



# Extreme Manufacturing

*API User's Guide*



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

# Contents

<b>Chapter 1 Introduction</b>	<b>4</b>
1.1 API Description	4
<b>Chapter 2 Platform Requirements</b>	<b>5</b>
2.1 Windows	5
2.2 Linux	5
2.3 Solaris	5
<b>Chapter 3 Device Discovery and Manipulation</b>	<b>6</b>
3.1 XMFG_Rescan_Targets	6
3.2 XMFG_Copy_Device_Data	6
3.3 XMFG_Set_Device_Value	7
3.4 XMFG_Refresh_Device_Value	8
<b>Chapter 4 Canned Test Functions</b>	<b>9</b>
4.1 XMFG_Multithreaded_Canned_Test	9
4.2 XMFG_Start_Stop_Test	12
4.3 XMFG_Corrupt_Media	13
4.4 XMFG_Canned_Test_Control	14
4.5 XMFG_Download_Microcode	15
4.6 XMFG_Reset	16
<b>Chapter 5 Issuing Commands</b>	<b>17</b>
5.1 XMFG_Issue_SCSI_Command	17
5.2 XMFG_Issue_ATA_Command	18
5.3 XMFG_Issue_SAS_Command (Coming Soon)	19
<b>Chapter 6 Buffer Functions</b>	<b>20</b>
6.1 XMFG_Fill_Buffer	20
6.2 XMFG_Resize_Buffer	21
6.3 XMFG_Embed_Write_Buffer	22
<b>Chapter 7 Error Handling</b>	<b>23</b>
7.1 Function Prototypes for Error Handlers	23
7.2 Registering Error Handlers	27
7.3 Implementing Error Handlers	28
<b>Chapter 8 Convenience Functions</b>	<b>29</b>
8.1 XMFG_Send_Email	29
8.2 XMFG_FTP_Login	30
8.3 XMFG_FTP_Logout	31
8.4 XMFG_FTP_Put	31

8.5 XMFG_FTP_Change_Directory_____	32
8.6 XMFG_FTP_Get_____	32
8.7 XMFG_IO_PORT_Open _____	33
8.8 XMFG_IO_PORT_Close _____	33
8.9 XMFG_IO_PORT_Send_____	34
8.10 XMFG_IO_PORT_Receive _____	35



## Chapter 1 Introduction

### 1.1 API Description

This document is intended to define an Application Programming Interface (API) for using the most powerful, unique components of Extreme Manufacturing in a 'C', 'C++' or Visual Basic Environment. Described herein are a set of functions and defines that will enable simple integration of the core of our flagship testing product into any application.

The API described is applicable to installations supporting Microsoft Windows, Linux and Solaris.



## Chapter 2 Platform Requirements

### 2.1 Windows

- Supports Microsoft Windows 2000, XP, 2003 and Vista family platforms.

### 2.2 Linux

- Supports 2.4.x, 2.5.x, and 2.6.x Linux kernels.

### 2.3 Solaris

- Supports Solaris 8.x and greater



## Chapter 3 Device Discovery and Manipulation

### 3.1 XMFG\_Rescan\_Targets

#### 3.1.1 Description

This API Function rescans all targets connected to the system and returns the number of devices found.

#### 3.1.2 Input

There are no input parameters for this function

#### 3.1.3 Output

The number of devices found is returned as an integer.

#### 3.1.4 Example

```
int devices
devices = XMFG_Rescan_Targets();
```

### 3.2 XMFG\_Copy\_Device\_Data

#### 3.2.1 Description

This API Function copies the device structure data from the DLL to the application for the specified device.

#### 3.2.2 Input

**int source\_idx** - The source index of the device structure to copy from the DLL

**XMFG\_device\_info \*\*dest\_device** - The destination device structure element that will receive the data copied from the DLL device structure.

#### 3.2.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

#### 3.2.4 Example

```
int x;
for(x=0;x<devices;x++)
{
    device[x] = NULL; //If a NULL is passed into this function it will malloc the space
                    //needed for the structure
    XMFG_Copy_Device_Data(x,&device[x]);
}
```

## 3.3 XMFG\_Set\_Device\_Value

### 3.3.1 Description

This API Function copies the device structure data from the DLL to the application for the specified device.

