the power of less

Integrated ANT-FS Client Interface Control Document

D00001417 Rev 1.0

P +1 403.932.9292 F +1 403.932.6521
228 River Avenue, Cochrane, Alberta, Canada T4C 2C1 thisisant.com

Copyright Information and Usage Notice

Table of Contents

1 Introduction

ANT-FS is an extension of the ANT protocol that facilitates the secure and automated transfer of files over a wireless ANT connection. Communication and handshaking occurs between an ANT-FS host and ANT-FS client device. The host device is often a data aggregator, or hub for transport to a database or web service. It is often implemented on a device such as a PC, tablet or Smartphone. Such devices often have lesser power constraints, run with an operating system and have a rich development environment. Development on these devices is facilitated by ANT-FS Host libraries that may be ported to any target operating system.

Conversely the client is often the data collector. The Client typically collects data from sensors and stores this data until it is in the vicinity of an ANT-FS host, which may then authenticate with the client and download the data. Client devices are often implemented on very resource constrained systems which operate on small batteries and have no or limited operating systems. For such systems the ANT-FS client functionality must be implemented by following embedded reference code examples. Furthermore, a file system must be implemented to allow data to be stored and retrieved as required. This is a cumbersome process that requires significant development time and an MCU capable of supporting the space and processing requirements of an ANT-FS client. The Integrated ANT-FS Client feature, available on some ANT chips, eliminates this problem by implementing the entire ANT-FS Client directly on the chip. Furthermore, a file system is implemented and completely managed by the ANT chip, further reducing the time and complexity of designing an ANT-FS client.

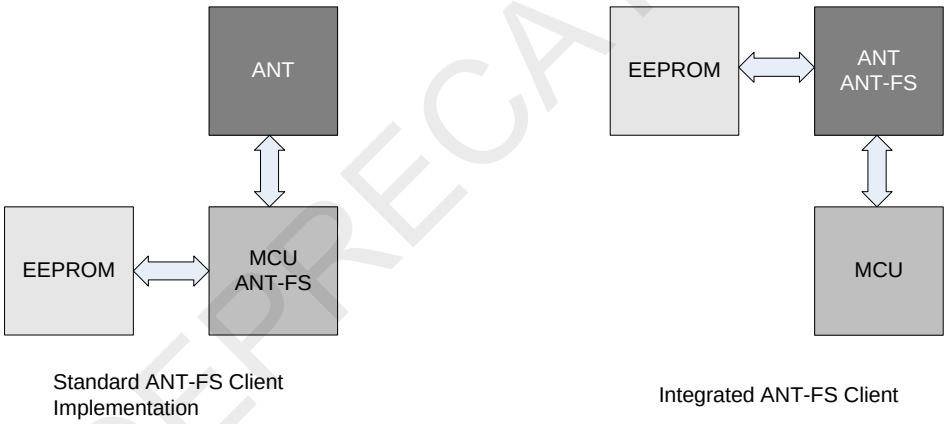

Figure 1. Standard vs. Integrated ANT-FS Implementation

The EEPROM memory device is connected directly to the ANT chip facilitating rapid development, reduced BOM costs and quick time to market for ANT-FS Client enabled devices.

2 Reference Documents

It is strongly recommended that the following documents be read and understood before using this document.

Table 1. Reference Documents

Document#	Document Title
D00000652	ANT Message Protocol and Usage
D00001025	ANT File Share (ANT-FS) Technology
D00001152	ANT-FS Reference Design User Manual
D00001275	ANT+ FIT File Protocol

3 File System

The Integrated ANT-FS Client includes support for a full robust file system, further simplifying the development effort to bring products to market. The file system requires that an external EEPROM be connected directly to the ANT chip for data storage (Figure 1). For interconnect information and supported EEPROM devices please consult the datasheet of the specific ANT part.

The integrated file system (FS) includes support for a directory structure, which is identical to that defined by the ANT-FS Technical Specification. Files may be downloaded, uploaded, and erased as required by the application MCU, or remotely by the ANT-FS Host. Concurrency is fully managed by the file system to ensure data consistency throughout transactions. For example, if the application MCU wishes to write data to a file that is currently being accessed by an ANT-FS host, it will be prevented from doing so.

The entire file system is fully exposed to the application MCU over an extended serial message protocol. This simple interface allows files to be created, downloaded, and erased. Flags may be set to further define behavior. On the wireless side, the file system is exposed over the ANT-FS protocol. Up to 8 files may be opened concurrently at any given time. As file handles are shared by the application process and by the ANT-FS Host, the application processor should be careful to have at least one file handle free if ANT-FS is enabled.

Files may be stored as raw data bytes, or they may be encrypted with AES-128 encryption. This feature is easily enabled by specifying the flag bit of a file as ‘Crypto’ upon creation. Once a file is flagged as ‘Crypto’, it will be encrypted by a specified key. The file is stored in the EEPROM in its encrypted format. Any transmission of the file over ANT is also done using the encrypted form of the file. If a file is opened for reading with the flag set to ‘Crypto’, it will be decrypted using the specified key and sent in its original format to the application MCU. Up to ten 32 byte keys may be stored and used for encryption and decryption of data.

Another feature of the integrated file system is the ability to store FIT file formats. By specifying a file type as “FIT” upon creation of a file, the file system will automatically update the FIT header for the file and manage the file length and CRC. As a result, the host MCU need only send the FIT definitions and data records as needed.

3.1 File System Configuration

Only EEPROMs with an SPI interface are supported. Please see ANT part datasheet for a complete description of what EEPROM parts are suitable for use. Before any FS or ANT-FS commands may be sent, the memory device must first be initialized by sending the number of bytes per EEPROM page (a page is the maximum number of bytes that may be written to the EEPROM in a single transaction) and the number of bytes used to address the EEPROM (2 or 3 bytes depending on the size of the EEPROM).

A memory device must be formatted before its first use. This is accomplished by using the Format Memory command and requires the user input the number of sectors and sectors used per page. Files are always stored as units of sector. Sector size can be defined by the application and will depend on whether the system expects to store a small number of large files, or a large number of small files. Pages are fixed in size for a specific EEPROM, please consult the specific EEPROM datasheet for page size. The diagram below describes the relationship between sectors and pages.

File Mapping	User Defined	EEPROM Defined
File 0	SECTOR 0	PAGE 0
		PAGE 1
	SECTOR 1	PAGE 2
		PAGE 3
	SECTOR 2	PAGE 4
		PAGE 5
File 1	SECTOR 3	PAGE 6
		PAGE 7
	SECTOR 4	PAGE 8
		PAGE 9
File 2	SECTOR 5	PAGE 10
		PAGE 11
	SECTOR 6	PAGE 12
		PAGE 13
	SECTOR 7	PAGE 14
		PAGE 15

Figure 2. Sample EEPROM Configuration

Once the *memory device* has been initialized and formatted, the *file system* must then be initialized by sending the Memory Init command. Please see Figure 3 for a flow chart of the steps required to initialize the memory device and the file system.

Following the Memory Init command the application MCU may first read out the directory structure of the EEPROM to check and which files currently exist. Files may subsequently be created, uploaded, downloaded and erased as desired. To upload or download a file, the file must exist in the directory structure; if not the file must be created using the Create File command. Optionally files may be created by an ANT-FS Host over the ANT-FS protocol, however this option must first be enabled by the host MCU with the appropriate command. When creating a new file, care must be taken to ensure the correct file type and flags are specified. Please consult the ANT-FS Technical Specification for details regarding the use of the ANT-FS directory and associated flags and file types. After a file has been created, either by the application MCU or the ANT-FS host, it may be opened for reading and writing. There are several flags associated with this operation, allowing the application MCU to append to an existing file or write a crypto file. There are several functions for reading and writing files, giving the application MCU the option of writing to absolute or relative addresses within the file.

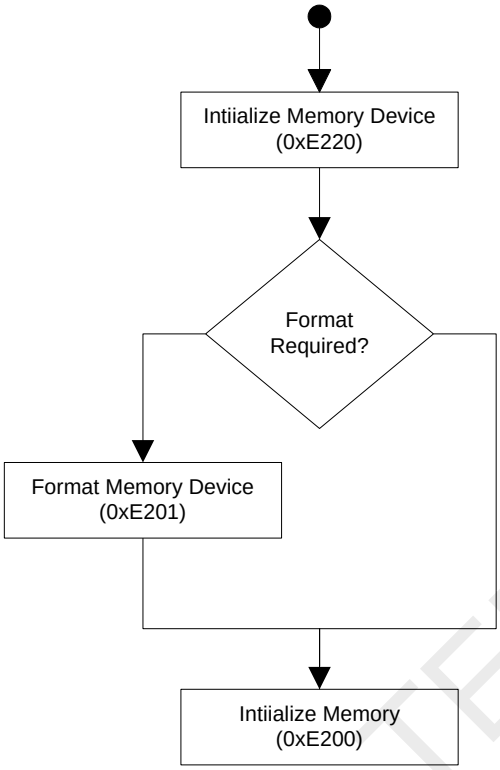

Figure 3. Initializing FS

After open files have received updates, the application MCU should save the changes to the directory structure by calling the Directory Save command. This ensures that the changes made to individual files are saved to the directory structure, ensuring consistency between what is reported in the directory and what actually exists on the EEPROM. This command should always be called before powering off the ANT chip. The directory may also be rebuilt to condense its size and remove any invalidated entries by calling the Directory Rebuild command.

3.2 Crypto Files

The Integrated FS/ANT-FS feature includes support for AES-128 encryption of files. Encrypted files are stored in an encrypted format on the EEPROM enabling secure over-the-air transfer of sensitive data. To enable encryption of a file, an AES-128 encryption key must first be specified. Keys are 32 bytes in length and may be stored in the non-volatile memory controlled by the ANT chip. Up to 10 keys may be stored. The index of the key to use must be specified before encryption or decryption takes place. Alternatively a session key may be specified. Session keys are not stored in non-volatile memory are are only valid until the part is reset.

Once a key is specified, encryption can be enabled by setting bit 2 of the General Flag. The General Flag is specified in the directory entry of the file and may be set upon creation of the file. For complete details please consult the ANT-FS Technical Specification. Figure 4 depicts the General Flag and the placement of the Crypto bit.

Once a file is flagged with the Crypto bit, the file must be opened with a Crypto enabled file handle. This is accomplished by setting the Crypto bit in the ‘Open Flags’ field when opening a file. Once opened, any data written to the file handle will be encrypted using the specified key. Any data read from the file handle will be decrypted using the specified key. Only one Crypto file handle may be used at any given time.

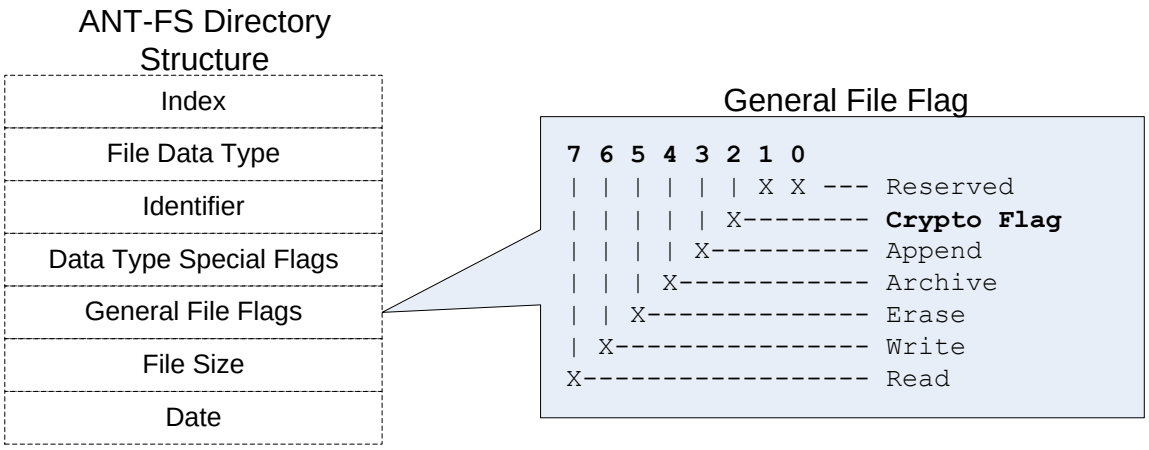

Figure 4. Setting Crypto bit in General File Flags

An encrypted file may be opened without the Crypto bit, in which case any data read will be in the encrypted format. This allows the application MCU to decrypt the data using the correct key. This is useful if, for example, designing a PC application that needs to work with USB devices that may not have the crypto feature.

