DRAFT

BSR E1.37-4 – 202x

Entertainment Technology - File Transfer Control with Remote Device Management over DMX512 Networks

CP/2017-1024r1
TG Draft Revision 4.0.030

Notice and Disclaimer

ESTA does not approve, inspect, or certify any installations, procedures, equipment or materials for compliance with codes, recommended practices or standards. Compliance with an ESTA standard or recommended practice is the sole and exclusive responsibility of the manufacturer or provider and is entirely within their control and discretion. Any markings, identification or other claims of compliance do not constitute certification or approval of any type or nature whatsoever by ESTA.

ESTA neither guaranties nor warrants the accuracy or completeness of any information published herein and disclaim liability for any personal injury, property or other damage or injury of any nature whatsoever, whether special, indirect, consequential or compensatory, directly or indirectly resulting from the publication, use of, or reliance on this document.

In issuing and distributing this document, ESTA does not either (a) undertake to render professional or other services for or on behalf of any person or entity, or (b) undertake any duty to any person or entity with respect to this document or its contents. Anyone using this document should rely on their own independent judgment or, as appropriate, seek the advice of a competent professional in determining the exercise of reasonable care in any given circumstance.

Published By:
Entertainment Services and Technology Association (ESTA)
271 Cadman Plaza PO Box 23200
Brooklyn, NY 11202-3200
USA
Phone: +1-212-244-1505
Email: standards@esta.org

ESTA's Technical Standards Program

The ESTA Technical Standards Program was created to serve the ESTA membership and the entertainment industry in technical standards related matters. The goal of the program is to take a leading role regarding technology and safety within the entertainment industry by creating recommended practices and standards, monitoring standards issues around the world on benefit of our members, and improving communications and safety within the industry. ESTA works closely with the technical standards efforts of other organizations within our industry including ESA, CITT, USITT and VPLT as well as representing the interests of ESTA members to ANSI, UL, ASCE, ICC, and the NFPA. ESTA is an ANSI Accredited Standards Developer.

The Technical Standards Council (TSC) established by ESTA's Board of Directors to oversee and coordinate the Technical Standards Program. Made up of individuals experienced in standards development work from throughout our industry, the Committee approves all projects undertaken and assigns them to the appropriate working group. The Technical Standards Council employs a Technical Standards Manager to coordinate the work of the Committee and its working groups as well as maintaining a "Standards Watch" on behalf of members. Working groups include: Control Protocols, Electrical Power, Event Safety, Floors, Fog and Smoke, Followspot Positions, Photometrics, Rigging, Stage Machinery, and Prop Weapons Safety.

ESTA encourages active participation in the Technical Standards Program. There are several ways to become involved. The easiest way to actively participate is to respond to any of the public reviews advertised on ESTA's public review web page. The next level of participation requires completion of an application to become a working group member; applications are available from the TSP's procedural documents web page. Application as an Observer member affords access to updates on standards development documents. Application as a voting member affords full participation as a consensus voice that helps shape the industry. Requirements for voting membership include responding to letter ballots and attending meetings, but membership in ESTA or any other organization is not a requirement for participation in the TSP. One can also become involved by requesting that the TSC develop a standard or a recommended practice in an area of concern to them.

The Control Protocols Working Group, which authored this standard, consists of a cross section of entertainment industry professionals representing a diversity of interests. ESTA is committed to developing consensus-based standards and recommended practices in an open setting.

Investors in Innovation

The Technical Standard Program (TSP) is financially supported by ESTA and by companies and individuals who make donations to the TSP. Contributing companies and individuals who have helped fund the TSP are recognized as “Investors in Innovation”. These are the Investors in Innovation when this standard was approved by ANSI's Board of Standards Review:

[Insert the current Investors table here. Source from latest Swatch Edition]

Memorial donor: The Estate of Ken Vannice

All donations to the Technical Standards Program benefit the entire program, and are not directed to any specific use or project within the program. Please help support the Technical Standards Program by becoming an Investor in Innovation. Visit our website at http://tsp.esta.org/invest, or contact the ESTA office at 1-212-244-1505 and select "TSP" from the menu.

Contact Information

Technical Standards Manager
Richard J. Nix
ESTA
271 Cadman Plaza PO Box 23200
New York, NY 11202-3200
USA
+1-212-244-1505
richard.nix@esta.org

Senior Technical Standards Manager
Karl G. Ruling
ESTA
271 Cadman Plaza PO Box 23200
New York, NY 11202-3200
USA
+1-212-244-1505
karl.ruling@esta.org

Technical Standards Council Chairperson
Mike Garl
Mike Garl Consulting LLC
+1-865-389-4371
mike@mikegarlconsulting.com

Alan Rowe
I.A.T.S.E Local 728
Phone: 1 310-702-2909
amrowe@iatse728.org

Control Protocols Working Group Co-chairpersons

Milton Davis
Doug Fleenor Design, Inc.
+1-805-481-9599
milton@dfd.com

Michael Lay
Candela Controls, Inc
+1-352-433-2479
michael_m_lay@yahoo.com

Acknowledgments

The Control Protocols Working Group members, when this document was approved by the working group on [insert WG approval date here], are shown below.

Voting members:

Observer (non-voting) members:

Interest category codes:

CP = custom-market producer
DE = designer
DR = dealer rental company
G = general interest
MP = mass-market producer
U = user

Table of Contents

List of Tables

1 Introduction (Informative)

1.1 Approach taken in this standard

While manufacturers have been able to implement their own proprietary means of file transfer these are limited to supporting transfers from a single manufacturer's controllers or testing tools to the same manufacturer's devices; this frequently required disconnecting the devices from the existing [DMX] infrastructure. There is no standardized method to support file transfer between a controller from Manufacturer-A and a device from Manufacturer-B.

This document provides developers of RDM enabled products with a standard method for transfer of files between RDM capable products using the existing basic communication structure provided by [RDM] and its related standards including ANSI E1.33 [RDMnet].

The design approach is intended to facilitate data transfers to responders that have very limited memory resources as well as devices that can support the largest possible [RDM] packet all while maintaining the physical [DMX] connections to multiple devices from multiple manufacturers.

The process described in this document is referred to as File Transfer Control (FTC) which encompasses file uploads and downloads.

2 Scope

This document sets out the minimum requirements for [RDM] Controllers to successfully support File Transfer Control (FTC) between the Controller and a Responder.

This standard specifically provides a transfer mechanism that can be applied to memory limited responders with transmit and receive buffers as small as 64 bytes, while allowing scalable improvements (a net reduction) in transfer times for responders that have larger buffers.

The extent of responder functionality during the FTC process is not defined by this standard, and is outside the scope of this document. This standard does include a method for the responder to declare whether or not functionality will be limited during FTC

2.1 Source Media

The choice of media support for source data is outside the scope of this standard. Equipment manufacturers are free to choose whatever means they feel is appropriate to their product. Examples of common source data media include USB key, SD Card, Email or Internet download.

2.2 File Structure

A file for transfer should contain data in binary format. A controller should have no interaction with the content of a file. The controller will calculate a FileCRC over the contents as outlined in section of this document.

This standard places no constraints on the naming of any source file or the internal structure of the file.

Transfer and subsequent installation of firmware files involves inherent risk to the operation of a device. It is recommended that a method to determine the integrity of the entire transferred firmware file is contained in the firmware file by the files author. The manner in which this is determined and implemented is the responsibility of the target equipment manufacturer, and beyond the scope of this standard. Example integrity checking methods include Checksum and CRC.

3 Definitions

Any definitions given in this standard are particular to this standard and are either not found in a standard dictionary or are used in this standard with a meaning different from what might be found in a standard dictionary.

Upload: An upload refers to the Controller initiated transfer of data from a Controller to a Responder.

Download: A download refers to the Controller initiated transfer of data from a Responder to a Controller.

Controller/Responder: Unless otherwise specified in this standard, the characteristics of an RDM Controller and RDM Responder are as described in [RDM] Appendix D.

One-to-One: A “One-to-One” transfer is a transfer between a Controller and a single Responder. Each transferred packet is acknowledged.

One-to-Many: A “One-to-Many” transfer is a transfer between a Controller and more than one Responders, frequently using [RDM] Broadcast messages. One-to-Many transfers are outside the scope of this document.

Cyclic Redundancy Check (CRC): A **cyclic redundancy check** (**CRC**) is an error-detecting code commonly used in digital networks and storage devices to detect accidental changes to digital data. Blocks of data entering these systems get a short *check value* attached, based on the remainder of a polynomial division of their contents. On retrieval, the calculation is repeated and, in the event the check values do not match, corrective action can be taken against data corruption.

4 Normative References

[DMX] ANSI E1.11-2008 (R2018)
Entertainment Technology -- USITT DMX512-A --
Asynchronous Serial Digital Data Transmission Standard for Controlling Lighting Equipment and Accessories.

Entertainment Services and Technology Association (ESTA)
P.O. Box 23200
Brooklyn, NY 11202-3200
USA
Phone: 1-212-244-1505
Email: standards@esta.org
http://tsp.esta.org

[RDM] E1.20 - xxxx, Entertainment Technology-RDM-Remote Device Management over USITT DMX512 Networks

This standard is maintained by ESTA.

[RDMnet] ANSI E1.33 Entertainment Technology - (RDMnet) Message Transport and Management for ANSI E1.20 (RDM) compatible and similar devices over IP Networks

This standard is maintained by ESTA.