### 3.3.2 Input

**int device\_index** - The index of the device whose value will be set

**int value\_type** - The Type of value to be set. Can be any one of the following defines:

```
//value_type defines
XMFG_COMMAND_TIMEOUT           //value is in milliseconds
XMFG_RETRY_LIMIT                //value is the maximum number of command retries
XMFG_DEVICE_STATE               //use the device states from XMDLL.h
XMFG_DEVICE_READ_CACHE         //used to change the device's read cache settings
XMFG_DEVICE_WRITE_CACHE        //used to change the device's write cache settings
XMFG_SOFT_ERROR_LIMIT          //used to change the device's soft error limit
XMFG_DEVICE_HARD_LIMIT         //used to change the device's hard error limit
XMFG_DEVICE_OTHER_LIMIT        //used to change the device's other error limit
XMFG_EXPECTED_SCSI_STATUS      //used to change the device's expected SCSI Status
XMFG_ERROR_COUNT                //used to change the device's error count
XMFG_HARD_ERRORS                //used to change the device's hard error count
XMFG_SOFT_ERRORS                //used to change the device's soft error count
XMFG_COMPARE_ERRORS            //used to change the device's compare error count
XMFG_BYTES_WRITTEN              //used to change the device's bytes written count
XMFG_BYTES_READ                 //used to change the device's bytes read count
XMFG_BYTES_VERIFIED            //used to change the device's bytes verified count
XMFG_IO_COUNT                   //used to change the device's I/O count
```

**int value** - Contains the desired value for the specified value type. Use XFG\_CACHE\_OFF or XMFG\_CACHE\_ON for CACHE value types.

### 3.3.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 3.3.4 Example

```
//Turns on Read Cache on the device specified by index idx
XMFG_Set_Device_Value(idx,
                       XMFG_DEVICE_READ_CACHE,
                       XMFG_CACHE_ON);
```

```
//Sets the command timeout to 10s (10000ms) on the device specified by index idx
XMFG_Set_Device_Value(idx,
                       XMFG_COMMAND_TIMEOUT,
                       10000);
```

## 3.4 XMFG\_Refresh\_Device\_Value

### 3.4.1 Description

This API Function copies the device structure data from the DLL to the application for the specified device and the specified value type.

### 3.4.2 Input

**int device\_index** - The index of the device whose value will be refreshed

**int value\_type** - The Type of value to be set. Can be any one of the following defines:

```
//value_type defines
XMFG_INQUIRY_DATA           //Refreshes any updated inquiry data
XMFG_CAPACITY_INFORMATION   //Refreshes device capacity information
XMFG_GROWN_LIST             //Refreshes the device's Grown List entry count
```

### 3.4.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 3.4.4 Example

```
//Refreshes any updated inquiry data
XMFG_Refresh_Device_Value( idx,
                           XMFG_INQUIRY_DATA);
```



## Chapter 4 Canned Test Functions

### 4.1 XMFG\_Multithreaded\_Canned\_Test

#### 4.1.1 Description

This API Function executes a multithreaded canned test with user specified options. This test only works on disk drives.

#### 4.1.2 Input

**int device\_index** - The index of the device that will run the test.

**int test\_type** - The type of test to be run. Can be any one of the following defines:

```
//test_type defines
XMFG_READ_TEST           //Read testing
XMFG_WRITE_TEST          //Write testing
XMFG_WRITE_READ_TEST     //Write / Read Testing
XMFG_SEEK_TEST           //Seek Testing (SCSI Only)
XMFG_VERIFY_TEST         //Verify Testing (SCSI Only)
```

**int test\_convention** - Contains the convention (style) of I/O to be issued. Can be any one of the following defines:

```
//test_convention defines
XMFG_SEQUENTIAL          //Sequential I/O
XMFG_RANDOM              //Random I/O
XMFG_OSCILLATING         //I/O to inner and outer logical block addresses of the media
XMFG_BUTTERFLY           //Butterfly style I/O
XMFG_3RDSTROKE          //I/O that cuts the media size into 3rds and issues at each 3rd
```

**int test\_limit** - Determines the end condition for the test. Can be any one of the following defines:

```
//test_limit defines
XMFG_TIME_LIMIT          //I/O will end after the user specified amount of time (seconds)
XMFG_IO_LIMIT            //I/O will end after the user specified amount of I/O's are issued
XMFG_ENTIRE_MEDIA        //I/O will end when the end of the media is reached
```

