map.
      "bytes" is the number of bytes of memory to map.
UTIL0 >
```

The following example maps a buffer that begins at offset 0 and is 100 system passes long and Linux was booted with the mem setting mem=63M:

```
UTIL0 > mapuserbytes 0 0x64000
RFM2gUserMemoryByte assigned UserMemoryPtr = 0x50000000
UTIL0 >
```

### 2.3.35 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

pattern	The pattern to write or verify (I).
offset	Width-aligned offset in Reflective Memory at which to begin the read or verify (I). Valid offset values are 0x0 to 0x3FFFFFFF for 64MB cards, and 0x0 to 0x7FFFFFFF for 128MB 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 write or verify (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 128MB Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit longword	16777216 (0x1000000)

verify	Writes (0) or verifies (1) the pattern in Reflective Memory.
float	Specifies whether the pattern is (1) or is not (0) a floating point value.
patterntype	Specifies the pattern 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
```

## 2.3.36 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 >
```

### 2.3.37 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 the [section \*Big Endian and Little Endian Data Conversions\*](#), on page 43 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 >
```

### 2.3.38 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
```

```
Abaco 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 >
```

### 2.3.39 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 [section \*Big Endian and Little Endian Data Conversions\*](#), on page 43 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 >
```

## 2.3.40 quit

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

### Syntax

```
quit
```

### Example

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

## 2.3.41 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.



### NOTE

See [section Big Endian and Little Endian Data Conversions](#), on page 43 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 0x3FFFFFFF for 64MB cards, and 0x0 to 0x7FFFFFFF for 128MB 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  $[\text{buffer size}] / \text{width}$ . 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 128MB Cards
bytes	134217728 (0x8000000)
words	67108864 (0x4000000)
32-bit longword	33554432 (0x2000000)
64-bit longword	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 >
```

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



### 2.3.43 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 >
```

## 2.3.44 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

`event` The type of interrupt event to send (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	4

`tonode` Who will receive the interrupt event (I) (-1 sends the event to all nodes).



### NOTE

A node cannot send an event to itself.

`ext_data` User-defined 32-bit extended data to send (I).

### Example

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

## 2.3.45 setdarkondark

The **setdarkondark** command sets the state of the RFM2g Dark on Dark feature. The utility `RFM2gSetDarkOnDark( )` sets the state of the Dark on Dark feature.

### Syntax

```
setdarkondark state
```

### Parameters

**state**                      Sets the state of Dark on Dark feature, which is one of the following (I):

State	Description
0	Turns Dark On Dark OFF
1	Turns Dark On Dark ON

### Example

```
UTIL0 > setdarkondark 1
      The Reflective Memory board's Dark On Dark feature is
      turned ON.
UTIL0 >
```

## 2.3.46 setdebug



### NOTE

Application programs should not use this command unless directed to do so by Abaco 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.

flag New debug flags (I). Valid strings are:

String	Description
allflags	Turns all debug flags ON
close	Trace <b>close(2)</b> system calls
dma	Trace DMA calls
error	Report critical errors
init	Trace device probing and search
intr	Trace interrupt service
ioctl	Trace <b>ioctl(2)</b> system calls
minphys	Trace minphys limits
mmap	Trace <b>mmap(2)</b> system calls
mutex	Trace synchronization and locking
not_intr	Trace non-RFM interrupt service
open	Trace <b>open(2)</b> system calls
peek	Trace peeks
poke	Trace pokes
read	Trace <b>read(2)</b> system calls
slow	Let <b>syslogd</b> get the message
strat	Trace read/write strategy
timer	Trace interval timer
trace	Trace subroutine entry/exit
write	Trace <b>write(2)</b> system calls

### Example

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

### 2.3.47 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

**state** 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 >
```

### 2.3.48 settled

The **settled** 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

```
settled state
```

#### Parameters

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

#### Example

```
UTIL0 > settled 1
The Reflective Memory board's status LED is On.
UTIL0 >
```

### 2.3.49 setloopback

The **setloopback** command sets the state of the RFM2g transmit loopback hardware. The utility calls `RFM2gSetLoopback()` to set the transmit loopback state.

#### Syntax

```
setloopback state
```

#### Parameters

**state** Sets the state of transmit loopback hardware, which is one of the following (I):

State	Description
0	Turns loopback OFF
1	Turns loopback ON

#### Example

```
UTIL0 > setloopback 0
The Reflective Memory board's transmit loopback is Off.
UTIL0 >
```

### 2.3.50 setmemoryoffset

The **setmemoryoffset** command sets the memory offset status of the Reflective Memory board. The utility calls `RFM2gSetMemoryOffset()` to set the memory offset.

#### Syntax

```
setmemoryoffset offset
```

#### Parameters

**offset** The offset to the network address description.

0	No offset
1	0x4000000
2	0x8000000
3	0xc000000

#### Example