[PIDS-1] ANSI E1.37-1-2011 (R2022)
Entertainment Technology - Additional Message Sets for ANSI E1.20 (RDM) – Part 1, Get/Set Parameter Messages

This standard is maintained by ESTA.

[UTF-8] The Unicode Consortium. The Unicode Standard, Version 13.0.0, (Mountain View, CA: The Unicode Consortium, 2020. ISBN 978-1-936213-26-9)
http://www.unicode.org/versions/Unicode13.0.0/
[EUI] Guidelines for use of a 48-bit Extended Unique Identifier (EUI-48)
http://standards.ieee.org/

This document is maintained by: IEEE Operations Center, 445 Hoes Lane, Piscataway, NJ 08854-4141, USA

[Modbus] Modicon Modbus Protocol Reference Guide
Modbus Organization, Inc.
PI-MBUS-300 Rev J., June 1996

MODICON, Inc., Industrial Automation Systems
One High Street North Andover, Massachusetts 01845

http://modbus.org/docs/PI_MBUS_300

5 Overview (Informative)

This section summarizes FTC operation. It is provided so that readers of the standard can determine suitability of this standard to their requirements.

A FTC Upload (File upload from controller to responder) transfer comprises three stages: Initiate, Transfer, and Commit.

Upload	***File Transfer Sequence***
Initiate	"Get Initiate": A Controller obtains details about required transfer configuration information from Responder(s) [optional]
	"Set Initiate": A Controller initiates a FTC Upload
Packet 1 Transfer	1st packet of data is sent based on transfer requirements obtained during Initiate or already known
Packet 1 Verify	Responder verifies integrity of the 1st packet using PacketCRC
Transfer/Verify	Additional packets are transferred and verified
Packet *n* Transfer	*n*th packet of data is sent where *n* is the total number of packets required to complete the transfer based on the file size and the transfer requirements obtained during Initiate (or already known)
Packet *n* Verify	Responder verifies integrity of *n*th packet using PacketCRC and validates the complete file using FileCRC
Commit	Get Commit : Controller requests Responder validation status Set Commit: Controller instructs the Responder to commit (save) the file
Responder integrity check	Optional: Responder verifies that the file is of an expected type and meets manufacturers requirements and integrity check

This standard introduces additional Parameter Messages ([RDM], Section 10).

Four mandated Parameter Messages are used to control FTC Upload and are the only Messages required to support FTC Upload. Additional optional Parameter Messages can be used to support FTC Downloads and multi-file capabilities.

5.1 File Compatibility

It is recommended that the information required to determine that the data is suitable for the target responder is contained in the first few file packets transferred. This might include information such as make, model and version or other details as chosen by the responder manufacturer.

The manner in which this is determined is the responsibility of the Responder manufacturer, and beyond the scope of this standard.

Manufacturers who elect to encode this information into the first 16 bytes of their files could allow a responder [from said Manufacturer] to reject the initiation of the transfer should the proprietary encoding not be satisfied.

This standard does not specify the format of any file generated by a manufacturer for a particular responder.

5.2 Data Integrity

A Cyclic Redundancy Check (CRC) is calculated by the Controller and offered to the Responder for each transfer packet and for the entire file.

Manufacturers may also wish to embed segment numbers, checksum or CRC information in the content that is transferred over the data link using this standard. The manner in which such checks are embedded in manufacturer's files is beyond the scope of this standard.

5.3 Responder Design Choices

Responder hardware will typically have some design choices that determine how much data may be accepted before it can be committed to storage. Microcontrollers or other off-chip memory devices may impose a "page" size, at which point a finite time (longer than the normal inter-packet time) is required to commit data to non-volatile storage. During this time the responder may not be able to accept RDM messages, or perform normal operations.

Responders may also have memory limitations that limit the size of each RDM packet that can be received.

Such characteristics can be declared so that a Controller can modify it's behavior appropriately.

5.4 Use of Proxy Devices

Consideration has been given to the use of Proxy Devices ([RDM],Section 8).

Methods of determining the existence of Proxy Devices are described in [RDM] Section 8.2 and [RDM] Section 8.4.

This is intended to provide controllers and responders with the ability to use this standard in conjunction with products such as wireless DMX512/RDM data links.

Controllers issuing Upload commands using this protocol should expect Proxy Devices to use the ACK_TIMER method described in the [RDM] standard when it is not possible to provide an immediate response to a controller.

5.5 Cancellation of Transfer

A Controller may, at any stage during a transfer, issue a command to cancel the transfer. Receipt of such a command by a Responder allows the responder to abandon all further expectation of data transfer until a new session is initiated by a controller.

A Responder may indicate the presence of an error condition to the Controller through the ResponseStatus field. Error conditions may indicate that the Controller needs to Cancel this Transfer Session.

5.6 Transfer Times

An example calculation of transfer times is provided in Appendix C.

5.7 Limitations

It should be recognized that:

- This standard does not support Sub-devices on responders. All FTC transactions must interact with the Responder's Root Sub-device (0x0000)
- Firmware upload is by its nature a maintenance operation and should not normally be undertaken during show performance situations.
- Responders may not be able to provide normal functionality while processing FTC requests and are not required by this standard to do so. Such limitations shall be declared by the Responder Capabilities declaration.

- Controllers may cease or delay generation of NULL start code or other ASC packets during FTC processing. This standard does not support the simultaneous transfer of multiple files to an individual responder.
- Some Responders cannot maintain important settings, calibration tables, etc. during a firmware upgrade

Note: *The use of the Download PID, Upload PID, Filelist PID, and FileID (as described in this standard) may be used to support save and restore of such important settings, calibration tables, etc.*

6 Message Structure

All FTC messages shall use the standard RDM message structure as defined in [RDM].

The standard NACK responses for Format Error or Data Out of Range, as defined in [RDM], shall be used when the message construct does not comply with this document, unless specifically stated in this document.

All messages shall return a response type [RDM] ACK with additional information as described in this document unless the message has a parametric RDM error. An example would be if the SET command is issued for a PID that does not support the SET Command Class with a NACK and a Nack Reason of Unsupported Command Class.

6.1 Text Fields

Text fields shall conform to [RDM] Section 10.1.

7 Controller Requirements

7.1 [RDM] Support Requirements

Controllers implementing this standard shall accept the maximum size RDM packet as defined in [RDM].

7.2 Functional Requirements

Controllers wishing to transfer data using this standard shall determine the required TransferBlock Size, Initial Delay, Inter-Packet Delay time, Accumulated Byte Count, associated Accumulated Byte Count Delay Time, Validation Delay Time and Communications Offline Delay Time appropriate to a Responder, in accordance with the methods described in this document.

The size of each transfer packet shall be determined by the controller from information obtained from the Responder.

Although the method of determining the existence of Proxy Devices is beyond the scope of this standard, Controllers that have determined the existence of Proxy Devices may also determine any required TransferBlock Size, Initial Delay, Inter-Packet Delay, Accumulated Byte Count, associated Accumulated Byte Count Delay and Commit Delay constraints appropriate to the Proxy Device, in accordance with the methods described in [RDM]

The Controller shall use these declared sizes and delays to regulate the transfer of data from Controller to Responder.

A Responder may indicate at any point in the transfer that the data offered is not suitable for the responder. Receipt of such an error code through the ResponseStatus field by a controller shall cause the Controller to abandon further data transfer in the current session.

8 Responder Requirements

There are some minimum requirements that should be facilitated by Responders to reliably use the transfer scheme detailed in this document. Responder manufacturers shall observe the following points to ensure the integrity of their products during any FTC process.

8.1 Design

Responders should use a design whereby a transfer can be resumed at any stage of the FTC process in the case of any error condition created by the Controller, or loss of data, or loss of power at the Responder.

Responders supporting firmware uploads using FTC may provision for a Bootloader that supports this standard. Such a bootloader shall either be capable of Discovery in accordance with [RDM] so that the transfer process may be repeated after disruption of the data transfer process or include a recovery mechanism incorporating a timeout of any current FTC transfer.

Manufacturers who offer some alternative fatal error recovery shall declare this in accordance with Section .

Controllers may use this declaration to inform the user of the risks associated with the proposed transfer.

8.2 Responder Firmware

Where possible, Responders should not commit any updated firmware before data transfer has completed and data validation has been confirmed. If a Responder has to use partial commit due to hardware constraints then the Responder shall support the Fail_May_Brick Responder Capabilities field as referenced in Section of this standard

8.3 Responder Limited Behavior

A Responder may limit or stop its normal functionality during the FTC process.

Responders shall declare if they allow [DMX] refresh or limit normal functions using the Capabilities field as described in section of this standard.

> *For example, a responder may allow [DMX] (Null Start Code) packet processing while transferring a firmware file, but may not respond to non-FTC [RDM] messages.*

Responders that cannot process transfers using [RDM] Broadcast messages shall declare this as outlined in Section of this standard.

8.4 Responder Declarations

Typically, Responder hardware will have some design constraint(s) that determine how much data may be accepted before it can be committed to storage. Microcontrollers or other off-chip memory devices may impose a “page” size, at which point a finite time (and possibly longer than the normal inter-packet time) is required to commit the data to non-volatile storage. In this standard these requirements are declared by use of the Accumulated Byte Count and Accumulated Byte Count Delay.

During any delay time the responder may not be able to accept DMX Packets, accept RDM messages, or perform normal operations.

Responders may also have memory limitations that limit the size of each RDM packet that can be received.