The creation and exchange of the 32 byte encryption key is not defined by the Integrated FS/ANT-FS feature. This process must be designed and specified for a given application to ensure required levels of protection.

3.3 FIT Files

The integrated FS system also includes support for FIT file types. A file may be specified as a FIT file by specifying the File Data Type field to be 0x80. This must be specified when creating the file.

Once a file type has been specified as FIT, the FIT header and CRC will be managed by the file system. All the application MCU needs to do is to send the initial file header and any message definitions and data messages as required for a particular FIT file type.

FIT files may also be encrypted. To do this the FIT file must be opened with the crypto file handle and follow the same procedures as any crypto file.

4 ANT-FS Client

The Integrated ANT-FS engine includes full support for an ANT-FS Client as described by the ANT-FS Technical specification. Coupled with the integrated FS feature, development of fully functional file based ANT devices is greatly simplified, reducing the computational burden from the application MCU and reducing NRE time and costs.

The Integrated ANT-FS Client is configured and controlled using an extended serial interface. An ANT-FS beacon may be assigned to any or all ANT channels available on the device. To configure an ANT-FS beacon, please follow the procedure below:

Set Network Key. Use the regular serial command (Msg ID 0x46). For ANT+ implementations this should be the ANT+ Managed Network key.

Assign channel as master. Each channel that is to run the ANT-FS Client should be assigned as a master using the regular assign channel command (Msg ID 0x42)

Set the channel id of each ANT-FS channel. The channel ID of each ANT-FS channel should be set using the channel id command (Msg ID 0x51). For ANT+ specific implementations the portions of the channel ID may be specified by the profile.

Configure the beacon using the extended serial message 0xE233.

Set Authentication strings if required using extended serial message 0xE234. By default no friendly name and no passkey is set.

Set beacon timeout (0xE238) and pairing timeout (0xE239) if required. The defaults are 10s and 30s respectively.

Configure the beacon RF frequency for each ANT-FS channel using the extended serial message 0xE237. A beacon can be disabled by setting the frequency to 0xFF.

Start ANT-FS using extended serial command 0xE231. The ANT-FS beacon will begin to broadcast on all channels assigned as ANT-FS. After this a LINK_EVENT will occur on channel 0xFF, meaning that the ANT-FS engine has gone to LINK without an active ANT-FS channel.

If an ANT-FS Host connects to the client an AUTH_EVENT will be sent along with the corresponding ANT channel number and all other ANT-FS channels shall broadcast the beacon indicating “BUSY” state.

Once the ANT-FS beacon has been configured, and the client has commenced beaconing in the link state, an ANT-FS host may attempt a connection. If the host attempts to pair to the client, a PAIRING_REQUEST event will occur. The application MCU may accept or reject this request using the extended serial command 0xE236. If authentication passes, the client shall move to the TRANSPORT state, if authentication fails the client will move to the LINK state. Both of these conditions will be indicated with an event.

Once in the TRANSPORT state, the ANT-FS engine will generate start and stop events for file upload and download. This allows the application MCU to be aware of when files have been read, updated and also when these operations are in progress allowing it to manage its resources appropriately.

The application MCU is also informed when a Command Pipe command has been received and if it has been processed by the ANT-FS engine automatically.

4.1 Authentication Methods

The integrated ANT-FS Client supports three methods of authentication – passthough, passkey and pairing. The actual method used is specified by setting the appropriate bit in the ANT-FS beacon status byte. The figure below shows the logic used by the ANT-FS engine to accept or reject a particular method based on what bit is set in the beacon.

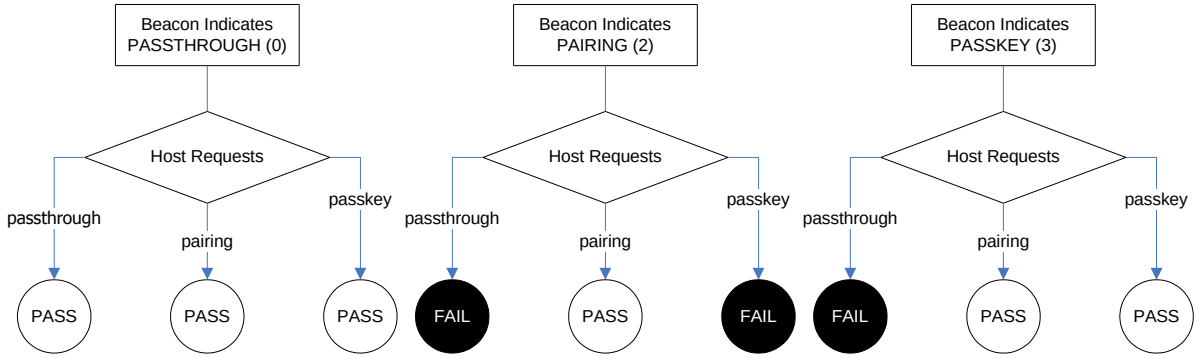

Figure 5. Authentication Decision Tree Based on Beacon Indicator

Please note that for the figure above a ‘pass’ for a pairing requests assumes the application MCU accepted the pairing request.

4.2 Command Pipe

The integrated ANT-FS Client also has support for command pipes. The client may send or receive commands over the ANT-FS command pipe using the extended serial requests 0xE23B and 0xE23C respectively.

Upon receiving a command on the ANT-FS command pipe the application MCU will be notified with one of the following events:

MESG_FS_ANTFS_EVENT_CMD_RECIEVED (0x0A)

MESG_FS_ANTFS_EVENT_CMD_PROCESSED (0x0B)

For details of each event please consult section 6.8.

The following commands are processed by the integrated ANT-FS engine:

Create File (must be enabled by application MCU)

Set System Time

Get System Time

Please consult the ANT-FS Technical Specification for more details on the ANT-FS Command pipe.

5 Extended Serial Messaging

To enable the functionality of the integrated FS/ANT-FS feature the serial protocol between the application MCU and the ANT chip had to be extended to accommodate the required number of serial messages and event. In addition to using 2 byte message ID’s, the extended serial protocol is more dynamic than the standard ANT serial protocol. For all commands outside of the FS/ANT-FS function the standard serial protocol still applies, as detailed by the ANT Message Protocol and Usage document.

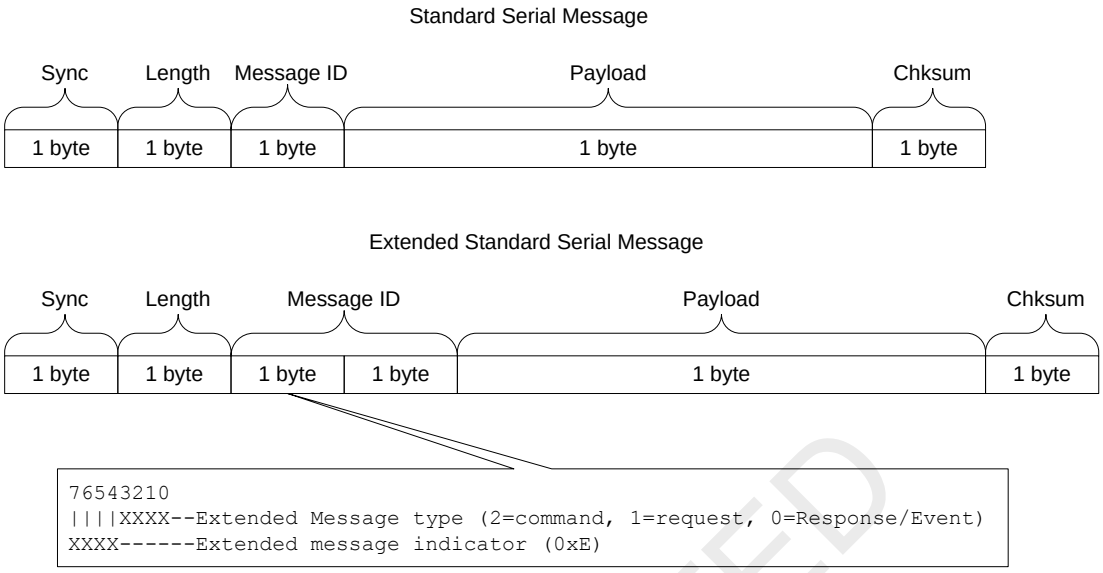

Figure 6. Serial Message General Packet Structure

The extended serial protocol defines three types of messages – commands, requests and responses/events. Compared to a standard serial message, an extended serial message is indicated by setting the top nibble message id to 0xE. The type of extended message is indicated in the following nibble (Command = 0xE2, Request = 0xE1 and Response/Event = 0xE0). The following byte is the message id, followed by the payload of the message.

The checksum for a standard and extended serial message is calculated as the XOR of all bytes including the sync byte. The length of a standard message includes the number of bytes in the payload of the message, whereas for extended serial messages the length includes the payload + 1 byte. As with the standard serial protocol, the use of optional padding bytes is also recommended for systems that are slow to react to hardware flow control (when using asynchronous serial interface).

In general, a command sent by the MCU to ANT elicits a response. A request elicits a command and an event is unsolicited from ANT to the MCU. This behavior is illustrated in the diagram below.

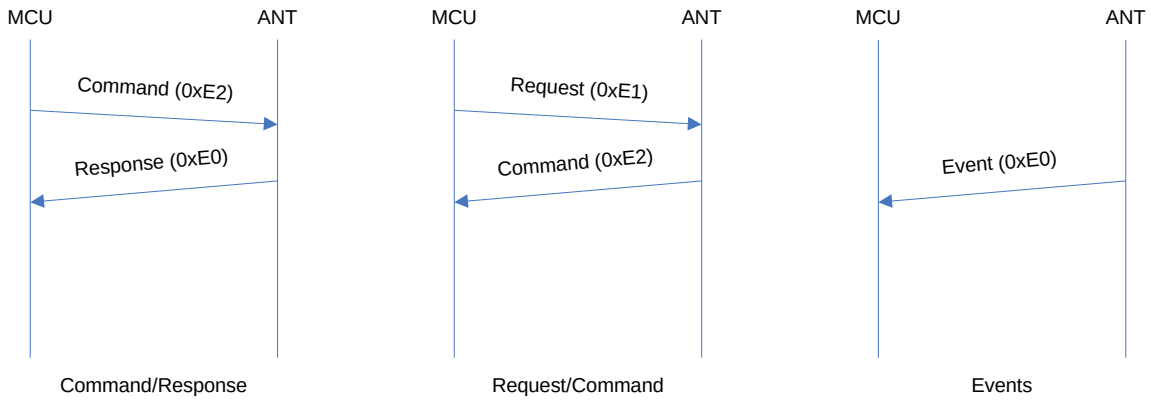

Figure 7. Extended Serial Message Types

5.1 Commands (0xE2)

Extended serial commands are used to execute specific commands on the ANT chip, or to get information about the state of the FS/ANT-FS engine. When sent by the application MCU they elicit a response from the ANT chip. If a message request is made by the application MCU, then ANT will send a command message in response.