```
UTIL0 > setmemoryoffset 1
The Reflective Memory board's memory offset is 0x40000000.
UTIL0 >
```

### 2.3.51 setparityenable

The **setparityenable** command sets the state of the RFM2g's parity enable. The utility `RFM2gSetParityEnable()` sets the state of the parity enable.

#### Syntax

```
setparityenable state
```

#### Parameters

state                      Sets the state of Parity Enable, which is one of the following (I):

State	Description
0	Turns Parity Enable OFF
1	Turns Parity Enable ON

#### Example

```
UTIL0 > setparityenable 0
        The Reflective Memory board's parity enable is Off.
UTIL0 >
```

### 2.3.52 setpiobyteswap

The **setpiobyteswap** command enables or disables byte swapping PIO 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 >
```

### 2.3.53 setslidingwindow

The **setslidingwindow** command sets the base Reflective Memory offset of the sliding memory window. The size of the sliding window is set with switches or jumpers on the Reflective Memory board. The available window sizes are 2 MByte, 16 MByte, 64 MByte, and the total memory size. The offset of the sliding window must be a multiple of the size of the sliding window. The utility calls `RFM2gGetSlidingWindow()` to get the window size, and `RFM2gSetSlidingWindow()` to set the offset.

#### Syntax

```
setslidingwindow offset
```

#### Example

```
UTIL0 > setslidingwindow 0x00400000
The 2 MByte Sliding Window begins at offset 0x00400000.
UTIL0 >
```

### 2.3.54 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 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 five 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 six to ten 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 >
```



### 2.3.55 settransmit

The **settransmit** command sets the state of the RFM2g's transmitter. The utility `RFM2gSetTransmit()` sets the state of the Reflective Memory board's transmitter.

#### Syntax

```
settransmit state
```

#### Parameters

**state** Sets the state of Transmitter, which is one of the following (I):

State	Description
0	Turns Transmitter OFF
1	Turns Transmitter ON

#### Example

```
UTIL0 > settransmit 1
        The Reflective Memory board's Transmitter is On.
UTIL0 >
```

### 2.3.56 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 an 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 >
```

### 2.3.57 unmapuser

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 >
```

### 2.3.58 unmapuserbytes

The **unmapuserbytes** command unmaps the RFM2g memory buffer from user memory space. The utility calls RFM2gUnMapUserBytes() to unmap the RFM2g memory bytes buffer.

#### Syntax

```
unmapuserbytes UserMemoryPtr bytes
```

#### Parameters

UserMemoryPtr Pointer returned by the **mapuserbytes** command (O).

Pages The number of pages mapped by the **mapuserbytes** command (O).

#### Example

```
UTIL0 > unmapuserbytes 0x50000000 100  
UTIL0 >
```

## 2.3.59 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

**event**                      The type of interrupt event on which to wait (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
RX FIFO Almost Full	8
SYNC Loss	9
Mem Write Inhibited	10
Local Mem Parity Error	11

**timeout**                   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 >
```

## 2.3.60 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.



### NOTE

See [section Big Endian and Little Endian Data Conversions](#), on page 43 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

value	Byte, word or longword value to write to the range specified by offset, length and width (I).
offset	Width-aligned offset in Reflective Memory at which to begin the write (I). Valid offset values are 0x0 to 0x3FFFFFFF for 64MB cards, and 0x0 to 0x7FFFFFFF for 128MB 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 write (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 128MB Cards
bytes	134217728 (0x80000000)
words	67108864 (0x40000000)
32-bit longword	33554432 (0x20000000)
64-bit longword	16777216 (0x10000000)

### Example

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

## 2.4 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.

### 2.4.1 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
Please define DEVICE_PREFIX for your driver.
```

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
Linux	-DRFM2G_LINUX
Windows	None

## 3 • RFM2g Sample Applications

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.

### 3.1 `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.

### 3.2 `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.

### 3.3 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.

### 3.4 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 **rfm2g\_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!
```



## 3.5 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!
```

© 2016 Abaco Systems, Inc.  
All rights reserved.

\* indicates a trademark of Abaco Systems, Inc. and/or its affiliates. All other trademarks are the property of their respective owners.

Confidential Information - This document contains Confidential/Proprietary Information of Abaco Systems, Inc. and/or its suppliers or vendors. Distribution or reproduction prohibited without permission.

THIS DOCUMENT AND ITS CONTENTS ARE PROVIDED "AS IS", WITH NO REPRESENTATIONS OR WARRANTIES OF ANY KIND, WHETHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO WARRANTIES OF DESIGN, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. ALL OTHER LIABILITY ARISING FROM RELIANCE ON ANY INFORMATION CONTAINED HEREIN IS EXPRESSLY DISCLAIMED.

## Abaco Systems Information Centers

### Americas:

1-866-652-2226 (866-OK-ABACO)  
or 1-256-880-0444 (International)

### Europe, Middle East and Africa:

+44 (0) 1327 359444

## Additional Resources

For more information, please visit the Abaco Systems website at:

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