The Responder characteristics and requirements are obtained using the GET:FTC_INITIATE Parameter Message as described in Section.

8.4.1 TransferBlock Size (8-bit)

The responder shall declare the number of bytes of file data to be sent in each packet.

The valid range is 0x01-0xE0 (1-224 bytes). A responder shall not request 0 bytes of file data.

This value shall be used by a Controller to determine the number of bytes of file data to send to or receive from the Responder in each FTC_TRANSFER_UPLOAD or FTC_TRANSFER_DOWNLOAD message packet.

The 224 byte limitation is derived from (i) the 231 byte maximum PDL payload as defined in [RDM], and (ii) the FTC header and PacketCRC within each transfer packet.

It is understood that the last TransferBlock sent may be less than the TransferBlock Size.

8.4.2 Initial Delay Time (32-bit)

Minimum time (in ms) that the responder requires the controller to wait after Initiation before the first FTC_TRANSFER_UPLOAD packet is sent.

The valid range is 0-4,194,304ms (apprx. 1.2hrs), where 0 is 176 µs, rounded to nearest ms.

This value shall be used by a controller to determine when to send the first FTC_TRANSFER_UPLOAD packet.

Responder designers are encouraged to keep this Initial Delay time as low as possible, and preferably zero. In any event, successive packets cannot be sent with less than 176 µs inter-packet time as stated in the [RDM] standard.

8.4.3 Inter-packet Delay Time (16-bit)

Minimum time (in ms) that the responder requires the controller to wait after each FTC_TRANSFER_UPLOAD packet before the next FTC_TRANSFER_UPLOAD packet can be sent.

The valid range is 0-65,535ms (apprx 1 minute), where 0 is 176 µs, rounded to nearest ms.

This value shall be used by a controller to determine when to send each FTC_TRANSFER_UPLOAD packet to a responder, and when to send the Get:Commit request following the successful transfer of the last packet of the file.

Responder designers are encouraged to keep this Inter-Packet Delay time as low as possible, and preferably zero. In any event, successive packets cannot be sent with less than 176us inter-packet time as stated in the [RDM] standard.

8.4.4 Accumulated Byte Count (32-bit)

The Accumulated Byte Count shall be used by a controller to determine when to further delay packet transfers after “N” bytes have been transferred to the responder. This field determines when to make use of the Accumulated Byte Delay Time.

The valid range is 0x00000000-0xFFFFFFFF

A responder that does not require any intermediate delays shall set this field to 0x00000000.

8.4.5 Accumulated Byte Delay Time (16-bit)

The valid range is 0-65,535ms (approx. 1 min.).

A responder that does not require any intermediate delays shall set this field to 0x0000.

A controller shall use this field to determine how long to wait (in ms) once the number of transmitted bytes reaches the declared Accumulated Byte Count.

This is to allow time for the responder(s) to save the accumulated data to non-volatile storage or perform other background tasks as may be required by their hardware designs.

Controllers shall reset their internal Byte counter every time the delay is applied.

This scheme allows for responders with limited buffer sizes to accept data as efficiently as possible prior to having the controller wait while the responder [potentially] goes off line to validate and write to local non-volatile storage.

8.4.6 Validation Delay Time (16-bit)

The validation delay time with a valid range of 0-65,535ms (approx. 1 min.).

This time shall be declared by a responder to indicate how long a controller should wait (in ms) after sending the last TransferBlock.

This time should be used by a controller to determine how long to wait (in ms) between when the Get:Commit request has been sent and acknowledged and before the Responder will be ready for a Set:Commit request.

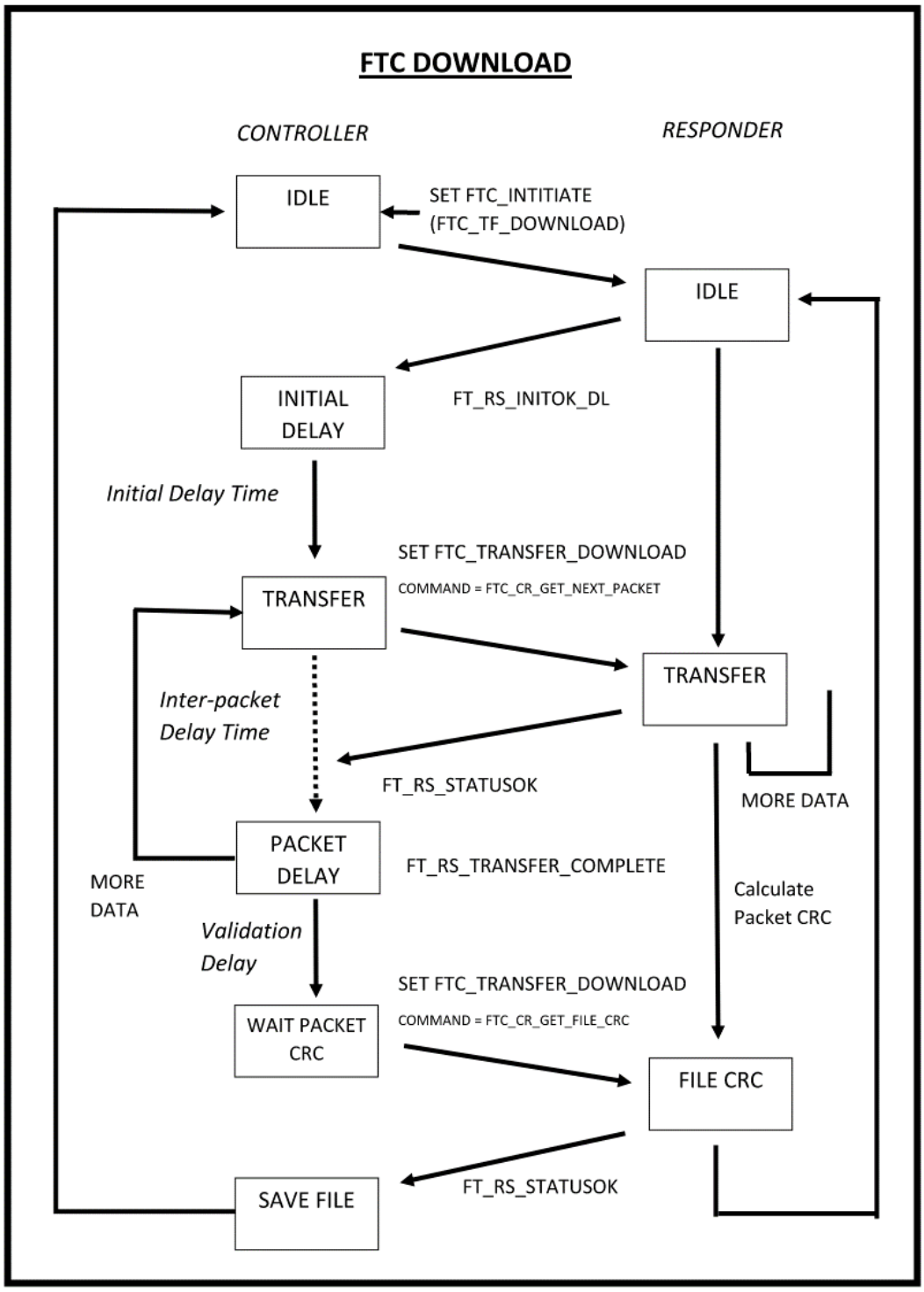
8.4.7 Responder Capabilities (32-bit)

A number of responder characteristics/capabilities (i.e., accepts Uploads/Downloads, processes/generates CRCs, supports Testmode, etc) are reported using a 32-bit capabilities field, in Big-Endian order. These are defined in Table A-7.

All unused and/or undefined bits shall be set to (0), Active bits shall be set to (1) unless otherwise specified.

9 Operational Overview

Controller and Responder support for the Parameter Message is required as noted in Table A-2.



9.1 Initiate Transfer

To obtain details about a possible file transfer, the controller may issue a GET: FTC_INITIATE Parameter Message in accordance with Section . This message allows collection of required transfer configuration information without preparing a responder for a new data transfer.

To initiate a file transfer in either direction (Upload or Download), the controller shall issue a SET: FTC_INITIATE Parameter Message as described in Section.

9.2 Transfer File Data

Files shall be uploaded to the responder by the controller using an [RDM] packet size in accordance with characteristics obtained from the responder.

The amount of file data that can be transferred in each [RDM] packet shall be determined from the responder during the Initiate stage. This amount is referred to as the TransferBlock Size. This TransferBlock Size may be different than any hardware “page” size constraints imposed by the responder. The maximum allowed TransferBlock Size is restricted by the maximum Parameter Data Length imposed by the [RDM] Standard.

***Note**: To send a “page” of data, it may be required to transmit one or more RDM packets.*

Failure or interruption of the data transfer process, or corruption of data shall not in any way affect the subsequent operational integrity of the responder.

Transfer of file data shall use the FTC_TRANSFER_UPLOAD Parameter Message in accordance with Section or the FTC_TRANSFER_DOWNLOAD Parameter Message in accordance with Section.

9.3 Commit File Data

The FTC_COMMIT Parameter Message shall be used to initiate reprogramming of hardware once the file transfer has been completed and validated. Controllers shall only issue an instruction to commit after confirmed receipt of the final TransferBlock. The commit stage may initiate responder reprogramming of code space as necessary.