The structure of a command is identical to that described by Figure 6, with the message indicator set to 2. The specific command to execute is indicated by the second byte of the message id. The payload is specific to a particular command and may be completely empty. The figure below describes an extended serial command.

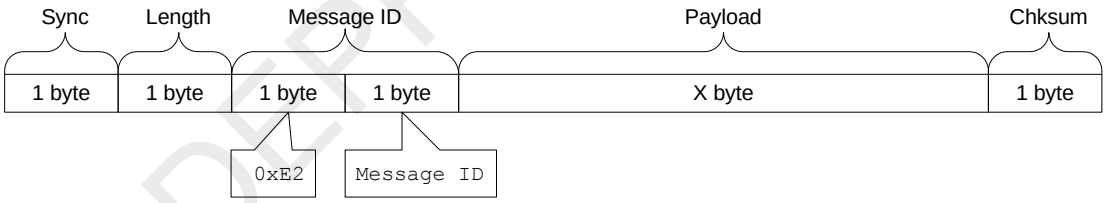

Figure 8. Extended Serial Command

When a command is requested by the application MCU the first byte of the payload of the command will generally be the FS Response Code which indicates the success or failure of the request. However, there are some cases where this is not true. For example, the Time command request does not have an associated response code..

5.2 Requests (0xE1)

Extended serial requests are used by the application MCU to request information from the ANT chip. The information requested comes to the MCU in the form of a command.

The structure of a request command is identical to that described by Figure 6. The first two bytes of the payload indicate what command is being requested. The remaining payload may be used to specify other criteria for the request, as require by the specific command.

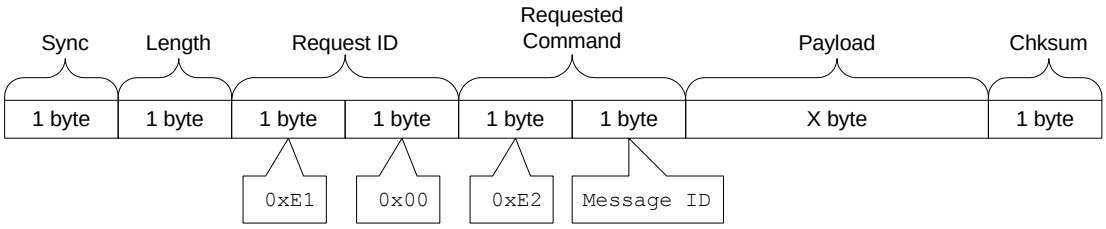

Figure 9. Extended Serial Request

5.3 Responses/Events (0xE0)

Responses are used to indicate the success or failure of a command. Events are a special form of response messages. They are unsolicited messages that come from the ANT chip and indicate some type of information which the application MCU must handle.

The structure of a Response/Event message is identical to that described by Figure 6. The payload of responses includes the id of the message being responded to as well as the response code and associated payload.

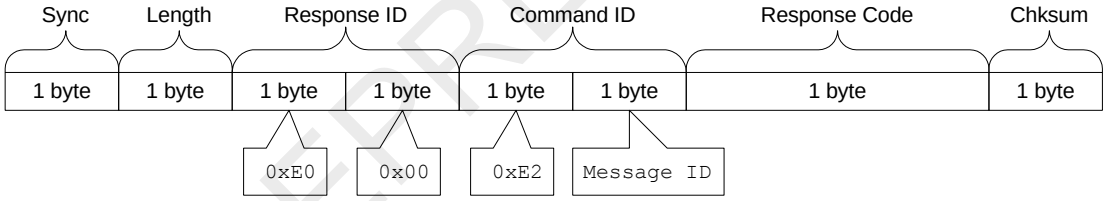

Figure 10. Extended Serial Response

The payload of events includes the event command id (0xE230) followed by the event code and the optional payload.

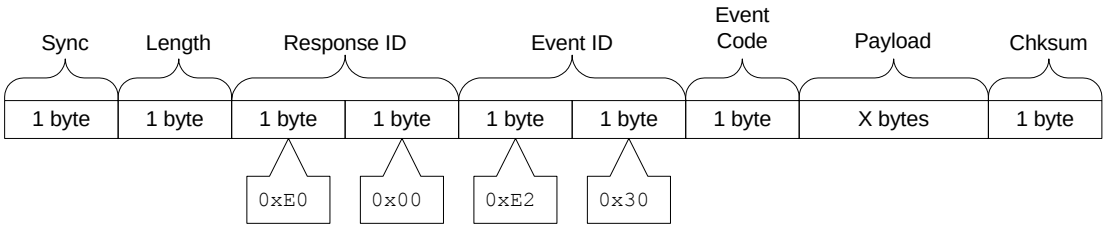

Figure 11. Extended Serial Event

6 ANT Message Summary

The following tables summarize all of the extended serial messages. Each is described in detail in the subsequent sections. The DLL library interface function is also described and used to explain the general use of a specific function.

Table 2. Summary of Commands and Responses

Class	ANT PC Interface Function	Message Structure				
Memory Device Commands	ANTFS_InitEEPROMDevice() 6.1.1 (p20)	4	0xE220	Page Write Size (2 bytes)	Address Bytes Cfg (1 byte)	
	Response	4	0xE000	0xE220	FS Response Code (1 byte)	
FS Commands	ANTFS_InitFSMemory() 6.2.1 (p21)	1	0xE200			
	Response	4	0xE000	0xE200	FS Response Code (1 byte)	
	ANTFS_FormatFSMemory() 6.2.2 p(22)	5	0xE201	FS Sectors (2-bytes)	Pages Per Sector (2-bytes)	
	Response	4	0xE000	0xE201	FS Response Code (1 byte)	
	ANTFS_SaveDirectory() 6.2.2 p(22)	5	0xE207			
	Response	4	0xE000	0xE207	FS Response Code (1 byte)	
	ANTFS_RebuildDirectory() 6.2.4 p(24)	1	0xE209			
	Response	4	0xE000	0xE209	FS Response Code (1 byte)	
	ANTFS_FileDelete() 6.2.5 p(25)	2	0xE20C	File Handle (1-byte)		
	Response	4	0xE000	0xE20C	FS Response Code (1 byte)	
	ANTFS_FileClose() 6.2.6 p(26)	2	0xE20D	File Handle (1-byte)		
	Response	4	0xE000	0xE20D	FS Response Code (1 byte)	
	ANTFS_SetFileSpecificFlags() 6.2.7 p(28)	3	0xE212	File Handle (1-byte)	Flags (1-byte)	
	Response	4	0xE000	0xE212	FS Response Code (1 byte)	
	ANTFS_DirectoryReadLock() 6.2.8 p(29)	2	0xE216	Lock (1 –byte)		
	Response	4	0xE000	0xE216	FS Response Code (1 byte)	
	ANTFS_SetSystemTime() 0 p(30)	5	0xE23D	Current Time (4-bytes)		
	Response	4	0xE000	0xE23D	FS Response Code (1 byte)	
FS Requests	ANTFS_GetUsedSpace()	3	0xE100	0xE202		

6.3.1 p(31)								
Response	6	0xE202	FS Response Code (1 byte)	Used Space (4-bytes)				
ANTFS_GetFreeSpace() 6.3.2 p(32)	3	0xE100	0xE203					
Response	6	0xE203	FS Response Code (1 byte)	Free Space (4-bytes)				
ANTFS_FindFileIndex() 6.3.3 p(33)	7	0xE100	0xE204	File Data Type (1-byte)	File Sub Type (1-byte)	File Number (2-bytes)		
Response	4	0xE204	FS Response Code (1 byte)	File Index (2-bytes)				
ANTFS_DirectoryReadAbsolute() 6.3.4 p(35)	8	0xE100	0xE205	Offset (4-bytes)	Size (1-byte)			
Response	3+X	0xE205	FS Response Code (1 byte)	Size Read (1-byte)	Read Payload (X-bytes)			
ANTFS_DirectoryReadEntry() 6.3.5 p(36)	5	0xE100	0xE206	File Index (2-bytes)				
Response	18	0xE206	FS Response Code (1 byte)	ANTFS Directory Entry (16 bytes)				
ANTFS_DirectoryGetSize() 6.3.6 p(37)	3	0xE100	0xE208					
Response	6	0xE208	FS Response Code (1 byte)	Directory Size (4-bytes)				
ANTFS_FileCreate() 0 p(38)	8	0xE100	0xE20A	File Index (2-bytes)	Directory Entry (6-bytes)			
Response	4	0xE20A	FS Response Code (1 byte)	File Index (2 bytes)				
ANTFS_FileOpen() 6.3.8 p(40)	6	0xE100	0xE20B	File Index (2 bytes)	Open Flags (1-byte)			
Response	3	0xE20B	FS Response Code (1 byte)	File Handle (1-byte)				
ANTFS_FileReadAbsolute() 6.3.9 p(43)	9	0xE100	oxE20E	File Handle (1-byte)	Offset (4-bytes)	Read Size (1-byte)		
Response	3+X	0xE20E	FS Response Code (1 byte)	Size Read (1-byte)	Read Payload (X-bytes)			
ANTFS_FileReadRelative() 6.3.10 p(45)	5	0xE100	0xE20F	File Handle (1-byte)	Read Size (1-byte)			
Response	3+X	0xE20F	FS Response Code (1 byte)	Size Read (1-byte)	Read Payload (X-bytes)			
ANTFS_FileWriteAbsolute() 6.3.11 p(46)	9+X	0xE100	0xE210	File Handle (1-byte)	Offset (4-bytes)	Write Size (1-byte)	Write Payload (X-bytes)	
Response	3	0xE210	FS Response Code (1 byte)	Size Written (1-byte)				
ANTFS_FileWriteRelative()	5+X	0xE100	0xE211	File Handle	Write Size	Write Payload		

	6.3.12 p(48)				(1-byte)	(1-byte)	(X-bytes)
	Response	3	0xE211	FS Response Code (1 byte)	Size Written (1-byte)		
	ANTFS_FileGetSize() 6.3.13 p(49)	4	0xE100	0xE213	File Handle (1-byte)		
	Response	4	0xE213	FS Response Code (1 byte)	File Size (4-bytes)		
	ANTFS_FileGetSizeInMem() 6.3.14 p(51)	4	0xE100	0xE215	File Handle (1-byte)		
	Response	6	0xE215	FS Response Code (1 byte)	File Size (4-bytes)		
	ANTFS_FileGetSpecificFlags() 6.3.15 p(52)	4	0xE100	0xE214	File Handle (1-byte)		
	Response	3	0xE214	FS Response Code (1 byte)	File Flags (1-byte)		
	ANTFS_FileGetSystemTime() 6.3.16 p(53)	3	0xE100	0xE23D			
	Response	5	0xE23D	Current Time (4-bytes)			
Crypto Commands	ANTFS_CryptoAddUserKeyIndex() 6.4.1 p(54)	34	0xE245	Key Index (1-byte)	User Key (32-bytes)		
	Response	4	0xE000	0xE245	FS Response Code (1 byte)		
	ANTFS_CryptoSetUserKeyIndex() 6.4.2 p(55)	2	0xE246	Key Index (1-byte)			
	Response	4	0xE000	0xE246	FS Response Code (1 byte)		
	ANTFS_CryptoSetUserKeyVal() 6.4.3 p(56)	33	0xE247	User Key (32-bytes)			
	Response	4	0xE000	0xE247	FS Response Code (1 byte)		
FIT Commands	ANTFS_FitFileIntegrityCheck() 6.5.1 p(57)	2	0xE250	File Handle (1-byte)			
	Response	4	0xE000	0xE250	FS Response Code (1 byte)		
ANTFS Commands	ANTFS_OpenBeacon() 6.6.1 p(58)	1	0xE231				
	Response	4	0xE000	0xE231	FS Response Code (1 byte)		
	ANTFS_CloseBeacon() 6.6.2 p(59)	1	0xE232				
	Response	4	0xE000	0xE232	FS Response Code (1 byte)		
	ANTFS_ConfigBeacon()	9	0xE233	Beacon Device Type	Beacon Manuf Type	Beacon Auth Type	Beacon Status