**\_\_int64 test\_limit\_value** - Specifies the value to be used for the selected test\_limit define.

**int test\_flags** - Determines the option flags to be used for the test. Can be combinations of the following defines:

*//test\_limit defines*

```

XMFG_DATA_COMPARE           //Only valid for XMFG_READ_TEST and XMFG_WRITE_READ_TEST
XMFG_EMBED_START_LBA       //Only valid for XMFG_WRITE_TEST and XMFG_WRITE_READ_TEST
XMFG_EMBED_END_LBA         //Only valid for XMFG_WRITE_TEST and XMFG_WRITE_READ_TEST
XMFG_FORCE_UNIT_ACCESS     //Only valid for READ, WRITE and WRITE_READ TESTS
XMFG_WRITE_VERIFY          //Only valid for XMFG_WRITE_TEST and XMFG_WRITE_READ_TEST
XMFG_RAW_IO                 /*Must be used exclusive of XMFG_FILE_IO and XMFG_CSMI_IO
XMFG_FILE_IO                 /*Must be used exclusive of XMFG_RAW_IO and XMFG_CSMI_IO
XMFG_CSMI_IO                 /*Must be used exclusive of XMFG_FILE_IO and XMFG_RAW_IO
XMFG_WRRD_RATIO             //Only valid for XMFG_WRITE_READ_TEST with XMFG_RANDOM

```

\*These flags determine the I/O interface to be used. XMFG\_RAW I/O uses SCSI Commands via SCSI\_PASSTHROUGH IOCTLs. XMFG\_FILE\_IO uses operating system ReadFile and WriteFile calls for overlapped IO (queued). XMFG\_CSMI\_IO uses SCSI Commands via the CSMI interface on select SAS HBA's for overlapped IO (queued).

**long transfer\_length** - Determines the actual size of the data (in bytes) that will be transferred during the canned test execution. Can be the following define if random transfer lengths are required.

*XMFG\_RANDOM\_TRANSFER\_LENGTH //used in place of a value to generate random transfers*

**int queue\_depth** - Determines the number of outstanding IO's that the software will generate for the target under test. This is only valid for XMFG\_CSMI\_IO or XMFG\_FILE\_IO.

**int cdb\_size** - Determines the size of the SCSI Command (int bytes) to be used during the canned test. This is only valid for XMFG\_CSMI\_IO or XMFG\_RAW\_IO. Can be one of the following defines:

*//cdb\_size defines*

```

XMFG_CDB_SIZE_10           //Only valid for XMFG_RAW_IO and XMFG_CSMI_IO
XMFG_CDB_SIZE_12           //Only valid for XMFG_RAW_IO and XMFG_CSMI_IO
XMFG_CDB_SIZE_16           //Only valid for XMFG_RAW_IO and XMFG_CSMI_IO

```

**unsigned \_\_int64 start\_lba** - Determines the starting LBA (Logical Block Address) for the test

**unsigned \_\_int64 end\_lba** - Determines the ending LBA (Logical Block Address) for the test

\*If both start\_lba and end\_lba have the same value then no LBA restrictions will be used.



## 4.2 XMFG\_Start\_Stop\_Test

### 4.2.1 Description

This API Function executes a multithreaded start/stop test user specified options. This test only works on disk drives.

### 4.2.2 Input

**int device\_index** - The index of the device that will run the test.

**int cycles** - Contains the number of Start/Stop Cycles to be executed.

**int cycle\_delay** - Contains the delay in between Starts and Stops in seconds

### 4.2.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 4.2.4 Example

```
int idx;  
XMFG_Start_Stop_Test(  idx,  //The device index we are running the test on  
                       10,  //10 cycles total  
                       60); //60 seconds between starts and stops
```

## 4.3 XMFG\_Corrupt\_Media

### 4.3.1 Description

This API Function creates ECC based recoverable and unrecoverable errors on SCSI Hard Drives using user-specified options.

### 4.3.2 Input

**int device\_index** - The index of the device that will run the test.

**int error\_type** - Contains the type of error to be created. Can be one of the following defines:

```
XMFG_RECOVERABLE_ERROR  
XMFG_UNRECOVERABLE_ERROR  
XMFG_CUSTOM_BIT_LENGTH_ERROR
```

**unsigned \_\_int64 lba** - Contains the Logical Block Address (LBA) to be corrupted

**int custom\_bit\_count** - Contains the number of bits to corrupt if specified error type is XMFG\_CUSTOM\_BIT\_LENGTH\_ERROR.