When the file being uploaded is a firmware file, the instruction to commit is the only instruction that initiates the responder to validate the firmware, save the firmware, and reload. Responders with limited memory may save the firmware during the file transfer phase. Responders shall not self initiate a reload and reboot without the commit instruction. Responders may reboot during the Commit phase, between the firmware save and the reload, and after the reload.

The FTC_COMMIT Parameter Message is described in Section.

9.4 Resuming Communications

Controllers should not expect to resume communication with a responder that is committing data until after the Commit Delay time has expired.

9.5 Cancellation of Transfer

The FTC_CANCEL Parameter Message shall be used when the controller wishes to abort any transfer in accordance with Section . FTC_CANCEL may be sent after sending SET:FTC_COMMIT but may be ignored by the Responder.

A controller should not rely on any data being held by the responder after any cancellation of the transfer.

10 Cyclic redundancy Check (CRC)

10.1 CRC Description

A cyclic redundancy check (CRC) is an error detecting code used to confirm the integrity of digital data by detecting accidental changes. The block of digital data is provided a small check value based on the remainder of a polynomial division of the block contents. The calculation is repeated during the check process (such as after receiving a transmitted block of data). If the values do not match then Devices can assume the data integrity has been compromised and take action accordingly. The CRC used in this standard is a 16-bit CRC, specifically CRC-16-ANSI (also known as CRC-16-IBM or Modbus CRC16). Additional information on CRC use and generation can be found in Appendix D or [Modbus] Appendix C.

10.2 Use of CRC

Where mention is made of FileCRC or PacketCRC in this standard, the following implementation shall be followed.

The CRC shall be the 16-bit remainder after division modulo 2 of the message polynomial by the generator polynomial $x^{16}+x^{15}+x^{2}+1$, calculated according to the algorithm described in Appendix D.

The calculated CRC shall be transmitted in Big-Endian Order in accordance with [RDM] Section 6.1.

10.3 FileCRC

Controllers shall provide a FileCRC as part of the SET:FTC_INITIATE Parameter Message for uploads.

Responders may provide FileCRC as part of the GET:FTC_TRANSFER_DOWNLOAD Parameter Message for downloads.

FileCRC is computed for the entire file contents.

10.4 PacketCRC

Controllers shall provide a PacketCRC as part of the FTC_TRANSFER_UPLOAD Parameter Message for uploads.

Responders may provide a PacketCRC as part of the FTC_TRANSFER_DOWNLOAD Parameter Message for downloads.

The PacketCRC shall be calculated from the first byte of the Parameter Data for a length of Parameter Data Length minus 2.

For example, if the PDL is 16, CRC will be calculated over the first 14 bytes of the Parameter Data.

This will include the FTC header and the File Data, but not the PacketCRC field itself.

11 SessionID, FileID, and FTC Version

11.1 Use of SessionID

The SessionID is a 8-Bit field used to uniquely identify a file transfer session between a controller and a responder. It is created by the controller and offered to the responder as part of the SET:Initiate stage. It is then used in the Transfer, Commit, and Cancel stages.

The SessionID shall be in the range 0x01-0xFE and controllers maintain the SessionID until it is committed or abandoned. Steps should be taken to randomize the SessionID after controller startup to reduce repeated use of the same SessionID.

SessionIDs allow that:

- A responder that has been disconnected during a file transfer, and subsequently reconnected during a different transfer has a means of rejecting the data as the new transfer should have a different SessionID.
- A responder that has been relocated onto another physical data link that is in the process of transferring file data can decide to resume or reject the transfer.

The SessionID FTC_DEF_SESSIONID_ALL is reserved for use when a controller wishes to cancel all current transfers.

Responders shall verify that the SessionID for the Transfer and Commit messages match the SessionID supplied in the SET:FTC_INITIATE message.

11.2 Use of FileID

The FileID field is a unique defined value that identifies the file to be transferred.

The value FTC_DEF_NO_FILEID_OFFERED is sent to the Responder from the Controller in the Get: FTC_INITIATE to query the files supported by the Responder.

If the Responder only supports a single file, then the Responder returns a value in the range as listed in Table A- and this is the FileID that should be sent in the SET: FTC_INITIATE message

If the Responder supports more than 1 file then the responder returns the ResponseStatus of FTC_RS_UNSUPPORTED_FILEID and FileID value of FTC_DEF_MULTIPLE_FILEID. The Controller should send the Get: FTC_FILELIST message to determine what files and file types are supported. Files may be for upload, download, or both as determined from the Capabilities fields returned with the Get: FTC_FILELIST message.

The value FTC_DEF_NO_FILEID_OFFERED may also be used with SET: FTC_INITIATE in place of using GET:FTC_INITIATE. If the responder only supports a single file then the FTC process will continue.

The values of FTC_DEF_NO_FILEID_OFFERED and FTC_DEF_MULTIPLE_FILEID are reserved and have specific meaning. The Responder can assign values as listed in Table A-to represent the files that are supported. They do not need to be consecutive and are not required to be the same for different Responder models.

The Controller should not assume that any given FileID represents a given file type. The Get: FTC_FILELIST Capabilities field shall be used to determine the file type specific details.

11.3 Use of FTCVersion

FTCVersion is a 16-bit field used to indicate the major and minor version of the FTC Standard being used by the Controller or Responder. Responders should reply with FTC_RS_FTCVERSION_ERROR if a GET:INTITIATE or SET:INTIIATE is sent with an unsupported version.

Valid versions are:

Version (Major.Minor)	Version Description

0.01	Initial draft version, 2024-02-15
0.02	Draft version, 2024-04-11
0.03	Draft version, 2024-06-20
0.xx	Future draft versions
xx.xx	Future ratified versions

Table 11-1: FTCVersion

This version of the FTC standard shall be referenced as **FTCVersion 0.03**.

12 Replies from Responders

All replies from a responder to the FTC_INITIATE, FTC_TRANSFER_UPLOAD, FTC_TRANSFER_DOWNLOAD, FTC_COMMIT and FTC_CANCEL Parameter Messages include the ResponseStatus and ResponseData fields.

12.1 ResponseStatus and ResponseData Fields

12.1.1 Good response (FTC_RS_STATUSOK)

The ResponseStatus of FTC_RS_STATUSOK is used to indicate successful execution of a request.

The ResponseData field may contain information regarding the successful packet transfer.

12.1.2 Good response – more time required (FTC_RS_STATUS_IN_PROGRESS)

The ResponseStatus FTC_RS_STATUS_IN_PROGRESS informs the controller that the responder requires further time to action the request.

The delay time (in ms) is returned in the ResponseData field.

The delay time returned in this field becomes the InterPacketDelay time for the next request.

Controllers may choose to use this time to dynamically adjust the inter-packet timing, but are not obligated to do so.

In the absence of any delay time reported via the FTC_RS_STATUS_IN_PROGRESS response, a controller shall use the default inter-packet timing provided during the Initiate phase.

> *Although this has functionality similar to the use of ACK_TIMER as described in [RDM], the ACK_TIMER granularity of 100ms is considered too coarse for timely upload of data that may require less than 10ms for the data to be saved by a responder in its internal non-volatile storage.*

Use of FTC_RS_STATUS_IN_PROGRESS is not equivalent to the ACK_TIMER in so far as no use of QUEUED_MESSAGES is implied or required.

13 Parameter Messages Use and Definition

Responders implementing Parameter Messages shown as GET only in Table A-2: Parameter ID Defines shall treat any SET_COMMAND as an error and report NRC_UNSUPPORTED_COMMAND_CLASS in accordance with [RDM].

Responders implementing Parameter Messages shown as SET only in Table A-2: Parameter ID Defines shall treat any GET_COMMAND as an error and report NRC_UNSUPPORTED_COMMAND_CLASS in accordance with [RDM].

13.1 Initiate Transfer (FTC_INITIATE)

13.1.1 Initiate Transfer (GET:FTC_INITIATE)

A Controller may use the GET:FTC_INITIATE Parameter Message to ascertain the capabilities of a responder prior to initiating a transfer using SET:FTC_INITIATE Parameter Message.

This command does not tell the responder anything about a pending data transfer.

Controller: (GET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) FTC_INITIATE	(PDL) 0x06

(PD)

SessionID (8-bit)
FileID (8-bit)
FTCVersion (16-bit)
TransferFlags (16-bit)

Data Description:

Session ID (8-bit):
This shall be set to FTC_DEF_NO_SESSIONID_OFFERED. *See* Table A-.

FileID (8-bit):
This shall be set to FTC_DEF_NO_FILEID_OFFERED or a known valid FileID. *See* Table A-.

FTCVersion (16-bit):
This field is used to indicate the major and minor version of the FTC Standard being used by the Controller. See Section .

TransferFlags (16-bit):
This field is used to indicate defined transfer information. See Table A-5.

13.1.2 Initiate Transfer (SET:FTC_INITIATE)

When uploading, the SET: FTC_INITIATE Parameter Message provides information about the size of file being offered and the response returns details to be used by the controller to manage the upload process.

A controller shall issue a SET: FTC_INITIATE Parameter Message to inform the responder that it is about to commence file transfer using one or more FTC_TRANSFER_UPLOAD or FTC_TRANSFER_DOWNLOAD Parameter Messages.

Responders implementing this standard shall reply to the FTC_INITIATE Parameter Message without use of the [RDM] ACK_TIMER mechanism.