	6.6.3 p(60)			(1-byte)	(2-bytes)	(1-byte)	(1-byte)
	Response	4	0xE000	0xE233	FS Response Code (1 byte)		
	ANTFS_SetAuthString() 6.6.4 p(62)	2+X	0xE234	String Enum (1-byte)	String (0-16 bytes)		
	Response	4	0xE000	0xE234	FS Response Code (1 byte)		
	ANTFS_SetBeaconState() 6.6.5 p(64)	2	0xE235	Beacon Status (1-byte)			
	Response	4	0xE000	0xE235	FS Response Code (1 byte)		
	ANTFS_PairResponse() 6.6.6 p(65)	2	0xE236	Response (1-byte)			
	Response	4	0xE000	0xE236	FS Response Code (1 byte)		
	ANTFS_SetLinkFrequency() 6.6.7 p(66)	2	0xE237	Channel Number (1-byte)	RF Freq (1-byte)		
	Response	4	0xE000	0xE237	FS Response Code (1 byte)		
	ANTFS_SetBeaconTimeout() 6.6.8 p(67)	2	0xE238	Timeout (1-byte)			
	Response	4	0xE000	0xE238	FS Response Code (1 byte)		
	ANTFS_SetPairingTimeout() 6.6.9 p(68)	2	0xE239	Timeout (1-byte)			
	Response	4	0xE000	0xE239	FS Response Code (1 byte)		
	ANTFS_EnableRemoteFileCreate() 6.6.10 p(69)	2	0xE23A	Enable (1-byte)			
	Response	4	0xE000	0xE23A	FS Response Code (1 byte)		
ANTFS Requests	ANTFS_GetCmdPipe() 6.7.1 p(70)	5	0xE100	0xE23B	Offset (1-byte)	Read Size (1-byte)	
	Response	2+X	0xE23B	Size Read (1-byte)	Read Payload (X-bytes)		
	ANTFS_SetCmdPipe() 6.7.2 p(71)	5+X	0xE100	0xE23C	Offset (1-byte)	Write Size (1-byte)	Write Payload (X bytes)
	Response	2	0xE23C	Size Written (1-byte)			

6.1 Memory Device Commands

6.1.1 MESG_MEMDEV_EEPROM_INIT (0xE220)

BOOL ANTFS_InitEEPROMDevice(USHORT usPageSize_, UCHAR ucAddressConfig_);

Init command must be called prior to issuing any FS command or requests. Currently, only SPI interface (min 2MHz rate) is supported with EEPROM devices. Data is clocked out MSB first and the clock polarity (CPOL/CPHA) is 1 and 1 respectively. The chips select line (CS) is used and toggles between write accesses and on boundaries.

Configuration fields should be specified from information found from the EEPROM datasheet. Successful initialization results in FS_NO_ERROR response code.

Page Write Size is the physical page write boundary of the EEPROM Device. For EEPROM, this is considered the maximum number of bytes that can be written in one pass and it must be 2^x value. The address bytes configuration field specifies the required number of bytes used to address the physical memory location on the EEPROM. For example, a 1MBit EEPROM device requires 3 address bytes.

Table 3. Initialize Memory Device Command Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x20	
Page Write Size	USHORT	0..MAX(USHORT)	Number of bytes that can be contiguously written to EEPROM at a time
Address Bytes Configuration	UCHAR	0-1	0: three byte addressing 1: two byte addressing

Table 4. Initialize Memory Device Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x20	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR. Otherwise command was not successful

Example Usage

```
ANTFS_InitEEPROMDevice(256, 0);     // 256 bytes per page, 3 byte address schem
```

6.2 File System Commands

6.2.1 MESG_FS_INIT_MEMORY (0xE200)

BOOL ANTFS_InitFSMemory();

Initializes existing file system from saved directory information in NVM. Unsaved information on open files will be lost. Init command must be called prior to using any FS related commands or requests. Init command must also be called after issuing FS Format Memory. Also resets encryption key used for crypto operations. Successful initialization results in FS_NO_ERROR response code.

Table 5. Initialize File System Memory Command Description

Parameters	Type	Range	Description
Size	UCHAR	1	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x00	

Table 6. Initialize File System Memory Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x00	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR. Otherwise command was not successful

Example Usage

```
ANTFS_InitFSMemory();// Initialize file system state variables
```

6.2.2 MESG_FS_FORMAT_MEMORY (0xE201)

BOOL ANTFS_FormatFSMemory(USHORT usNumberOfSectors_, USHORT usPagesPerSector);

Command used to create a new empty file system. Any existing directory information and files will be discarded. Minimum number of sectors must be 2 (1 for directory and 1 for each file). Successful format operation results in FS_NO_ERROR response code.

Number of Sectors * Num Pages per sector * Page Size must not exceed the size of the memory device. If inappropriate values are entered, format may succeed, but FS will be unusable. The FS sector size is defined by the Num Pages per Sector field where the page size is defined in MEMDEV_EEPROM_INIT for EEPROM device.

Specifying the correct number of sectors and pages per sector is application dependent. If the system is expected to have a small number of large files then it may be more efficient to have fewer, larger sectors. Conversely, a large number of small files would require a setup that has more, smaller sector defined. However, the efficiencies gained will vary greatly per application and must be evaluated on a case by case basis.

Table 7. Format File System Command Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x01	
Sectors	USHORT	0..MAX(USHORT)	Number of sectors in file system.
Pages per sector	USHORT	0..MAX(USHORT)	Number of pages per sector.

Table 8. Initialize File System Memory Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x01	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR. Otherwise command was not successful

Example Usage

```
ANTFS_FormatFSMemory(32, 8);        // 32 sectors, 8 pages per sector
                                    // Assuming 256 byte page, EEPROM size =
                                    // 256X32X16 = 65536 bytes
```

6.2.3 MESG_FS_DIRECTORY_SAVE (0xE207)

BOOL ANTFS_SaveDirectory();

Save all open file information into the directory NVM. This should be called before device power off or any unsaved data will be lost. Successful save operation results in FS_NO_ERROR response code.

Table 9. Directory Save Command Description

Parameters	Type	Range	Description
Size	UCHAR	1	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x07	

Table 10. Directory Save Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x07	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR. Otherwise command was not successful

Example Usage

```
ANTFS_SaveDirectory();      // Updates directory with any changes in file system
```

6.2.4 MESG_FS_DIRECTORY_REBUILD (0xE209)

BOOL ANTFS_DirectoryRebuild();

Rebuilds FS directory and condenses directory size by removing invalidated entries. Rebuilding directory also updates auto file index counter.

This command is automatically called whenever a directory is requested from the EEPROM. Any time a number of files is deleted from the file system this command should be called.

Successful rebuild results in FS_NO_ERROR response code.

Table 11. Directory Re-build Command Description

Parameters	Type	Range	Description
Size	UCHAR	1	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x09	

Table 12. Directory Re-build Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x09	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR. Otherwise command was not successful

Example Usage

```
ANTFS_DirectoryRebuild();          // Rebuild FS directory
```

6.2.5 MESG_FS_FILE_DELETE (0xE20C)

BOOL ANTFS_FileDelete(UCHAR ucFileHandle_);

Delete existing open file. Delete permission must be set on file handle. On successful deletion (FS_NO_ERROR response code), the file handle is freed. If FS_MEMORY_WRITE_ERROR is returned, memory occupied by file is lost but the handle is freed. Any other response codes results in file deletion failure and the file handle remains associated to the open file.

Table 13. Delete File Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0C	
File Handle	UCHAR	0..MAX(UCHAR)	File handle of opened file.

Table 14. Delete File Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x0C	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_MEMORY_WRITE_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_FileDelete(7);      // Delete file associated with file handle 7.
```

6.2.6 MESG_FS_FILE_CLOSE (0xE20D)

BOOL ANTFS_FileClose(UCHAR ucFileHandle_);

Close an open file handle. Any open file handle information is saved to the directory.

FS_NO_ERROR response code is returned if file close operation is successful (with the exceptions discussed in FIT File and Crypto FIT File). Any other response code resulted in file close failure and the file handle still assigned to the file.

Crypto File

When specifying a crypto handle (handle with encryption/decryption properties), the crypto handle is freed/unassigned when the file is closed successfully (FS_NO_ERROR, with exceptions discussed in Crypto FIT File).

FIT File

When closing a handle with write permissions (write/erase/append flag) to a FIT file, the file’s FIT structure is updated with changes made to the file. The update consists of updating the data size field in the FIT header and appending a 2 byte CRC to the end of the file. File update failures result in the following response codes but allows the file to be closed:

FS_FIT_FILE_HEADER_ERROR - FIT file has incorrect header information or < minimum header size
FS_FIT_FILE_SIZE_INTEGRITY_ERROR - FIT file size does not match calculated size from FIT header info
FS_MEMORY_NO_FREE_SECTORS_ERROR – no free memory to append CRC to file

When closing a handle with read-only permissions to a FIT file, no changes are made to the file unless there is a non-read locked write handle still opened on the file. If the file is valid, then the file is resized by removing the 2-byte CRC at the end of the file.

Crypto FIT File

Closing a crypto handle to an encrypted FIT file results in the same behaviour described in FIT file. When closing an encrypted FIT file from a non encryption/decryption handle, the file contents will not change, however a non-FS_NO_ERROR response code will be returned.

Table 15. Close File Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0D	
File Handle	UCHAR	0..MAX(UCHAR)	File handle of opened file.

Table 16. Close File Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x0D	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_FIT_FILE_HEADER_ERROR FS_FIT_FILE_SIZE_INTEGRITY_ERROR FS_MEMORY_NO_FREE_SECTORS_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_FileClose(7);  // Delete file associated with file handle 7.
```

6.2.7 MESG_FS_FILE_SET_SPECIFIC_FLAGS (0xE212)

BOOL ANTFS_SetFileSpecificFlags(UCHAR ucFileHandle_, UCHAR ucFlags);

Update application defined flags on file, but it is not saved to directory NVM. On success, FS_NO_ERROR is returned. File flags will be written back to the EEPROM after a directory save command, or when a file handle is closed.

Table 17. Set Specific File Flags Command Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x12	
File Handle	UCHAR	0..MAX(UCHAR)	File handle of opened file.
Flag	UCHAR	Bit map	Defined by specific file data type.

Table 18. Set Specific File Flags Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x12	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_SetFileSpecificFlags(7, 0x01); // Set bit 0 of application defined file flag on
handle 7.
```

6.2.8 MESG_FS_DIRECTORY_READ_LOCK (0xE216)

UCHAR ANTFS_DirectoryReadLock(BOOL bLock_);

When directory is locked, directory information is prevented from changing. When directory is unlocked, directory information is allowed to change. Locking the directory prevents the following FS operations from succeeding:

MESG_FS_DIRECTORY_SAVE (0xE207)
MESG_FS_FILE_CREATE (0xE20A)
MESG_FS_FILE_DELETE (0xE20C)
MESG_FS_FILE_CLOSE (0xE20D), when closing write handle or read handle with archive permissions
MESG_FS_DIRECTORY_REBUILD (0xE209)

Attempting to lock a directory that is already locked will result in an error. Attempting to unlock a directory that is already unlocked will result in an error.

It may be useful to lock the directory before requesting it to ensure that the directory is not modified by the ANT-FS Host simultaneously.

Table 19. Read Lock Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x16	
Lock	UCHAR	0-1	0 = unlock 1 = lock

Table 20. Read Lock Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x16	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_DirectoryReadLock(TRUE);      // Lock directory.
```

6.2.9 MESG_FS_SYSTEM_TIME (0xE23D)

BOOL ANTFS_SetSystemTime(ULONG ulTime_);

When this message ID is used in a command message, the specified starting value of the system time to be used in FS can be set. If successful, FS_NO_ERROR is returned.