### 4.3.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 4.3.4 Example

```
int idx;
```

```
XMFG_Corrupt_Media( idx,  
                    XMFG_RECOVERABLE_ERROR,  
                    lba,  
                    0);    //only used with XMFG_CUSTOM_BIT_LENGTH_ERROR
```

## 4.4 XMFG\_Canned\_Test\_Control

### 4.4.1 Description

This API Function creates ECC based recoverable and unrecoverable errors on SCSI Hard Drives using user-specified options.

### 4.4.2 Input

**int device\_index** - The index of the device that will run the test.

**int control\_code** - Contains the type of control to be exerted on the currently running device(s). Can be one of the following defines:

```
XMFG_STOP_TEST_ON_DEVICE           //stops testing on specified device index
XMFG_STOP_TEST_ON_ALL_DEVICES       //stops testing on all devices
XMFG_PAUSE_TEST_ON_DEVICE           //pauses testing on all devices
XMFG_PAUSE_TEST_ON_ALL_DEVICES      //pauses testing on all devices
XMFG_RESUME_TEST_ON_DEVICE          //resumes testing on all devices
XMFG_RESUME_TEST_ON_ALL_DEVICES     //resumes testing on all devices
```

### 4.4.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 4.4.4 Example

```
int idx;
```

```
XMFG_Canned_Test_Control( idx, XMFG_STOP_TEST_ON_DEVICE);
```



## 4.6 XMFG\_Reset

### 4.6.1 Description

This API Function causes either a physical BUS RESET or DEVICE\_RESET Message.

### 4.6.2 Input

**int device\_index** - The index of the device that will run the test.

**int reset\_type** - Contains the type of RESET to create. Can be one of the following defines:

`XMFG_RESET_BUS` //Causes a physical BUS RESET on the specified target's BUS  
`XMFG_RESET_TARGET` //Causes a DEVICE RESET MESSAGE to be sent to the target

### 4.6.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the `XMFG_error_code` element of the device structure to determine the cause of failure.

### 4.6.4 Example

```
int idx;  
XMFG_Reset(idx, XMFG_RESET_BUS);
```



## 5.2 XMFG\_Issue\_ATA\_Command

### 5.2.1 Description

This API Function issues a user created ATA Register Command to the specified target.

### 5.2.2 Input

**int device\_index** - The index of the device that will receive the command.

**ATA\_REGISTER \*cmd\_register** - The ATA Register structure to be sent. to the device.

**unsigned long data\_length** - Contains the length (bytes) of the data being transferred.

**int data\_direction** - Contains the direction of the data with respect to the initiator (adapter). Must be one of the following defines:

```
XMFG_DATA_IN           //Data will be sent by the device and received by the initiator
XMFG_DATA_OUT         //Data will be sent by the initiator and received by the device
XMFG_DATA_NONE        //No Data will be transferred
```

### 5.2.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the IOCTL\_status, SCSI\_status, SCSI\_sk, SCSI\_asc, and SCSI\_ascq fields in the device structure or parse the sensebuf element of the device structure (sense buffer) yourself.

### 5.2.4 Example

```
int idx = 0;
ATA_REGISTER cmd_regs;

memset(&cmd_regs,0,sizeof(ATA_REGISTER));

//Issue an Identify Command
cmd_regs.Command      = 0xEC;
cmd_regs.SectorCount  = 1;
cmd_regs.size         = 7; // must be either 7 or 12

XMFG_Issue_ATA_Command( idx,
                        cmd_regs,           //ATA_REGISTER *cmd_register,
                        512,                //unsigned long data_length,
                        XMFG_DATA_IN);     //int data_direction
```

## 5.3 XMFG\_Issue\_SAS\_Command (Coming Soon)

### 5.3.1 Description

This API Function issues a user created SAS Command to the specified target.