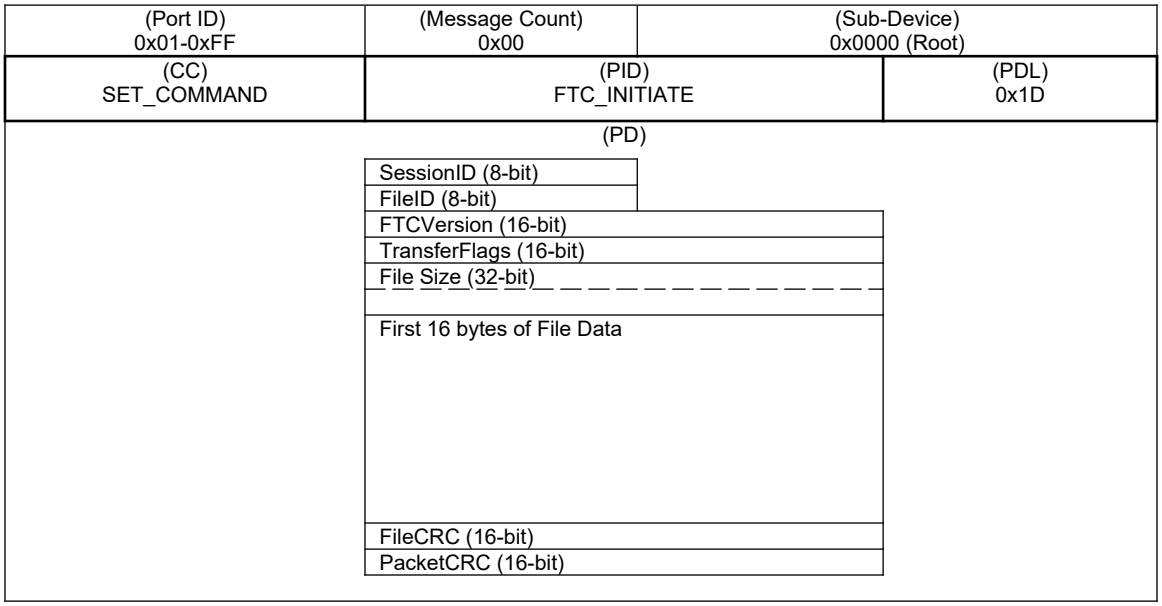
A responder shall ACK the SET: FTC_INITIATE Parameter Message provided there are no RDM packet format errors in the request. In the event of such low levels errors the existing [RDM] NACK response shall be used.

The responder shall treat out of range values in the SET request differently to other [RDM] responses. It is not appropriate to use the [RDM] NACK : DATA_OUT_OF_RANGE response when replying to the SET: FTC_INITIATE.

In the event that the responder detects invalid or out of range parameter data values the responder shall still ACK the request and return the error using the ResponseStatus and ResponseData fields as described in this standard.

Responders that are rejecting a transfer negotiate request for more than one issue shall only reply with a single Response Code as listed in Table A-. Response statuses are not prioritized. The single Response Code is at the discretion of the responder.

Controller: (SET) *Controller request to initiate a file transfer.*



Data Description:

SessionID (8-bit) :
The SessionID shall be created and issued by a controller at the FTC_INITIATE stage. The session ID shall be used by the controller and this responder until the FTC Transfer is cancelled or otherwise aborted. SessionID is as defined in Section of this standard.

FileID (8-bit) :
This shall be set to FTC_DEF_NO_FILEID_OFFERED or a known valid FileID*. See* Table A-

FTCVersion (16-bit):
This field is used to indicate the major and minor version of the FTC Standard being used by the Controller. See Section .

TransferFlags (16-bit):
This field is used to indicate defined transfer information. See table Table A-5.

File Size (32-bit)
For a controller initiating an upload these fields shall be set to total file size in bytes.

File Data (16 bytes)
For a controller initiating an upload this field shall be set to the first 16 bytes of the file.

In the event that the file data is less than 16 bytes, controllers shall set the remaining bytes as 0x00.

A Responder may use this data to validate that the file to be transferred is in fact appropriate to the responder.

The 16 byte limit in the negotiate stage is to satisfy the requirements of memory limited responders. The overall size of the FTC_INITIATE Parameter Message (PDL plus RDM header) is designed to be less than 64bytes. This allows for responders which can only support small RDM packets to use this standard.

Where manufactures require more than 16 bytes to validate the file is appropriate, there is nothing in this standard that prevents manufacturers of responders rejecting the data transfer after actual transfer has commenced.

The data does not need to be stored by the responder at this point, and shall be sent again by the controller as part of the first packet transfer using the FTC_TRANSFER_UPLOAD Parameter Message.

Responders that do not verify suitability of the data using this method are still free to reject data transfers at any time during the upload process.

FileCRC (16-bit) :
For a controller initiating an upload this field shall be set to the FileCRC of the file being offered, in accordance with the method referenced in section of this standard

PacketCRC (16-bit) :
For a controller initiating an upload this field shall be set to the PacketCRC of the parameter data being transferred, in accordance with the method referenced in section of this standard

13.1.3 Responder : Reply to GET:INITIATE or SET:INITIATE (Transfer Flag = Upload)

Responder : (GET/SET Response) – FTC_TRANSFER_UPLOAD

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) GET/SET_COMMAND_RESPONSE	(PID) FTC_INITIATE	(PDL) 0x22

(PD)

ResponseStatus (8-bit)
ResponseData (32-bit)
SessionID (8-bit)
FileID (8-bit)
FTCVersion (16-bit)
File Size (32-bit)
Capabilities Bit Field (32-bit)
DataOffset (32-bit)
TransferBlock Size (8-bit)
Initial Delay Time (32-bit)
Inter-packet Delay Time (16-bit)
Accumulated Byte Count (32-bit)
Accumulated Byte Delay Time (16-bit)
Validation Delay Time(16-bit)
Delay After Broadcast (16-bit)

Data Description:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit)

This shall be set by the responder in accordance with Section of this standard.

SessionID (8-bit)
Responders that are not in a current FTC transfer process shall report FTC_DEF_NO_SESSIONID_OFFERED.

Responders that have begun a FTC Transfer process shall report the SessionID offered by the most recent successful SET:FTC_INITIATE.

FileID (8-bit)
The Responder should set the FileID value as defined in Table A- to indicate the file supported or set to FTC_DEF_MULTIPLE_FILEID to indicate that multiple files are supported.

Responders that have begun a FTC Transfer process shall report the FileID offered by the most recent successful SET:FTC_INITIATE.

Note: *If FileID is set to FTC_DEF_MULTIPLE_FILEID then the Controller should send a Get: FTC_FILELIST to determine the appropriate files, file types, and capabilities (by FileID).*

FTCVersion (16-bit)
This field is used to indicate the major and minor version of the FTC Standard being used by the Responder. See Section .

File Size (32-bit)
These fields shall be set to total file size in bytes. If the file size is not known the responder shall respond with 0x00000000.

Capabilities Bit Field (32-bit)
This shall be set by the responder in accordance with Section of this standard.

A controller should utilize the declared capabilities to optimize the operation of the transfer process, or inform the user of any limitations.

DataOffset (32-bit):
The responder shall return the current file offset.

This should equate to the last offset sent by the controller in the SET plus the length of data stored by the responder. Where no data has yet been received by a responder this field shall be set to 0x00000000.

Controllers may use this information to recover or continue a transfer.

TransferBlock Size (8-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a Controller in accordance with Section of this standard.

Initial Delay Time (32-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Inter-packet Delay Time (16-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Accumulated Byte Count (32-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Accumulated Byte Delay Time: (16-bit)
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

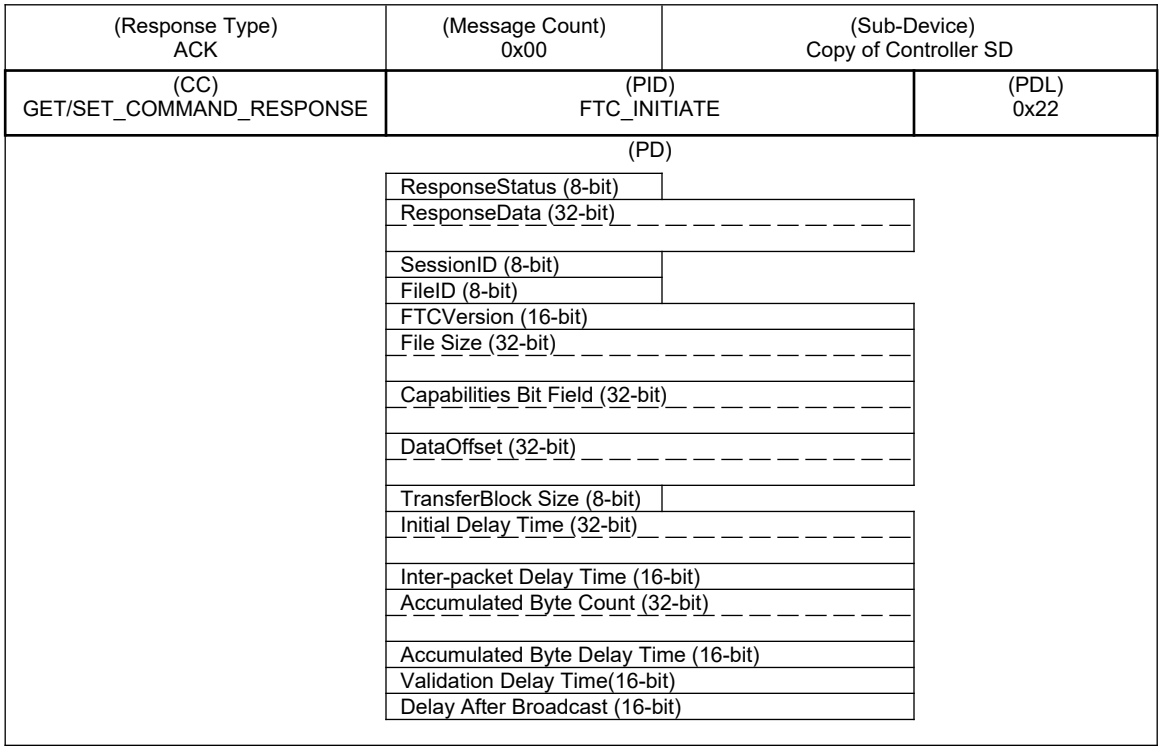
Validation Delay Time(16-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Delay After Broadcast (16-bit)
Responder provides a delay time that the controller should wait before sending the next GET.