Table 21. System Time Command Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x3D	
System time	ULONG	0-MAX(ULONG)	Time base to be used for file system.

Table 22. System Time Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x3D	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_SetSystemTime(12389328); // Set time to 12389328s.
```

6.3 File System Requests

6.3.1 MESG_FS_GET_USED_SPACE (0xE202)

ULONG ANTFS_GetUsedSpace();

Returns number of used bytes in FS in sector sized increments. On success, returns FS_NO_ERROR response code and the used space value. If any other response code is returned, an invalid used space size value is returned (0xFFFFFFFF).

Table 23. Used Space Request Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Extended ID Response	UCHAR	0xE1	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x02	

Table 24. Used Space Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x02	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
Used space	ULONG	0-MAX(ULONG)	Bytes used by file system. 0xFFFFFFFF indicates an error condition.

Example Usage

```
ULONG ulSpace = ANTFS_GetUsedSpace();

if(ulSpace != 0xFFFFFFFF)
       printf(“Used space is %d bytes\n”, ulSpace);
else
       printf(“Error %d\n”, ANTFS_GetLastError());
```

6.3.2 MESG_FS_GET_FREE_SPACE (0xE203)

ULONG ANTFS_GetFreeSpace();

Returns number of free bytes in FS in sector sized increments. On success, returns FS_NO_ERROR response code and the free space value. If any other response code is returned, an invalid free space size value is returned (0xFFFFFFFF).

Table 25. Free Space Request Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x03	

Table 26. Free Space Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x03	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
Used space	ULONG	0-MAX(ULONG)	Free byte in file system. 0xFFFFFFFF indicates an error condition.

Example Usage

```
ULONG ulSpace = ANTFS_GetFreeSpace();

if(ulSpace != 0xFFFFFFFF)
       printf(“Space left is %d bytes\n”, ulSpace);
else
       printf(“Error %d\n”, ANTFS_GetLastError());
```

6.3.3 MESG_FS_FIND_FILE_INDEX (0xE204)

USHORT ANTFS_FindFileIndex(UCHAR ucFileDataType_, UCHAR ucFileSubType_, USHORT usFileNumber_);

Return file index of first file in directory that matches specified identifier. On success, returns FS_NO_ERROR response code and the file index. If any other response code is returned, an invalid file index is returned (0x0000).

Table 27. Find File Index Request Description

Parameters	Type	Range	Description
Size	UCHAR	7	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x04	
File Data Type	UCHAR	0-MAX(UCHAR)	File Data Type is application defined. Please consult FIT documentation for FIT data types.
File Sub Type	UCHAR	0-MAX(UCHAR)	File Sub Type is application defined. Please consult FIT documentation for FIT file data types.
File Number	USHORT	0-MAX(USHORT)	File number in directory (user defined)

Table 28. File Index Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x04	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
File Index	USHORT	0-MAX(USHORT)	File index in FS directory

Example Usage

```
USHORT usFileIndex = ANTFS_FindFileIndex(1,2,3);

if(usFileIndex!= 0x0000)
       printf(“File index is %dn”, usFileIndex);
else
       printf(“Error %d\n”, ANTFS_GetLastError());
```

6.3.4 MESG_FS_DIRECTORY_READ_ABSOLUTE (0xE205)

UCHAR ANTFS_ReadDirectoryAbsolute(ULONG ulOffset_, UCHAR ucSize_, UCHAR* pucBuffer_);

Read from absolute offset into directory as if it were an ANTFS directory (16-byte blocks). First entry in directory consists of a 16-byte ANTFS directory header followed by 16-byte directory entries of each file in the FS. It is recommended to rebuild the directory before using this command in order to avoid invalid entries.

On successful reads (FS_NO_ERROR), the returned number of bytes read as well as the payload is returned. Reading past the end of the directory results in FS_EOF_REACHED_ERROR, however the number of read bytes and the payload prior to reaching EOF is returned.

Table 29. Directory Read Absolute Request Description

Parameters	Type	Range	Description
Size	UCHAR	8	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x05	
Offset	ULONG	0-MAX(ULONG)	Offset in bytes into the directory structure.
Size	UCHAR	0-MAX(UCHAR)	Number of bytes to read from directory

Table 30. Directory Read Absolute Output Description

Parameters	Type	Range	Description
Size	UCHAR	3+X	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x05	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_EOF_REACHED_ERROR Otherwise command was not successful
Size Read	UCHAR	0-MAX(UCHAR)	File index in FS directory
Payload	UCHAR[X]	Array[MAX(UCHAR)]	Buffer of bytes read from directory

Example Usage

```
UCHAR aucSystemTime[4]; // Bytes will be read into this array.
UCHAR ucError = ANTFS_ReadDirectoryAbsolute(8,4,aucSystemTime);

if(ucError == FS_NO_ERROR)
{
       ULONG* pulTime = aucSystemTime;
       printf("System time in directory header = %d\n", *pulTime);
}

else
{
       printf("Error reading from directory %d\n", ucError);
}
```

6.3.5 MESG_FS_DIRECTORY_READ_ENTRY (0xE206)

UCHAR ANTFS_DirectoryReadEntry (USHORT usFileIndex_, UCHAR* ucFileDirectoryBuffer_);

Return FS_NO_ERROR response code and ANTFS directory entry for the file matching the specified file index on successful match. Any other error codes results in an invalid entry being returned (all 0xFF). This command retrieves the most up to date directory entry, even if the entry has not yet been saved to NVM.

Table 31. Directory Read Entry Request Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x06	
File Index	USHORT	0-MAX(USHORT)	Index of file for which to retrieve directory entry.

Table 32. Directory Read Entry Output Description

Parameters	Type	Range	Description
Size	UCHAR	18	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x06	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
Directory	UCHAR[16]	Array[16]	ANTFS directory structure

Example Usage

```
DirectoryEntryStruct stDirEntry;
UCHAR ucError = ANTFS_DirectoryReadEntry(1,(UCHAR*) &stDirEntry);

if(ucError == FS_NO_ERROR)
{
printf("File Index 1 file size = %d\n", stDirEntry.ulFileSize);
}
else
{
       printf("Error reading from directory %d\n", ucError);
}
```

6.3.6 MESG_FS_DIRECTORY_GET_SIZE (0xE208)

ULONG ANTFS_DirectoryGetSize();

Returns size in bytes as if it were an ANTFS directory (16-byte blocks). On success, returns FS_NO_ERROR response code and the ANTFS directory size value. If any other response code is returned, an invalid ANTFS directory size value is returned (0xFFFFFFFF).

Table 33. Directory Size Request Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x08	

Table 34. Directory Size Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x08	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
Directory size	ULONG	0-MAX(ULONG)	Size of directory in bytes 0xFFFFFFFF indicates invalid

Example Usage

```
ULONG ulDirectorySize= ANTFS_DirectoryGetSize();
if(ulDirectorySize!= 0xFFFFFFFF)
       printf(“Directory size is %d\n”, ulDirectorySize);
else
       printf(“Error %d\n”, ANTFS_GetLastError());
```

6.3.7 MESG_FS_FILE_CREATE (0xE20A)

USHORT ANTFS_FileCreate(USHORT usFileIndex_, UCHAR ucFileDataType_, ULONG ulFileIdentifier_, UCHAR ucFileDataTypeSpecificFlags_, UCHAR ucGeneralFlags);

Allocates a free sector and saves directory entry of the new file. If 0x0000 is supplied as file index, FS will auto generate a valid index for the file, otherwise a valid index must manually be supplied. Directory entry information to be specified comprises of the following (in order):

1 byte File Data Type
3 byte File Identifier
1 byte File Data Type Specific Flags
1 byte General Flags

If file creation is successful (FS_NO_ERROR is returned), the file index assigned to the created file is returned. Any other errors results in an invalid file index being returned (0x0000). If a auto generation of the file index fails, this command should be retried.

Crypto File

When creating an encrypted file, the user key must be specified beforehand and the crypto flag bit (0x04) must be specified in the 1 byte general flags field. Also, the crypto handle must be free and not be in use by any opened file.

FIT File

When creating a FIT file, the File Data Type field must be specified as 0x80.

Table 35. Create File Request Description

Parameters	Type	Range	Description
Size	UCHAR	11	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x0A	
File Index	USHORT	0-MAX(USHORT)	Index at which to create file. Setting this number to 0 will result in the next available index being used.
Data Type	UCHAR	0..MAX(UCHAR)	Application defined data type. Please consult the FIT documentation for FIT specific data types. 0x80 = FIT file type
Identifier	UCHAR[3]	Array[3]	Application defined file identifier. Please consult the FIT documentation for FIT file identifiers. Please note that some applications, including FIT, use parts of this identifier to define a file sub type.
Data Type Specific Flags	UCHAR	Bit map	Application specific file flags.
General Flags	UCHAR	Bit map	General File Flags 76543210 ||||||XX-→Reserved |||||X---→Crypto ||||X----→Append |||X-----→Archive ||X------→Erase |X-------→Write X--------→Read Please consult ANTFS specification for complete

			details.

Table 36. Create File Output Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0A	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
File Index	USHORT	1..MAX(USHORT)	File index of file created.

Example Usage