### 5.3.2 Input

**int device\_index** - The index of the device that will receive the command.

**int command\_type** - The type of SAS Command to issue. Must be one of the following defines:

XMFG\_SSP\_PASSTHROUGH

XMFG\_STP\_PASSTHROUGH

XMFG\_SMP\_PASSTHROUGH

XMFG\_TASK\_MANAGEMENT

**void \*command** - The Command Frame Data to be sent to the device (Formats TBD)

**unsigned long data\_length** - Contains the length (bytes) of the data being transferred.

**int data\_direction** - Contains the direction of the data with respect to the initiator (adapter). Must be one of the following defines:

XMFG\_DATA\_IN               //Data will be sent by the device and received by the initiator

XMFG\_DATA\_OUT             //Data will be sent by the initiator and received by the device

XMFG\_DATA\_NONE            //No Data will be transferred

### 5.3.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 5.3.4 Example

COMING SOON



## Chapter 6 Buffer Functions

### 6.1 XMFG\_Fill\_Buffer

#### 6.1.1 Description

This API Function fills the user-selected buffer with a user-defined or selected data pattern.

#### 6.1.2 Input

**int device\_index** - The index of the device that will receive the Fill Buffer request.

**int buffer\_type** - The buffer type to be filled. Must be one of the following defines:

XMFG\_READ\_BUFFER  
XMFG\_WRITE\_BUFFER  
XMFG\_SENSE\_BUFFER

**unsigned long start\_offset** - The starting offset in the buffer at which the fill should begin.

**unsigned long fill\_size** - Contains the number of bytes to be filled.

**int pattern\_type** - Contains the type of pattern to fill the buffer with.

Must be one of the following defines:

XMFG\_MARCH\_1\_THROUGH\_0\_PATTERN\_HL  
XMFG\_MARCH\_1\_THROUGH\_0\_PATTERN\_LH  
XMFG\_MARCH\_0\_THROUGH\_1\_PATTERN\_HL  
XMFG\_MARCH\_0\_THROUGH\_1\_PATTERN\_LH  
XMFG\_DECREMENTING\_PATTERN  
XMFG\_INCREMENTING\_PATTERN  
XMFG\_RANDOM\_PATTERN  
XMFG\_USER\_DEFINED\_PATTERN

**char \*pattern** - Contains a string with the data pattern to fill if using a user defined pattern.

#### 6.1.3 Output

This function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

#### 6.1.4 Example

```
XMFG_Fill_Buffer( idx,  
                 XMFG_READ_BUFFER,  
                 0,  
                 device[idx]->read_buffsz, //fill the entire read buffer  
                 XMFG_USER_DEFINED_PATTERN,  
                 "0000"); //Zero out the read buffer
```



## 6.3 XMFG\_Embed\_Write\_Buffer

### 6.3.1 Description

This API Function creates custom embedding options for the write buffer during the execution of a canned test that uses the write buffer.

### 6.3.2 Input

**int device\_index** - The index of the device that will receive the Embed Write Buffer request.

**char \*value** - A character array containing the value to embed.

**int type\_of\_embed** - A character array containing the value to embed.

`XMFG_EMBED_STATIC` //embeds the value without refreshing its contents

`XMFG_EMBED_REALTIME` //refreshes the value on each embed operation

`XMFG_EMBED_CLEAR` //clears all previous embed options

**unsigned long start** - The starting position in the buffer (in bytes) at which to begin the embed

**unsigned long end** - The ending position in the buffer (in bytes) at which to finish the embed

**int repeat\_every** - The number of bytes to skip before repeating the embed

### 6.3.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the `XMFG_error_code` element of the device structure to determine the cause of failure.

### 6.3.4 Example

```
int idx = 0;
```

```
//Embed today's date in the data
```

```
XMFG_Embed_Write_Buffer(    idx,  
                             "DATE",    //embeds the value of the define DATE into data  
                             XMFG_EMBED_STATIC,  
                             0,  
                             262144,  
                             device[idx]->block);    //Repeat the embed every block
```



## Chapter 7 Error Handling

Just as in Extreme Manufacturing, this API has the ability to call Error Handlers when certain types of errors occur during test or command execution. This chapter deals with prototyping, registering, including and implementing error handlers.

### 7.1 Function Prototypes for Error Handlers

Before error handlers can be registered or used, they must be prototyped. The following shows sample function prototypes for each class of error handler. The function names can be anything you wish but the arguments for each class of error handler must be exactly as shown.

#### 7.1.1 Data Compare Handler Class

This class is used to handle Data Compare Errors. By default the software will flag the data compare and change the device state to STATE\_MISCOMPARE. A date, timestamp, cdb and the Logical Block Address of the error will be logged. Finally, 4 bytes prior and post the first compare error in both the read buffer and write buffer will be logged with () surrounding the first compare error.