13.1.4 Responder : Reply to GET:INITIATE or SET:INITIATE (Transfer Flag = Download)

+--
Responder : (GET/SET Response) – FTC_TRANSFER_DOWNLOAD



Data Description:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit)
This shall be set by the responder in accordance with Section of this standard.

SessionID (8-bit)
Responders that are not in a current FTC transfer process shall report FTC_DEF_NO_SESSIONID_OFFERED.

Responders that have begun a FTC Transfer process shall report the SessionID offered by the most recent successful SET:FTC_INITIATE.

FileID (8-bit)

The Responder should set the FileID value as defined in Table A- to indicate the file supported or set to FTC_DEF_MULTIPLE_FILEID to indicate that multiple files are supported.

Responders that have begun a FTC Transfer process shall report the FileID offered by the most recent successful SET:FTC_INITIATE.

Note: *If FileID is set to FTC_DEF_MULTIPLE_FILEID then the Controller should send a Get: FTC_FILELIST to determine the appropriate files, file types, and capabilities (by FileID).*

FTCVersion (16-bit)
This field is used to indicate the major and minor version of the FTC Standard being used by the Responder. See Section .

File Size (32-bit)
For a Responder responding to an initiate Download, these fields shall be set to total file size in bytes. If the file size is not known the responder shall respond with 0x00000000.

Capabilities Bit Field (32-bit)
This shall be set by the responder in accordance with Section of this standard.

A controller should utilize the declared capabilities to optimize the operation of the transfer process, or inform the user of any limitations.

DataOffset (32-bit):
The responder shall set this field to 0x00000000.

TransferBlock Size (8-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a Controller in accordance with Section of this standard.

Initial Delay Time (32-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Inter-packet Delay Time (16-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Accumulated Byte Count (32-bit):
The responder shall set this field to 0x00000000.

Accumulated Byte Delay Time: (16-bit)
The responder shall set this field to 0x00000000.

Validation Delay Time(16-bit):
This field shall be set by the responder in accordance with Section of this standard.

This value shall be used by a controller in accordance with Section of this standard.

Delay After Broadcast (16-bit)
Responder provides a delay time that the controller should wait before sending the next GET.

13.2 Transfer Data Upload (SET: FTC_TRANSFER_UPLOAD)

The FTC_TRANSFER_UPLOAD Parameter Message is used to transfer data to a responder.

Responders should only expect a data transfer after receipt of the SET:FTC_INITIATE Parameter Message.

Responders shall reject a FTC_TRANSFER_UPLOAD message using the ResponseStatus value of FTC_RS_UNSOLISITED_TRANSFER if it is received before a SET: FTC_INITIATE has been processed by the responder.

13.2.1 Sending Data TO a Responder

Controllers shall use the TransferBlock Size returned by the responder during the Initiate stage to determine the number of data bytes to be sent in each packet.

Controllers shall use the Initial-packet Packet delay values returned by the responder during the Initiate stage to control the time between the SET_FTC_INITIATE message and the first packet sent using the FTC_TRANSFER_UPLOAD Parameter Message.

Controllers shall use the Inter-packet Packet delay values returned by the responder during the Initiate stage to control the time between each packet sent.

Where a responder has declared a non zero Accumulated Byte Count and non zero Accumulated Byte Delay Time as part of the negotiate stage, the controller shall use these values in lieu of the inter-packet delay time after every N bytes where N = Accumulated Byte Count.

Controller: (SET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) FTC_TRANSFER_UPLOAD	(PDL) 0x07- 0xE7
(PD) SessionID (8-bit) DataOffset (32-bit) Data (0-224 Bytes) PacketCRC (16-bit)		

Data Description:

SessionID (8-bit) :
The SessionID is issued by a controller and will maintain the ID until the SET:Commit has completed.

The controller shall set this to the SessionID generated at the time of the SET:Initiate stage and as defined in section .

Responders shall reject the packet if the SessionID does not match that offered by the controller during the SET:Initiate stage using the ResponseStatus value of FTC_RS_SESSIONID_MISMATCH. The ResponseData should contain the expected SessionID.

DataOffset (32-bit) :
The Controller shall indicate the byte offset within the file for the data in this packet. This shall be used by the responder to store the data. The DataOffset will increment byte the TransferBlock Size after a GET:FTC_TRANSFER_UPLOAD ResponseStatus FTC_RS_STATUSOK.

Data (0-224 Bytes):
This field contains the data being transferred from the controller to the responder. The length of the data should be the TransferBlock Size for each packet sent, with the exception of the final packet where the length will be less than or equal to the TransferBlock Size.

PacketCRC (16-bit) :
Set as calculated in accordance with Section of this standard.

The PacketCRC shall be calculated from the first byte of the Parameter Data for a length of Parameter Data Length minus 2.

For example, if the PDL is 16, CRC will be calculated over the first 14 bytes of the Parameter Data.

This will include the SessionID, Data Offset and Data, but not the PacketCRC field itself.

13.2.1.1 Responder : SET Response

Response: (SET:Response)

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) FTC_TRANSFER_UPLOAD	(PDL) 0x09
(PD)		
ResponseStatus (8-bit)		
ResponseData (32-bit)		
DataOffset (32-bit)		

13.2.1.2 Responder : Data accepted : OK to continue

Data Description:

ResponseStatus (8-bit):
FTC_RS_STATUSOK: if everything was accepted by the Responder and it is ready for the next FTC packet. If the Responder requires more time to process the packet then the ResponseStatus shall include FTC_RS_STATUS_IN_PROGRESS and the ResponseData shall contain the time in milliseconds to delay before the next FTC message. See Section .

FTC_RS_TRANSFER_COMPLETE: The Responder believes that all of the data has been transferred and accepted. Internal validation will begin.

See Table A-3 for other possible ResponseStatus values and their meanings.

ResponseData (32-bit)
This shall be set by the responder in accordance with Section of this standard.

DataOffset (32-bit):
The responder shall return the next expected file offset. This should equate to the offset sent by the controller in the SET plus the length of data stored by the responder.

The controller may verify the expected and actual offsets in order to determine successful processing of the packet by the responder.

13.2.1.3 ResponseStatus : FTC_RS_STATUS_IN_PROGRESS
The responder shall respond with the ResponseStatus and ResponseData set to indicate successful transfer but a requirement for the controller to delay before continuing.

The use of this ResponseStatus should not normally be required if the responder has correctly declared it's delay characteristics in the Initiate stage

Use of FTC_RS_STATUS_IN_PROGRESS is not equivalent to the ACK_TIMER in so far as no use of QUEUED_MESSAGES is implied or required. In fact the ACK_TIMER response is not allowed from the Responder, only from a proxy device.

ResponseData (32-bit):

Responder shall return a value to indicate the wait time in ms.

The valid range is 0x00000000-0xFFFFFFFF.

Controllers shall wait the specified time in addition to any inter-packet delay time before continuing with the next FTC_TRANSFER_UPLOAD packet.

After the delay, the Controller should issue a GET:FTC_TRANSFER_UPLOAD until the ResponseStatus value is FTC_RS_STATUSOK. Alternatively, the Controller may send the next SET:FTC_TRANSFER_UPLOAD.

***Note**: As stated in Section the Controller may abort the transfer by issuing a SET:FTC_CANCEL*.

Data Offset (32-bit):
The responder shall return the next expected file offset. This should equate to the offset sent by the controller in the SET plus the length of data stored by the responder.

13.2.1.4 ResponseStatus : FTC_RS_PACKET_CRC_ERROR

The responder shall respond with the ResponseStatus if it has rejected the transfer because of a PacketCRC Error.

ResponseData (32-bit):
Responder shall report the received CRC value as calculated in the upper 16 bits and the Calculated CRC value in the lower 16 bits

The process of transfer is not complete.

A Controller may elect to resend the packet or abandon the transfer. A controller shall limit resends to a maximum of three attempts for any one packet.

In the event that the Controller decides to abandon the transfer (received error code or resend count exceeded), it shall do so in accordance with section

Data Offset (32-bit):
The responder shall return the next expected file offset. This should equate to the offset sent by the controller in the SET plus the length of data stored by the responder.

13.2.2 Getting Status from the Responder

Controller: (GET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) FTC_TRANSFER_UPLOAD	(PDL) 0x01
(PD) SessionID (8-bit)		

Response: (GET:Response)

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) FTC_TRANSFER_UPLOAD	(PDL) 0x05
(PD) ResponseStatus (8-bit) ResponseData (32-bit)		

Use of the GET is not normally required in a one-to-one transaction but may be required for situatoins outside the scope of this standard.