```
USHORT usFileIndex;
usFileIndex = ANTFS_FileCreate(0, 1, 0x00ABCDEF, 0x00, 0xE0); // r/w/e
If(usFileIndex != 0x0000)
{
       printf(“File created at index %d\n”, usFileIndex);
}
else
{
       printf(“Error creating file. Error: %d\n”, ANTFS_GetLastError());
}
```

6.3.8 MESG_FS_FILE_OPEN (0xE20B)

UCHAR ANTFS_FileOpen(USHORT usFileIndex_, UCHAR ucOpenFlags_);

Open existing file in FS, open flags need to be supported by the file. By default, read and write pointers are set at the beginning of the file. If append flag is set in Open Flags parameter, then the write pointer is set to the end of the file.

The following table describes the open flags.

Define	ID	Description
FS_FILE_OPERATION_OVERRIDE_ATTRIBUTES	0x01	Allows user to force open parameters (for example to write to a read-only file).
ANTFS_GENERAL_FLAGS_CRYPTO	0x04	Open with crypto handle
ANTFS_GENERAL_FLAGS_APPEND_ONLY	0x08	Open handle to end of file and append
ANTFS_GENERAL_FLAGS_ARCHIVED	0x10	Open archived file
ANTFS_GENERAL_FLAGS_CAN_ERASE	0x20	Handle can erase file
ANTFS_GENERAL_FLAGS_CAN_WRITE	0x40	Handle can write to file
ANTFS_GENERAL_FLAGS_CAN_READ	0x80	Handle can read from file

If file open is successful (FS_NO_ERROR, with the exceptions discussed in FIT File and Crypto FIT File), the file handle number is returned. Any other response code results in file open failure and the file handle returned being invalid (0xFF).

Crypto File

When dealing with an encrypted file (file general flags has crypto bit 0x04 present), to be able to decrypt (read) or encrypt (write) data, the user key must be specified beforehand. To allow file encryption/decryption operation on a handle, the crypto bit (0x04) must be set in Open flags parameter. This effectively assigns the handle as the encryption/decryption handle (crypto handle). Only 1 encryption/decryption handle can be assigned.

An encrypted file can be opened as is without the crypto handle. An unencrypted file cannot be opened with the crypto handle.

FIT File

When opening a handle with write permissions (write/erase/append flag) to a valid FIT file, the file is resized by removing the 2-byte file CRC at the end of the file. If the file is invalid, then the file is not resized and the following response codes are returned, but the file is allowed to be opened.

FS_FIT_FILE_HEADER_ERROR - FIT file has incorrect header information or < minimum header size
FS_FIT_FILE_SIZE_INTEGRITY_ERROR -FIT file size does not match calculated size from FIT header info

When opening a read-only handle to a FIT file, the validity of the FIT file is checked. If the file is invalid, the following response codes are returned, but the file is allowed to be opened.

FS_FIT_FILE_HEADER_ERROR - FIT file has incorrect header information or < minimum header size
FS_FIT_FILE_SIZE_INTEGRITY_ERROR - FIT file size does not match calculated size from FIT header info

Crypto FIT File

When dealing with an encrypted FIT file, a crypto handle should be used to properly access the file. If the encrypted FIT file is opened without this handle, then the file check will fail but still allow the file to be opened and not be resized (when opened for writing).

Table 37. Open File Request Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x0B	
File Index	USHORT	0-MAX(USHORT)	Index of file to open
Open Flags	UCHAR	Bit Map	Open Flags Definition: 76543210 ||||||X→ OVERRIDE ATTRIBUTES |||||X--→ CRYPTO ||||X---→ APPEND ONLY |||X----→ ARCHIVED ||X-----→ CAN ERASE |X------→ CAN WRITE X-------→ CAN READ

Table 38. Open File Output Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0B	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_FIT_FILE_HEADER_ERROR FS_FIT_FILE_SIZE_INTEGRITY_ERROR Any other: command was not successful
File Handle	UCHAR	0-MAX(UCHAR)	File handle to opened file. 0xFF file handle is invalid.

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
       printf(“File opened handle = %d\n”, ucFileHandle);
else
       printf(“Error error opening file. Error %d\n”, ANTFS_GetLastError());
```

6.3.9 MESG_FS_FILE_READ_ABSOLUTE (0xE20E)

UCHAR ANTFS_FileReadAbsolute(UCHAR ucFileHandle_, ULONG ulOffset_, UCHAR ucReadSize_, UCHAR* pucReadBuffer_);

Read from absolute offset into a file. File must be opened for reading beforehand. After reading, read pointers positioned at the end of the bytes read.

On successful reads (FS_NO_ERROR), the returned number of bytes read as well as the payload is returned. Reading past the end of the file results in FS_EOF_REACHED_ERROR, however the number of read bytes and the payload prior to reaching EOF is returned.

Crypto File

If reading from a crypto handle, the return payload contains the file’s decrypted content.

FIT File or Crypto FIT File

If file is read-locked, reading from the write handle (handle that is allowed to write) is prevented (returns with response code FS_BAD_PERMISSIONS_ERROR). A write handle is read-locked when there is another handle that has the file opened for read-only.

Table 39. Read File Absolute Request Description

Parameters	Type	Range	Description
Size	UCHAR	9	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x0E	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle
Offset	ULONG	0-MAX(ULONG)	Byte offset into file.
Read Size	UCHAR	0-32	Number of bytes to read

Table 40. Read File Absolute Output Description

Parameters	Type	Range	Description
Size	UCHAR	3+X	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0E	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_EOF_REACHED_ERROR FS_BAD_PERMISSIONS_ERROR Any other: command was not successful
Size Read	UCHAR	0-32	Number of bytes read
Payload	UCHAR[X]	Array[X]	Buffer of requested bytes. Max size of buffer is 32 bytes

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
```

```
{
        UCHAR aucBuffer[16];
        UCHAR usReadBytes = ANTFS_FileReadAbsolute(ucFileHandle,8,16, aucBuffer);
        printf("Read = %d bytes\n", usReadBytes);
}
else
{
        printf("Error reading file. Error %d\n", ANTFS_GetLastError());
}
```

6.3.10 MESG_FS_FILE_READ_RELATIVE (0xE20F)

UCHAR ANTFS_FileReadRelative(UCHAR ucFileHandle_, UCHAR ucReadSize_, UCHAR* pucReadBuffer_);

Read from current read pointer position in file. File must be opened for reading beforehand. After reading, read pointers positioned at the end of the bytes read.

On successful reads (FS_NO_ERROR), the returned number of bytes read as well as the payload is returned. Reading past the end of the directory results in FS_EOF_REACHED_ERROR, however the number of read bytes and the payload prior to reaching EOF is returned.

Crypto File

If reading from a crypto handle, the return payload contains the file’s decrypted content.

FIT File or Crypto FIT File

If file is read-locked, reading from the write handle (handle that is allowed to write) is prevented (returns with response code FS_BAD_PERMISSIONS_ERROR). A write handle is read-locked when there is another handle that has the file opened for read-only.

Table 41. Read File Relative Request Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x0F	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle

Read Size	UCHAR	0-32	Number of bytes to read

Table 42. Read File Relative Output Description

Parameters	Type	Range	Description
Size	UCHAR	3+X	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x0F	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_EOF_REACHED_ERROR FS_BAD_PERMISSIONS_ERROR Any other: command was not successful
Size Read	UCHAR	0-32	Number of bytes read
Payload	UCHAR[X]	Array[X]	Buffer of requested bytes. Max size of buffer is 32

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
{
	UCHAR aucBuffer[16];
	UCHAR usReadBytes = ANTFS_FileReadAbsolute(ucFileHandle, 16, aucBuffer);
	printf(“Read = %d bytes\n”, usReadBytes);
}
else
{
	printf(“Error reading file. Error %d\n”, ANTFS_GetLastError());
}
```

6.3.11 MESG_FS_FILE_WRITE_ABSOLUTE (0xE210)

UCHAR ANTFS_FileWriteAbsolute(UCHAR ucFileHandle_, ULONG ulFileOffset_, UCHAR ucWriteSize_, const UCHAR* pucWriteBuffer_, UCHAR* ucBytesWritten_);

Write to absolute offset into a file. File cannot be written to if it was opened for reading by another handle (read-locked). If this is the case a FS_BAD_PERMISSIONS_ERROR (0x0E) will result. When using a non-crypto handle, write absolute cannot be used if file only opened for append operation. After writing, write positioned at the end of the bytes written.

On successful writes (FS_NO_ERROR), the number of bytes written is returned. If any writing failure occurs, the response code is returned along with the number of bytes managed to be written. Possible response codes are:

Crypto File

If writing to a crypto handle, the input payload is encrypted then written to FS.

Table 43. Write File Absolute Request Description

Parameters	Type	Range	Description
Size	UCHAR	9+X	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x10	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle
Offset	ULONG	0-MAX(ULONG)	Byte offset into file.
Write Size	UCHAR	0-32	Number of bytes to read
Payload	UCHAR[X]	Array[X]	Buffer of bytes to write. Max size of buffer is 32.

Table 44. Write File Absolute Output Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x10	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_MEMORY_NO_FREE_SECTORS_ERROR Any other: command was not successful
Bytes Written	UCHAR	0-32	Number of bytes successfully written to EEPROM.

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
{
       UCHAR aucBuffer[4] = {0xAA, 0xBB, 0xCC, 0xDD };

       // Write 4 bytes at offset 8 bytes
       UCHAR ucBytes = ANTFS_FileWriteAbsolute(ucFileHandle,8,4, aucBuffer);
       printf("Wrote = %d bytes\n", ucBytes);
}
else
{
       printf("Error reading file. Error %d\n", ANTFS_GetLastError());
}
```

6.3.12 MESG_FS_FILE_WRITE_RELATIVE (0xE211)

UCHAR ANTFS_FileWriteRelative(UCHAR ucFileHandle_, UCHAR ucWriteSize_, const UCHAR* pucWriteBuffer_, UCHAR* ucBytesWritten_);

Write to current write pointer position in file. If this is the case a FS_BAD_PERMISSIONS_ERROR (0x0E) will result. File cannot be written to if opened by another handle for reading (read-locked). After writing, write positioned at the end of the bytes written.

On successful writes (FS_NO_ERROR), the number of bytes written is returned. If any writing failure occurs, the response code is returned along with the number of bytes managed to be written.

Crypto File

If writing to a crypto handle, the input payload is encrypted then written to FS.

Table 45. Write File Relative Request Description

Parameters	Type	Range	Description
Size	UCHAR	5+X	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x11	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle
Write Size	UCHAR	0-32	Number of bytes to read

Payload	UCHAR[X]	Array[X]	Buffer of bytes to write. Max size of buffer is 32.

Table 46. Write File Relative Output Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x11	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR FS_MEMORY_NO_FREE_SECTORS_ERROR Any other: command was not successful
Bytes Written	UCHAR	0-32	Number of bytes successfully written to EEPROM.

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
{
	UCHAR aucBuffer[4] = {0xAA, 0xBB, 0xCC, 0xDD };
	// Write 4 bytes from file
	UCHAR ucBytes = ANTFS_FileWriteRelative(ucFileHandle,4, aucBuffer);
	printf(“Wrote = %d bytes\n”, ucBytes);
}
else
{
	printf(“Error reading file. Error %d\n”, ANTFS_GetLastError());
}
```

6.3.13 MESG_FS_FILE_GET_SIZE (0xE213)

ULONG ANTFS_FileGetSize(UCHAR ucFileHandle_);

Get size of open file in bytes. If successful, FS_NO_ERROR is returned along with the file size in bytes. If any other response code is returned, an invalid file size value is returned (0xFFFFFFFF).

Table 47. File Size Request Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x13	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle

Table 48. File Size Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x13	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
File Size	ULONG	0-MAX(ULONG)	Size of file in bytes (0xFFFFFFFF invalid).

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);

if(ucFileHandle != 0xFF)
{
       // Write 4 bytes from file
       ULONG ulSize = ANTFS_FileGetSize(ucFileHandle);
       printf("File size = %d bytes\n", ulSize);
}
else
{
       printf("Error reading file. Error %d\n", ANTFS_GetLastError());
}
```

6.3.14 MESG_FS_FILE_GET_SIZE_IN_MEM (0xE215)

ULONG ANTFS_FileGetSizeInMem(UCHAR ucFileHandle_);

Get size of the file in terms of the number of total bytes allocated to the file in the FS (size in memory). If successful, FS_NO_ERROR is returned along with the size in bytes. If any other response code is returned, an invalid file size value is returned (0xFFFFFFFF).

Table 49. File Size in Memory Request Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x15	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle

Table 50. File Size in Memory Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x15	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
File Size	ULONG	0-MAX(ULONG)	Size of file in bytes (0xFFFFFFFF invalid).

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);
if(ucFileHandle != 0xFF)
{

        // Write 4 bytes from file
        ULONG ulSize = ANTFS_FileGetSizeInMem(ucFileHandle);
        printf("File size in memory = %d bytes\n", ulSize);
}
else
```

```
{
        printf("Error reading file. Error %d\n", ANTFS_GetLastError());
}
```

6.3.15 MESG_FS_FILE_GET_SPECIFIC_FILE_FLAGS (0xE214)

UCHAR ANTFS_FileGetSpecificFlags(UCHAR ucFileHandle_);

Gets the application defined flags of opened file. If successful, FS_NO_ERROR is returned along with the application defined flags on the file. If any other response code is returned, flag value of 0x00 is returned.