*//Function Prototype (can be any desired function name but must have these exact arguments)*

```
int XMFG_Compare_Handler(    int idx,
                             unsigned __int64 lba,
                             unsigned long xlen,
                             unsigned char *rdbuf,
                             unsigned char *wrtbuf,
                             unsigned long byte_offset);
```

The following data will be returned to your Data Compare Handler Function function...

idx	The index of the device that took the error
lba	The Logical Block Address at which the error occurred.
xlen	The transfer length that had been used at the time of the error.
rdbuf	A pointer to the read buffer that contains the errant data
wrtbuf	A pointer to the write buffer that contains the data being compared against
byte_offset	The location within the buffer that contains the first byte in error.

### 7.1.2 SCSI Error Handler Class

This class is used to handle SCSI Errors. By default the software will flag the SCSI Error and change the device state to STATE\_RUNNING\_WITH\_ERRORS. A date, timestamp, cdb and both short and long versions of the sense data will be logged.

*//Function Prototype (can be any desired function name but must have these exact arguments)*

```
int XMFG_SCSI_Error_Handler( int idx,  
                             unsigned __int64 lba,  
                             unsigned long xlen,  
                             unsigned char *rdbuf,  
                             unsigned char *wrtbuf,  
                             unsigned char *sense_buffer,  
                             unsigned char *cdb,  
                             int cdb_size);
```

The following data will be returned to your Data Compare Handler Function function...

idx	The index of the device that took the error
lba	The Logical Block Address at which the error occurred.
xlen	The transfer length that had been used at the time of the error.
rdbuf	A pointer to the read buffer used at the time of the error
wrtbuf	A pointer to the write buffer used at the time of the error
sense_buffer	A pointer to the sense buffer that contains error information
cdb	A pointer to the CDB (Command Descriptor Block) issued at the time of the error
cdb_size	The size of the CDB issued at the time of the error

### 7.1.3 ATA Error Handler Class

This class is used to handle ATA Errors. By default the software will flag the ATA error and change the device state to STATE\_RUNNING\_WITH\_ERRORS. A date, timestamp, command registers, status registers will be logged.

*//Function Prototype (can be any desired function name but must have these exact arguments)*

```
int XMFG_SCSI_Error_Handler( int idx,  
                             unsigned __int64 lba,  
                             unsigned long xlen,  
                             unsigned char *rdbuf,  
                             unsigned char *wrtbuf,  
                             ATA_REGISTER *status_regs,  
                             ATA_REGISTER *cmd_regs);
```

The following data will be returned to your Data Compare Handler Function function...

idx	The index of the device that took the error
lba	The Logical Block Address at which the error occurred.
xlen	The transfer length that had been used at the time of the error.
rdbuf	A pointer to the read buffer used at the time of the error
wrtbuf	A pointer to the write buffer used at the time of the error
status_regs	A pointer to the status registers returned by the device during the error

### 7.1.4 IOCTL Error Handler Class

This class is used to handle IOCTL Errors. IOCTL errors occur when the HBA (Host Bus Adapter) driver cannot connect to the device, does not recognize the request, receives an illegal request or causes an internal error. By default the software will flag the IOCTL error and change the device state to STATE\_RUNNING\_WITH\_ERRORS. A date, timestamp, command information and IOCTL error information (including description) will be logged.

*//Function Prototype (can be any desired function name but must have these exact arguments)*

```
int XMFG_IOCTL_Error_Handler( int idx,  
                             unsigned __int64 lba,  
                             unsigned long xlen,  
                             unsigned char *rdbuf,  
                             unsigned char *wrtbuf,  
                             int error_code);
```

The following data will be returned to your Data Compare Handler Function function...

idx	The index of the device that took the error
lba	The Logical Block Address at which the error occurred.
xlen	The transfer length that had been used at the time of the error.
rdbuf	A pointer to the read buffer used at the time of the error
wrtbuf	A pointer to the write buffer used at the time of the error
error_code	An integer containing the IOCTL error that occurred.

## 7.2 Registering Error Handlers

### 7.2.1 Description

Before using Error Handler Callback Functions, you must first register them with the DLL. To do this, we have provided four convenience functions which only require you to pass in the name of the function being used. If a NULL is passed in, then the Error Handler Callback will be unregistered.