13.3 Commit Data (FTC_COMMIT)

A SET:FTC_COMMIT shall be used by a controller to instruct a responder to commit the data. The Responder is responsible for ensuring that the data has been validated and is appropriate for the device.

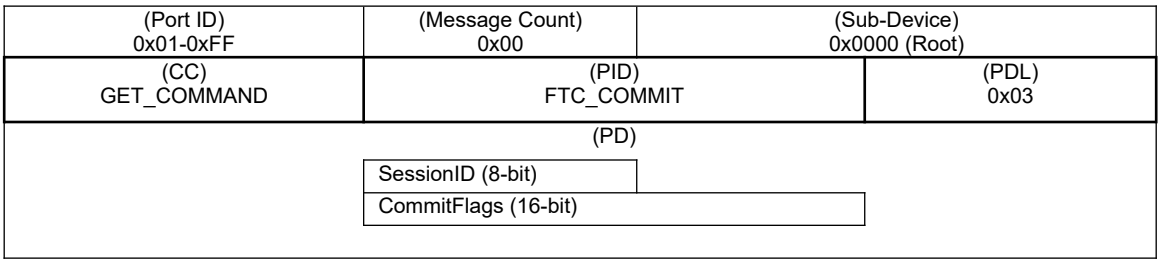
A controller shall only issue this command following the successful receipt of FTC_RS_STATUSOK in response to the FTC_TRANSFER_UPLOAD of the final TransferBlock.

Responders shall ensure that a response to the SET:FTC_COMMIT is issued *prior* to commencement of any firmware reload that results in the responder going off-line.

Controllers shall expect that, following receipt of a good response to the commit, the responder will go off-line. The responder may not necessarily re-appear with the same RDM UID. It may be necessary to run a discovery process before further communications with the responder(s) are possible. A responder can inform a controller of an expected change of UID via the ExpectedUID field.

13.3.1 Controller : GET Request to Commit Data

Controller: (SET)



Data Description:

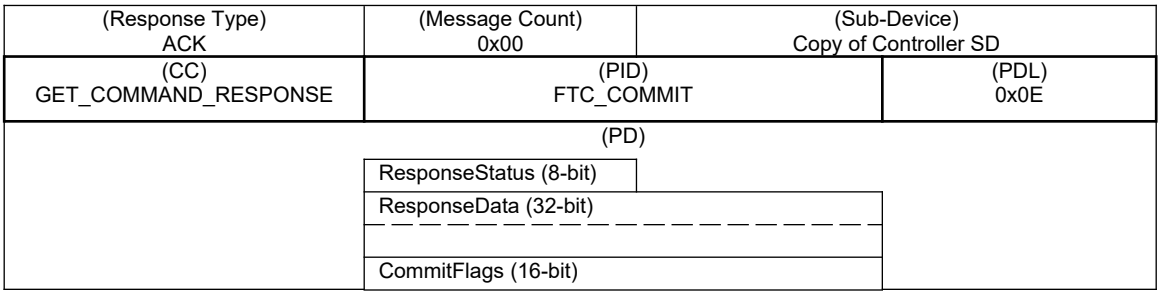
SessionID (8-bit) :
The controller shall set this to the SessionID relating to the transfer that has been validated and it now wishes to commit.

Controllers shall NOT use the value FTC_DEF_SESSIONID_ALL with this Parameter Message.
Responders shall test the SessionID given in this message against the SessionID given in the SET:INITIATE and issue a ResponseStatus of FTC_RS_SESSIONID_MISMATCH if they are different.

CommitFlags (16-bit) :
Same values as the TransferFlags in Table A-5.

13.3.2 Responder : GET Response to GET:FTC_COMMIT

Responder: (GET)



	CalculatedFileCRC (16-bit)
	ExpectedUID (6 bytes)

Data Description:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit)
This shall be set by the responder in accordance with Section of this standard.

CommitFlags (16-bit) :
Same values as the TransferFlags in Table A-5.

CalculatedFileCRC:
The CRC that the Responder has calculated for the entire FTCupload data received. The Responder checks this value against the FileCRC received from the Controller in the SET:FTC_INITIATE Parameter Message.

The CalculatedFileCRC is returned to the Controller in the response to the GET:FTC_COMMIT and SET:FTC_COMMIT Parameter Messages. When the Responder responds with FTC_RS_STATUSOK the CalculatedFileCRC shall match the FileCRC received from the Controller.

If the Responder has been unable to calculate the CalculatedFileCRC it returns CalculatedFileCRC as zero.

When the Responder rejects the FTC_Commit with FTC_RS_FILE_CRC_ERROR, the Responder shall report in the ResponseData field the Received File CRC value (from the SET:FTC_INITIATE)~~as calculated~~ in the upper 16 bits and the Calculated File CRC value in the lower 16 bits

ExpectedUID:
The UID that the Responder expects to be using after the FTC Upload has been successfully completed. When set to zero this indicates that no change in UID is expected – the Responder will continue to use it's current UID after the upload has completed.

13.3.3 Controller : SET Request to Commit Data

Controller: (SET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) FTC_COMMIT	(PDL) 0x03
(PD)		
	SessionID (8bit)	
	CommitFlags (16-bit)	

Data Description:

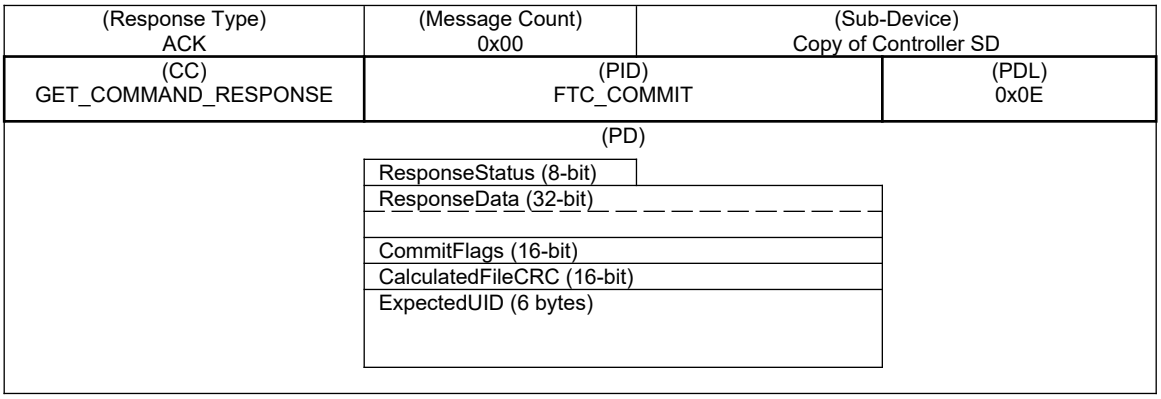
SessionID (8-bit) :
The controller shall set this to the SessionID relating to the transfer that has been validated and it now wishes to commit.

Controllers shall NOT use the value FTC_DEF_SESSIONID_ALL with this Parameter Message.
Responders shall test the SessionID given in this message against the SessionID given in the SET:INITIATE and issue a ResponseStatus of FTC_RS_SESSIONID_MISMATCH if they are different.

CommitFlags (16-bit) :
Same values as the TransferFlags in Table A-5.

13.3.4 Responder : SET Response

Responder: (SET)



Data Description:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit):
The valid range is 0x00000000-0xFFFFFFFF (ms).

Responder shall return a value to indicate the estimated time in ms the Responder may be offline and not able to communicate.

A controller shall not attempt to communicate with the responder during the estimated commit time.

CommitFlags (16-bit) :
This field is used to indicate defined commit information. See Table A-5.

CalculatedFileCRC:
The CRC that the Responder has calculated for the entire FTC Upload data received. The Responder checks this value against the FileCRC received from the Controller in the SET:FTC_INITIATE Parameter Message.

The CalculatedFileCRC is returned to the Controller in the response to the GET:FTC_COMMIT and SET:FTC_COMMIT Parameter Messages. When the Responder responds with FTC_RS_STATUSOK the CalculatedFileCRC shall match the FileCRC received from the Controller.

If the Responder has been unable to calculate the CalculatedFileCRC it returns CalculatedFileCRC as zero.

When the Responder rejects the FTC_COMMIT with FTC_RS_FILE_CRC_ERROR, the Responder shall report in the ResponseData field the Received File CRC value (from the FTC_INITIATE) in the upper 16 bits and the Calculated File CRC value in the lower 16 bits

ExpectedUID:
The UID that the Responder expects to be using after the FTC Upload has been successfully completed. When set to zero this indicates that no change in UID is expected – the Responder will continue to use it's current UID after the upload has completed.

Responder Manufacturers are strongly advised to not change the RDM UID as a result of any firmware update. This standard specifically does not provide support for responders that will change to an unknown UID after the

commit. Controllers will need to do a complete discovery if the expected UID does not appear after a reasonable amount of time has elapsed.

13.4 Cancel Transfer (FTC_CANCEL)

A controller that wishes to cancel transfer of data may issue a SET:FTC_CANCEL and the file transfer process shall be abandoned by the target responder, based on the specified SessionID or if the SessionID is FTC_DEF_SESSIONID_ALL. See Table A-.

After successful processing of a SET:FTC_CANCEL Parameter Message, a SessionID of FTC_DEF_NO_SESSIONID_OFFERED to any GET/SET:FTC_INITIATE requests.