Table 51. Specific File Flags Request Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x14	
File Handle	UCHAR	1-MAX(UCHAR)	Valid file handle

Table 52. Specific File Flags Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x14	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful
File Flags	UCHAR	0-MAX(UCHAR)	File specific flags

Example Usage

```
UCHAR ucFileHandle = ANTFS_FileOpen(1,0);
if(ucFileHandle != 0xFF)
{
        // Write 4 bytes from file
        ULONG ucFlags = ANTFS_FileGetSpecificFlags(ucFileHandle);
        printf("Specific Flags = 0x%02x\n", ucFlags);
```

```
}
else
{
        printf("Error reading file. Error %d\n", ANTFS_GetLastError());
}
```

6.3.16 MESG_FS_SYSTEM_TIME (0xE23D)

ULONG ANTFS_FileGetSystemTime();

When this message ID is used in a request message, the current system time used in FS is returned.

Table 53. System Time Request Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x3D	

Table 54. System Time Output Description

Parameters	Type	Range	Description
Size	UCHAR	6	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x3D	
System time	ULONG	0-MAX(ULONG)	System time in seconds.

Example Usage

```
ULONG ulTime = ANTFS_FileGetSystemTime();
printf("Time in FS = %ds\n", ulTime);
```

6.4 FS-Crypto Commands

6.4.1 MESG_FS_CRYPTO_ADD_USER_KEY_INDEX (0xE245)

UCHAR ANTFS_CryptoAddUserKeyIndex(UCHAR ucIndex_, UCHAR* pucKey_);

Adds specified user key to be stored in internal memory. Keys are enumerated by Key Index. Up to 10 keys can be used. If successfully stored, FS_NO_ERROR is returned.

Table 55. Crypto Add User Key Command Description

Parameters	Type	Range	Description
Size	UCHAR	34	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x45	
Key Index	UCHAR	0-9	Index to insert key into.
User Key	UCHAR[]	Array[32]	Array with user key.

Table 56. Crypto Add User Key Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x45	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
UCHAR aucKey[] = {    0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
                      0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,
                      0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
                      0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F );
UCHAR ucResponse = ANTFS_CryptoAddUserKeyIndex(0, aucKey);
if(ucResponse == FS_NO_ERROR)
       printf(“Key Set successfully\n”);
```

6.4.2 MESG_FS_CRYPTO_SET_USER_KEY_INDEX (0xE246)

UCHAR ANTFS_CryptoSetUserKeyIndex(UCHAR ucIndex_);

Specify stored user key (specified by Key Index) to be used by FS Encryption/Decryption process. Key remains active until reset, memory re-initialization via MESG_FS_INIT MEMORY or another key is specified. If key successfully selected, FS_NO_ERROR is returned.

Table 57. Crypto Use Command Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x46	
Key Index	UCHAR	0-9	Key Index to use for crypto operations.

Table 58. System Time Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x46	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
UCHAR ucResponse = ANTFS_CryptoSetUserKeyIndex(3);

if(ucResponse == FS_NO_ERROR)
       printf(“Key Set to index 3 successfully\n”);
```

6.4.3 MESG_FS_CRYPTO_SET_USER_KEY_VAL (0xE247)

UCHAR ANTFS_CryptoSetUserKeyVal(UCHAR* pucKey_);

Specify non-stored user key to be used by FS Encryption/Decryption process. Key remains active until reset, memory re-initialization via MESG_FS_INIT_MEMORY or another key is specified. Setting this command overrides any other key setting. If key successfully set, FS_NO_ERROR is returned.

Table 59. Crypto Add User Key Value Command Description

Parameters	Type	Range	Description
Size	UCHAR	33	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x47	
User Key	UCHAR[]	Array[32]	Array with user key.

Table 60. Crypto Add User Key Value Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x47	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
UCHAR aucKey[] = {    0x00,0x01,0x02,0x03,0x04,0x05,0x06,0x07,
                      0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,
                      0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,
                      0x18,0x19,0x1A,0x1B,0x1C,0x1D,0x1E,0x1F );

UCHAR ucResponse = ANTFS_CryptoAddUserKeyIndex(aucKey);
```

```
if(ucResponse == FS_NO_ERROR)
       printf(“Key Set successfully\n”);
```

6.5 Fit Commands

6.5.1 MESG_FS_FIT_FILE_INTEGRITY_CHECK (0xE250)

UCHAR ANTFS_FitFileIntegrityCheck(UCHAR ucFileHandle_);

When this command is issued, a file integrity check is performed on the selected FIT file by calculating the file 2 byte CRC and comparing it versus the appended 2 byte CRC, and by checking file size and the header. If file integrity is intact, then FS_NO_ERROR is returned.

Provided file handle must be pointing to a FIT file (0x80 data type in file directory entry) as well as opened as read-only. Performing an integrity check on a non FIT file and/or a write handle (write/erase/append open flags) is not allowed.

Table 61. Fit File Integrity Check Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x50	
File Handle	UCHAR	0-MAX(UCHAR)	File handle of opened file.

Table 62. Fit File Integrity Check Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x50	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
UCHAR ucResponse = ANTFS_FitFileIntegrityCheck(ucFileHandle);

if(ucResponse == FS_NO_ERROR)
       printf(“Fit file is intact\n”);
```

6.6 ANT-FS Commands

6.6.1 MESG_FS_ANTFS_OPEN (0xE231)

UCHAR ANTFS_OpenBeacon();

Starts ANT-FS Beacon on all channels that have been enabled as ANT-FS using the command MESG_FS_ANTFS_SET_LINK_FREQ.

Table 63. Open Beacon Command Description

Parameters	Type	Range	Description
Size	UCHAR	1	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x31	

Table 64. Open Beacon Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x31	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
…
ANTFS_SetLinkFrequency(0,72); // Configure channel 0 to be ANT-FS, 2472MHz
ANTFS_SetLinkFrequency(1,57); // Configure channel 1 to be ANT-FS, 2457MHz
```

```
…
ANTFS_OpenBeacon();// Open ANT-FS beacon on channel 0 and 1
if(ucResponse == FS_NO_ERROR)
       printf(“ANT-FS Beacons Active\n”);
```

6.6.2 MESG_FS_ANTFS_CLOSE (0xE232)

UCHAR ANTFS_CloseBeacon();

Stop all active beacons, regardless of the state they are in.

Table 65. Close Beacon Command Description

Parameters	Type	Range	Description
Size	UCHAR	1	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x32	

Table 66. Close Beacon Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x32	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
ANTFS_CloseBeacon();
if(ucResponse == FS_NO_ERROR)
       printf(“ANT-FS Beacons Closed\n”);
```

6.6.3 MESG_FS_ANTFS_CONFIG_BEACON (0xE233)

UCHAR ANTFS_ConfigBeacon(USHORT usDeviceType_, USHORT usManufacturer_, UCHAR ucAuthType_, UCHAR ucBeaconStatus_);

Configure beacon parameters. Please note that changing beacon parameters will also change the behavior of the beacon. For example, if the channel period indicated in the beacon is changed, the channel period will also change without the requirement to send a separate channel period command.

The authentication supported by the Client device is indicated by the Beacon Auth Type field. The criteria used by the Client to accept or reject a Host request is governed by the table below.

Table 67. Beacon Authentication Logic

Beacon Indicates→		Passthrough	Pair	Passkey
Host Requests→	Passthrough	Pass	Fail	Fail
	Pair	Pass	Pass	Pass
	Passkey	Pass	Fail	Pass

Table 68. Configure Beacon Command Description

Parameters	Type	Range	Description
Size	UCHAR	9	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x33	
Beacon Device Type	USHORT	0-MAX(USHORT)	Identifier assigned to specific device. This identifier is generally assigned by the manufacturer.
Beacon Manufacturer ID	USHORT	0-MAX(USHORT)	Identifies the manufacturer of the transmitting device. For ANT+ implementations this number is assigned by ANT. Please contact ANT for details.
Beacon Auth Type	UCHAR	0-MAX(UCHAR)	0 – passthrough 2 – pairing 3 – passkey
Beacon Status	UCHAR	0-MAX(UCHAR)	For details of the status byte please refer to the ANT-FS Technical Specification.

			76543210 |||||***-→ Beacon Channel Period ||||*----→ Pairing enabled |||*-----→ Upload enabled ||*------→ Data Available **-------→ Reserved

Table 69. Config Beacon Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x33	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
USHORT usDeviceType = 1;            // Assigned by manufacturer
USHORT usManufacturer = 257;        // Assigned by ANT+
UCHAR ucAuthType = 3;               // Passkey
UCHAR ucBeaconStatus = 0x2B;        // Data available
                                    // No Upload
                                    // Pairing enabled
                                    // 4Hz channel period

UCHAR ucResponse = ANTFS_ConfigBeacon(usDeviceType, usManufacturer, ucAuthType,
ucBeaconStatus);

if(ucResponse == FS_NO_ERROR)
       printf(“ANT-FS Beacon Configured\n”);
```

6.6.4 MESG_FS_ANTFS_SET_AUTH_STRING (0xE234)

UCHAR ANTFS_SetFriendlyName(UCHAR ucLength_, const UCHAR* pucString_);

Or

UCHAR ANTFS_SetPasskey(UCHAR ucLength_, const UCHAR* pucString_);

Set the friendly name or authentication passkey strings. The friendly name is used in the authentication states to identify the client to the host using a readable name. The authentication passkey is used by the client when the host attempts to use the passkey method of authentication. The passkey will also be sent to the host upon a successful pairing authentication.

Table 70. Set Authentication String Command Description

Parameters	Type	Range	Description
Size	UCHAR	2+X	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x34	
String Type	UCHAR	0-1	Differentiate what type of string is being sent. 0 – Friendly Name 1 – Passkey
String	UCHAR[X]	Array[16(max)]	Friendly name or passkey bytes.

Table 71. Set Authentication String Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x34	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
const UCHAR aucPasskey = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF, 0x11, 0x22};

UCHAR ucResponse = ANTFS_SetFriendlyName (3, “Bob”);

if(ucResponse == FS_NO_ERROR)
       printf(“Friendly name set\n”);

ucResponse = ANTFS_SetPasskey (8, aucPasskey );

if(ucResponse == FS_NO_ERROR)
       printf(“Passkey set\n”);
```

6.6.5 MESG_FS_ANTFS_SET_BEACON_STATE(0xE235)

UCHAR ANTFS_SetBeaconState(UCHAR ucBeaconStatus_)

Set beacon status byte 1. Please note that changing beacon parameters will also change the behavior of the beacon. For example, if the channel period indicated in the beacon is changed, the channel period will also change without the requirement to send a separate channel period command.

Table 72. Set Beacon State Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x35	
Beacon Status	UCHAR	0-MAX(UCHAR)	For details of the status byte please refer to the ANT-FS Technical Specification. 76543210 |||||***-→ Beacon Channel Period ||||*-----→ Pairing enabled |||*------→ Upload enabled ||*-------→ Data Available **--------→ Reserved

Table 73. Set Beacon State Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x35	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
UCHAR ucBeaconStatus = 0x2B;        // Data available
                                    // No Upload
                                    // Pairing enabled
                                    // 4Hz channel period

UCHAR ucResponse = ANTFS_SetBeaconState (ucBeaconStatus);

if(ucResponse == FS_NO_ERROR)
       printf(“Beacon set\n”);
```

6.6.6 MESG_FS_ANTFS_PAIR_RESPONSE (0xE236)

BOOL ANTFS_PairResponse(BOOL bAccept_)

Respond to pairing request event (MESG_FS_ANTFS_EVENT_PAIR_REQUEST). The pairing response should generally come from the user of the client device through some type of user interface. If pairing is not enabled in the beacon the pairing request will be automatically rejected by the client, without the need for the external host to send this command.

Table 74. Pairing Response Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x36	
Pairing Response	UCHAR	0-1	Accept or reject pairing request. 0 – Reject 1 – Accept

Table 75. Pairing Response Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x36	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
switch (ucEvent_)
{
        …
        case MESG_FS_ANTFS_EVENT_PAIR_REQUEST:
        {
                // Accept or reject generally involves a UI operation
                ANTFS_PairResponse(TRUE);
                break;
        }
        …
}
```

6.6.7 MESG_FS_ANTFS_SET_LINK_FREQ (0xE237)

UCHAR ANTFS_SetLinkFrequency(UCHAR ucChannelNumber_, UCHAR ucFrequency_)

Set the RF frequency at which the LINK beacon will operate. The ANT-FS channel may be disabled by setting the RF frequency to 0xFF. This effectively detaches the ANT-FS engine from the channel, without otherwise modifying it. If the channel is opened all regular channel messages will cease to be filtered by ANT-FS and will be sent directly to the application MCU for processing. For ANT+ implementations please consult the specific ANT+ device profile for assigned LINK frequency.