### 7.2.2 Functions

The following function calls are used to register their respective error handlers and to attach a local callback function to the handler so that it may be called during an error condition.

```
void XMFG_Register_Compare_Handler(<your function name>);  
void XMFG_Register_SCSI_Handler(<your function name>);  
void XMFG_Register_ATA_Handler(<your function name>);  
void XMFG_Register_IOCTL_Handler(<your function name>);
```

### 7.2.3 Output

There are no return values for these functions.

### 7.2.4 Examples

*//Function Prototype (can be any desired function name but must have these exact arguments)*

```
int XMFG_Compare_Handler(    int idx,  
                            unsigned __int64 lba,  
                            unsigned long xlen,  
                            unsigned char *rdbuf,  
                            unsigned char *wrtbuf,  
                            unsigned long byte_offset);
```

*//Register the Error Handler*

```
XMFG_Register_Compare_Handler(XMFG_Compare_Handler);
```

## 7.3 Implementing Error Handlers

### 7.3.1 SCSI Error Handler Example

This example is a simple error handler that keeps track of the number of Unrecoverable (03) and Recoverable (01) errors and stops testing if they exceed a certain number.

```
int XMFG_SCSI_Error_Handler ( int idx,
                             unsigned __int64 lba,
                             unsigned long xlen,
                             unsigned char *rdbuf,
                             unsigned char *wrtbuf,
                             unsigned char *sense_buffer,
                             unsigned char *cdb,
                             int cdb_size)
{
    unsigned char message[1024];
    char tmptxt[32];
    int x;
    static int count_03 = 0; //unrecoverable errors
    static int count_01 = 0; //recoverable errors

    if(sense_buffer[2] == 0x03)
    {
        count_03++;

        //Stop testing if 03 Error Count Exceeds 10
        if(count_03 > 9)
            XMFG_Canned_Test_Control(idx, XMFG_STOP_TEST_ON_DEVICE);
    }
    else if(sense_buffer[2] == 0x01)
    {
        count_01++;

        //Stop testing if 01 Error Count Exceeds 50
        if(count_01 > 49)
            XMFG_Canned_Test_Control(idx, XMFG_STOP_TEST_ON_DEVICE);
    }

    return 1;    // return 1 to tell the DLL to continue processing the error normally,
                // 0 to tell the DLL to stop processing the error
}
```



## Chapter 8 Convenience Functions

Just as in Extreme Manufacturing, this API has the several convenience functions to simplify what would otherwise be an extensive programming task. This chapter deals with using email, ftp and port convenience functions.

### 8.1 XMFG\_Send\_Email

#### 8.1.1 Description

This API Function provides a mechanism for sending email via Extreme Manufacturing. This can be extremely useful if placed for example in an error handler to notify an engineer of any type of failure or to email yourself logs or results at the end of a test.

#### 8.1.2 Input

**int device\_index** - The index of the device whose value will be set.

**char \*from** - A character array that will be shown in the from field in the recipient's inbox.

**char \*to** - A character array containing the intended recipient's email address.

**char \*cc** - A character array containing an additional recipient's email address.

**char \*subject** - A character array containing the subject of the email.

**char \*message** - A character array containing the message body of the email.

**char \*attach** - A character array containing the full path and filename of a file to be attached.

#### 8.1.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

#### 8.1.4 Example

```
char message_txt[1024], to_txt[40], from_txt[40], subject[40]
XMFG_Send_Email( idx,
                 from_txt,
                 to_txt,
                 NULL,
                 subject,
                 message_txt,
                 NULL);
```

## 8.2 XMFG\_FTP\_Login

### 8.2.1 Description

This API Function allows the user to login to an FTP site.

### 8.2.2 Input

**char \*host** - A character array contains the host site information

**int port** - A integer containing the FTP port

**char \*user\_id** - A character array containing the user id used to login.

**char \*password** - A character array containing the password used to login

### 8.2.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.2.4 Example

```
char host[40], user_id[40], password[40];
int port;
```

```
port = 21;
sprintf(host, "ftp.mydomain.com");
sprintf(user_id, "my_login_name");
sprintf(password, "my_login_password");
```

```
XMFG_FTP_Login( host,
                port,
                user_id,
                password);
```

## 8.3 XMFG\_FTP\_Logout

### 8.3.1 Description

This API Function allows the user to logout of an FTP site.

### 8.3.2 Input

There are no input parameters to this function.

### 8.3.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.3.4 Example

```
XMFG_FTP_Logout();
```

## 8.4 XMFG\_FTP\_Put

### 8.4.1 Description

This API Function allows the user to put local files on the FTP server.

### 8.4.2 Input

**char \*filename** - A character array contains full path and filename to send to the FTP server.

**int type**- An integer identifying the type of file. Must be one of the following defines:

XMFG\_FTP\_ASCII

XMFG\_FTP\_BINARY

### 8.4.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.4.4 Example

```
char filename[255];  
sprintf(filename,"c:\\temp\\mylog.log");  
  
XMFG_FTP_Put(filename, XMFG_FTP_ASCII);
```

## 8.5 XMFG\_FTP\_Change\_Directory

### 8.5.1 Description

This API Function changes the remote directory on the FTP server.

### 8.5.2 Input

**char \*remote\_directory**- A character array contains directory to change to.

### 8.5.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.5.4 Example

```
char dir[255];  
sprintf(dir,"myfolder");  
  
XMFG_FTP_Change_Directory(dir);
```

## 8.6 XMFG\_FTP\_Get

### 8.6.1 Description

This API Function allows the user to get files from the FTP server.

### 8.6.2 Input

**char \*filename** - A character array contains full path and filename to get from the FTP server.

**int type**- An integer identifying the type of file. Must be one of the following defines:

```
XMFG_FTP_ASCII  
XMFG_FTP_BINARY
```

### 8.6.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.6.4 Example

```
char filename[255];  
sprintf(filename,"myfile.txt");  
  
XMFG_FTP_Get(filename, XMFG_FTP_ASCII);
```

## 8.7 XMFG\_IO\_PORT\_Open

### 8.7.1 Description

This API Function opens a connection to the specified port.

### 8.7.2 Input

**int device\_index** - The index of the device that will open the port.

**char \*port\_name**- A character array contains the port name ex: "LPT1" , "COM1" etc.

### 8.7.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.7.4 Example

```
char port[16];
int idx = 0;

sprintf(port,"LPT1");

XMFG_IO_PORT_Open(idx, port);
```

## 8.8 XMFG\_IO\_PORT\_Close

### 8.8.1 Description

This API Function closes a connection to a previously opened port.

### 8.8.2 Input

**int device\_index** - The index of the device that opened the port.

### 8.8.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.8.4 Example

```
int idx = 0;

XMFG_IO_PORT_Close(idx);
```

## 8.9 XMFG\_IO\_PORT\_Send

### 8.9.1 Description

This API Function sends a user data buffer across the specified port.

### 8.9.2 Input

**int device\_index** - The index of the device that has opened the port.

**char \*port\_name** - A character array contains the port name ex: "LPT1", "COM1" etc.

**unsigned char \*buffer** - The buffer to send across the port.

**int start\_offset** - The start offset in the buffer at which transmission will begin

**int length** - The length of the data to be transmitted

### 8.9.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.9.4 Example

```
char port[16], buffer[256];  
int idx = 0;  
int start = 0;  
int length = 256;
```

```
sprintf(port,"COM1");
```

```
XMFG_IO_PORT_Send(  idx,  
                    port,  
                    buffer,  
                    start,  
                    length);
```

## 8.10 XMFG\_IO\_PORT\_Receive

### 8.10.1 Description

This API Function receives data into a user's buffer across the specified port.

### 8.10.2 Input

**int device\_index** - The index of the device that has opened the port.

**char \*port\_name** - A character array contains the port name ex: "LPT1", "COM1" etc.

**unsigned char \*buffer** - The buffer to receive the data into.

**int start\_offset** - The start offset in the buffer at which receipt of data will begin

**int length** - The length of the data to be received

### 8.10.3 Output

The is function returns a 1 if successful and a 0 if an error occurs. If an error occurs you can look at the XMFG\_error\_code element of the device structure to determine the cause of failure.

### 8.10.4 Example

```
char port[16], buffer[256];
```

```
int idx = 0;
```

```
int start = 0;
```

```
int length = 256;
```

```
sprintf(port,"COM1");
```

```
XMFG_IO_PORT_Receive(idx,  
                    port,  
                    buffer,  
                    start,  
                    length);
```