Table 76. Set Link Frequency Command Description

Parameters	Type	Range	Description
Size	UCHAR	3	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x37	
Channel Number	UCHAR	0-7	ANT channel number.
RF Freq	UCHAR	0-MAX(UCHAR)	RF Frequency (+2400MHz) 0xFF = Disable

Table 77. Set Link Frequency Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x37	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
// Set channel 0 link frequency to 2457MHz
UCHAR ucResponse = ANTFS_SetLinkFrequency(0, 57);

if(ucResponse == FS_NO_ERROR)
       printf(“Link Frequency set\n”);
```

6.6.8 MESG_FS_ANTFS_SET_BEACON_TIMEOUT (0xE238)

UCHAR ANTFS_SetBeaconTimeout(UCHAR ucTimeout_)

The beacon timeout is the amount of time the Client will stay in the authentication or transport state without receiving any commands from the host before returning to the link state. The default beacon timeout is 10s.

Table 78. Set Beacon Timeout Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x38	
Timeout	UCHAR	0-MAX(UCHAR)	Timeout in seconds.

Table 79. Set Beacon Timeout Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x38	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
// Set timeout to 20 seconds.
UCHAR ucResponse = ANTFS_SetBeaconTimeout(20);

if(ucResponse == FS_NO_ERROR)
       printf(“Timeout set\n”);
```

6.6.9 MESG_FS_ANTFS_SET_PAIRING_TIMEOUT (0xE239)

UCHAR ANTFS_SetPairingTimeout(UCHAR ucTimeout_)

The pairing timeout is the amount of time the Client will wait for the host MCU to accept or reject a pairing request. The pairing request is generally accepted or rejected by the user of the client device through some type of user interface. The default pairing timeout is 30s.

Table 80. Set Pairing Timeout Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x39	
Timeout	UCHAR	0-MAX(UCHAR)	Timeout in seconds.

Table 81. Set Pairing Timeout Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x39	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
// Set timeout to 20 seconds.
UCHAR ucResponse = ANTFS_SetPairingTimeout(20);

if(ucResponse == FS_NO_ERROR)
       printf(“Timeout set\n”);
```

6.6.10 MESG_FS_ANTFS_REMOTE_FILE_CREATE_EN (0xE23A)

UCHAR ANTFS_EnableRemoteFileCreate(BOOL bEnable_)

Enables or disables the ability of the ANT-FS host to create files on the client. This functionality is disabled by default.

Table 82. Enable Remote File Create Command Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x3A	
Enable	UCHAR	0-1	Enable or disable remote file create. 0 – disable 1 - enable

Table 83. Enable Remote File Create Response Description

Parameters	Type	Range	Description
Size	UCHAR	4	Message size
Extended ID Response	UCHAR	0xE0	Extended ID Response
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being responded to.
	UCHAR	0x3A	
Response code	UCHAR	FS Response Code	Response Code: FS_NO_ERROR Otherwise command was not successful

Example Usage

```
// Enable remote creation of files.
UCHAR ucResponse = ANTFS_EnableRemoteFileCreate(1);

if(ucResponse == FS_NO_ERROR)
       printf(“Remote file creation enabled.\n”);
```

6.7 ANT-FS Reponses

6.7.1 MESG_FS_GET_CMD_PIPE (0xE23B)

UCHAR ANTFS_GetCmdPipe(UCHAR ucOffset_, UCHAR ucReadSize_, UCHAR* pucReadBuffer_)

Read a command from the command pipe. The host MCU will be notified of a new command with the MESG_FS_ANTFS_EVENT_CMD_RECIEVED event. For more details on the command pipe as well as defined commands please consult the ANT-FS Technical Specification.

Table 84. Get Command Pipe Request Description

Parameters	Type	Range	Description
Size	UCHAR	5	Message size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x3B	
Offset	UCHAR	0-32	Offset into command being read
Read Size	UCHAR	0-32	Number of bytes to read.

Table 85. Get Command Pipe Output Description

Parameters	Type	Range	Description
Size	UCHAR	2+X	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x3B	
Read Size	UCHAR	0-32	Number of bytes read
Read Payload	UCHAR[X]	Array(UCHAR)	Read payload. Max size of buffer is 32.

Example Usage

N/A

6.7.2 MESG_FS_ANTFS_SET_CMD_PIPE (0xE23C)

UCHAR ANTFS_SetCmdPipe(UCHAR ucOffset_, UCHAR ucWriteSize_, const UCHAR* pucWriteBuffer_)

Send commands over the command pipe.

Table 86. Set Command Pipe Request Description

Parameters	Type	Range	Description
Size	UCHAR	5+X	Message Size
Extended ID Request	UCHAR	0xE1	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x3C	
Offset	UCHAR	0-MAX(UCHAR)	Offset into command being read.
Write Size	UCHAR	0-32	Number of bytes to write.
Write Payload	UCHAR[X]	Array(UCHAR)	Command bytes to write. Max buffer size is 32.

Table 87. Set Command Pipe Output Description

Parameters	Type	Range	Description
Size	UCHAR	2	Message size
Message ID	UCHAR	0xE2	Extended message id
	UCHAR	0x3C	
Bytes written	UCHAR	0-32	Bytes successfully written.

Example Usage

N/A

6.8 ANT-FS Events

The integrated ANT-FS client will generate events to convey certain information to the host MCU. The general format of the response is described below.

Table 88. Set Command Pipe Request Description

Parameters	Type	Range	Description
Size	UCHAR	4+X	Message size
Extended ID Request	UCHAR	0xE0	Extended ID Request
	UCHAR	0x00	
Extended ID	UCHAR	0xE2	Extended ID being requested.
	UCHAR	0x30	
Event Code	UCHAR	0-MAX(UCHAR)	Event generated by ANT-FS
Payload	UCHAR[X]	Array(UCHAR)	Payload of the generated event

The event code and the payload are unique depending on the event generated. The table below describes the defined ANT-FS Client events.

Table 89. Integrated ANT-FS Events

ID	Event	Payload	Description
0x01	MESG_FS_ANTFS_EVENT _PAIR_REQUEST	Remote Friendly Name (up to 16 bytes)	This event is generated if a pairing request is received from the host.
0x02	MESG_FS_ANTFS_EVENT _DOWNLOAD_START	File Index (2 bytes)	This event is generated if the download of a file by a host has started.
0x03	MESG_FS_ANTFS_EVENT _UPLOAD_START	File Index (2 bytes)	This event is generated if the upload of a file by a host has started.
0x04	MESG_FS_ANTFS_EVENT _DOWNLOAD_COMPLETE	File Index (2 bytes)	This event is generated once the download of a file by a host has completed.
0x05	MESG_FS_ANTFS_EVENT _UPLOAD_COMPLETE	File Index (2 bytes)	This event is generated once the upload of a file by a host has completed.
0x06	MESG_FS_ANTFS_EVENT _ERASE_COMPLETE	File Index (2 bytes)	This event is generated after a successful erase request by the host.
0x07	MESG_FS_ANTFS_EVENT _LINK_STATE	ANT Channel (1 byte)	This event is generated whenever the Client state machine moves into the LINK state on the specified channel.
0x08	MESG_FS_ANTFS_EVENT _AUTH_STATE	ANT Channel (1 byte)	This event is generated whenever the Client state machine moves into the AUTH state on the specified channel
0x09	MESG_FS_ANTFS_EVENT _TRANSPORT_STATE	ANT Channel (1 byte)	This event is generated whenever the Client state machine moves into the TRANSPORT state on the specified channel
0x0A	MESG_FS_ANTFS_EVENT _CMD_RECEIVED	CMD ID (1 byte)	This event is generated whenever a command is received by the client on the command pipe. The command has not been

				processed by integrated ANT-FS.
0x0B	MESG_FS_ANTFS_EVENT_CMD_PROCESSED	CMD ID (1 byte)	File Index (2 bytes)[1]	This event is generated whenever a command is received by the client and processed by the integrated ANT-FS.

[1] File Create only

6.9 Response Codes

Table 90. General ANT-FS Response Codes

ID	Event	Description
0x00	FS_NO_ERROR	Operation successful
0x01	FS_MEMORY_UNFORMATTED_ERROR	NVM not formatted before attempting a file operation
0x02	FS_MEMORY_NO_FREE_SECTORS_ERROR	NVM has no free sectors left to save
0x03	FS_MEMORY_READ_ERROR	Error reading from NVM
0x04	FS_MEMORY_WRITE_ERROR	Error writing to NVM
0x05	FS_MEMORY_ERASE_ERROR	Error erasing NVM
0x06	FS_TOO_MANY_FILES_OPEN_ERROR	No more file handles free (8 max)
0x07	FS_FILE_INDEX_INVALID_ERROR	Invalid directory index specified
0x08	FS_FILE_INDEX_EXISTS_ERROR	Tried to create file at index that already exists in file system
0x09	FS_AUTO_INDEX_FAILED_TRY_AGAIN_ERROR	Auto index is pointing to a file index that already exists. Try calling function again.
0x0A	FS_FILE_ALREADY_OPEN_ERROR	Attempting to open file that is already opened
0x0B	FS_FILE_NOT_OPEN_ERROR	Attempting operations on file that has not been opened
0x0C	FS_DIR_CORRUPTED_ERROR	Directory has become corrupted, must format and rebuild.
0x0D	FS_INVALID_OFFSET_ERROR	Invalid file offset specified
0x0E	FS_BAD_PERMISSIONS_ERROR	File does not have permissions to execute the requested operation.
0x0F	FS_EOF_REACHED_ERROR	End of File reached
0x10	FS_INVALID_FILE_HANDLE_ERROR	File handle specified is not valid

Table 91. FS Crypto Response Codes

ID	Event	Description
0x32	FS_CRYPTO_OPEN_PERMISSION_ERROR	Attempting to open a non-crypto file with crypto handle
0x33	FS_CRYPTO_HANDLE_ALREADY_IN_USE	Crypto handle already in use.
0x34	FS_CRYPTO_USER_KEY_NOT_SPECIFIED	AES-128 key has not been specified
0x35	FS_CRYPTO_USER_KEY_ADD_ERROR	Error adding AES-128 key
0x36	FS_CRYPTO_USER_KEY_FETCH_ERROR	Unable to retrieve key from EEPROM
0x37	FS_CRYPTO_IVNONE_READ_ERROR	Error reading crypto IV header

Table 92. FS FIT Response Codes

ID	Event	Description
0x64	FS_FIT_FILE_HEADER_ERROR	FIT file has incorrect header information or < minimum header size
0x65	FS_FIT_FILE_SIZE_INTEGRITY_ERROR	FIT file size does not match calculated size from FIT header info
0x66	FS_FIT_FILE_CRC_ERROR	FIT CRC is incorrect
0x68	FS_FIT_FILE_CHECK_FILE_TYPE_ERROR	Incorrect file type