13.4.1 Controller : SET Request to Cancel:

Controller: (SET) *Controller Request to cancel transfer.*

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) SET_COMMAND	(PID) FTC_CANCEL	(PDL) 0x01
(PD) SessionID		

Data Description:

SessionID (8-bit) :
The controller shall set this to the SessionID relating to the transfer that it wishes to cancel.

Controllers that wish to cancel a responder from any current transfers, without reference to a specific SessionID shall set this field to FTC_DEF_SESSIONID_ALL.

13.4.2 Responder : SET Response to Controller

Responder Response .

(Response Type) ACK	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) SET_COMMAND_RESPONSE	(PID) FTC_CANCEL	(PDL) 0x05
(PD) ResponseStatus (8-bit) ResponseData (32-bit)		

Data Descriptions:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit)
This shall be set by the responder in accordance with Section of this standard.

13.5 GET File LIST (FTC_FILELIST)

The FTC_FILELIST Parameter Message is used to retrieve File Information supported by the Responder.

Controller: (GET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)

(CC) GET_COMMAND	(PID) FTC_FILELIST	(PDL) 0x00
(PD) Not Present		

13.5.1 Responder : Reply to GET:FTC_FILELIST

This list is a packed list.

Responder : (GET Response)

(Response Type) ACK/ACK_OVERFLOW	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) FTC_FILELIST	(PDL) 0x2B-0xD7
(PD) FileID (8-bit) Capabilities (32-bit) File Description Text (Fixed 32 bytes) File Suffix Text (Fixed 6 bytes)		

Data Description:

FileID (8-bit):
The FileID field represents a unique defined identification for each file in the packed list. The valid range is as defined in Table A-. Values of FTC_DEF_NO_FILEID_OFFERED and FTC_DEF_MULTIPLE_FILEID are reserved with special meaning and should not be returned in this field.

The FileID must be unique for different files supported by a given Responder model, and may be different for different Responder models.

Capabilities Bit Field (32-bit)
This shall be set by the responder in accordance with section of this standard.

A controller should utilize the declared capabilities to optimize the operation of the transfer process, or inform the user of any limitations.

File Description Text (Fixed 32 byte field):
This field is a user defined description of the file content or use.

File Suffix Text (fixed 6 byte field):
This field is a user defined extension that may be used to select the desired filename. Any suffix separator should not be included in the file suffix text.

13.6 GET FTC_Transfer_Download (FTC_TRANSFER_DOWNLOAD)

The FTC_TRANSFER_DOWNLOAD Parameter Message is used to transfer data from a responder to a controller.

Responders should only expect a data transfer after receipt of the SET:FTC_INITIATE Parameter Message.

Responders shall reject a FTC_TRANSFER_DOWNLOAD message using the ResponseStatus value of FTC_RS_UNSOLICITED_TRANSFER if it is received before a SET: FTC_INITIATE has been processed by the responder.

Controller: (GET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) 0x0000 (Root)
(CC) GET_COMMAND	(PID) FTC_TRANSFER_DOWNLOAD	(PDL) 0x06
(PD)		
SessionID (8-bit)		
Command (8-bit)		

Data Description:

SessionID (8-bit) :
The controller shall set this to the SessionID relating to the transfer that it wishes to cancel.

Command (8-bit):
See Table A-6

Responder: (GET)

(Port ID) 0x01-0xFF	(Message Count) 0x00	(Sub-Device) Copy of Controller SD
(CC) GET_COMMAND_RESPONSE	(PID) FTC_TRANSFER_DOWNLOAD	(PDL) 0x08- 0xE7
(PD)		
ResponseStatus (8-bit)		
ResponseData (32-bit)		
SessionID (8-bit)		
Data (0-223 Bytes)		
PacketCRC (16-bit)		

Data Description:

ResponseStatus (8-bit)
This shall be set by the responder in accordance with Section of this standard.

ResponseData (32-bit)
This shall be set by the responder in accordance with Section of this standard.

SessionID (8-Bit)
This shall be set by the responder in accordance with Section of this standard.

Data (0-223 Bytes):
This field contains the data being transferred from the responder to the controller. The length of the data should be the TransferBlock size, with the exception of the final packet where the length will be less than or equal to the TransferBlock size.

PacketCRC (16-bit):
Set as calculated in accordance with Section of this standard.

The PacketCRC shall be calculated from the first byte of the Parameter Data for a length of Parameter Data Length minus 2.

If the responder can generate PacketCRC (as declared in the Generate_PacketCRC capability) the responder shall set the calculated CRC in the PacketCRC field.

The controller shall calculate the CRC on the received data and verify it matches the PacketCRC.

If the responder does not support generating PacketCRC, then it shall set the PacketCRC field to 0 and the controller shall ignore the field.

13.6.1 Getting Data FROM a responder

The controller sends a GET_FTC_TRANSFER_DOWNLOAD message with the command field set to FTC_CR_GET_NEXT_PACKET.

The responder shall decide the number of data bytes from the requested file to be returned in each response packet and will set the Parameter Data Length (PDL) field accordingly.

The controller can determine the number of file bytes received by examining the Parameter Data Length (PDL) field and subtracting the 6 bytes of the FTC_TRANSFER_DOWNLOAD response header.

If there is more data to return the responder sets the response code to FTC_RS_STATUSOK and the controller continues to send GET_FTC_TRANSFER_DOWNLOAD messages.

When there is no more data to return the responder sets the response status to FTC_RS_TRANSFER_COMPLETE.

The controller completes the file download process by sending a GET_FTC_TRANSFER_DOWNLOAD request with the command field set to FTC_CR_GET_FILE_CRC.

If the responder can generate FileCRC (as declared in the Generate_FileCRC capability) the responder shall set the calculated CRC for the whole fie in the ResponseData.

If the Responder does not support generating a FileCRC then the Responder would respond with a Response Status of FTC_RS_FILE_CRC_NOT_SUPPORTED.

The controller shall calculate the CRC on the whole file data received and verify it matches the FileCRC. If the CRC mismatches then it should discard all of the received data and retry the whole file download process.

13.6.2 Packet Timing

Controllers shall use the Initial-packet Packet delay values returned by the responder during the Initiate stage to control the time between the SET_FTC_INITIATE message and the first packet sent using the FTC_TRANSFER_DOWNLOAD Parameter Message.

Controllers shall use the Inter-packet Packet delay values returned by the responder during the Initiate stage to control the time between each packet request.

Accumulated Byte Count and Accumulated Byte Delay Time are not used for FTC_TRANSFER_DOWNLOAD.

Appendix A: Defined Parameters

Table A-1: General Defines

FTC_DEF_NO_FILEID_OFFERED 0x00
FTC_FILEID (*Valid range)* 0x01 – 0xFE
FTC_DEF_MULTIPLE_FILEID 0xFF
FTC_DEF_NO_SESSIONID_OFFERED 0x00
FTC_SESSIONID (*Valid range)* 0x01 – 0xFE
FTC_DEF_SESSIONID_ALL 0xFF

Table A-2: Parameter ID Defines

Note : Parameter Message PID values are not yet defined – final values will be assigned at time of formal release of the standard.

GET	SET	RDM Parameter ID's (Slot 20-21)	Value	Comment	Required
✓	✓	FTC_INITIATE			✓
✓	✓	FTC_TRANSFER_UPLOAD			✓
✓	✓	FTC_COMMIT			✓
	✓	FTC_CANCEL			✓
✓		FTC_FILELIST			
✓		FTC_TRANSFER_DOWNLOAD			

Table A-3: ResponseStatus Defines (GET/SET ResponseStatus)

ResponseStatus Defines	Value	Comments
FTC_RS_STATUSOK	0x00	Last request was good
FTC_RS_INITOK_UL	0x01	Requirements Response to an UPLOAD request was good [Controller sends TO Responder]
FTC_RS_INITOK_DL	0x02	Requirements Response to an DOWNLOAD request was good [Controller gets FROM Responder]
FTC_RS_STATUS_IN_PROGRESS	0x03	The Responder is busy with the last requested action
FTC_RS_TRANSFER_COMPLETE	0x04	Upload or Download final packet transfer was good
FTC_RS_MODAL_ERROR	0x05	The Responder is in the wrong state to accept the command.
FTC_RS_UNSOLICITED_TRANSFER	0x06	The Responder has received an unexpected data transfer request
FTC_RS_SESSIONID_MISMATCH	0x07	SessionID does not match expected ID
FTC_RS_UNSUPPORTED_FILEID	0x08	The Responder does not support the requested/offered FileID
FTC_RS_FILE_NOT_COMPATIBLE	0x09	The file is not compatible with the responder to Upload
FTC_RS_FILE_NOT_AVAILABLE	0x0A	The file is not currently available from responder to Download
FTC_RS_PACKET_CRC_ERROR	0x0B	The Responder reports CRC mismatch on this packet
FTC_RS_FILE_CRC_ERROR	0x0C	The Responder reports CRC mismatch on this FILE
FTC_RS_VALIDATION_ERROR	0x0D	The Responder cannot validate the data uploaded by the controller. The transfer has an invalid internal check, checksum or Manufacturer reason for rejection
FTC_RS_E137_LOCKACTIVE	0x10	Locked: Unlock using E137 PIDs
FTC_RS_OTHER_LOCKACTIVE	0x11	Locked: Unlock using other scheme
FTC_RS_WRITE_PROTECT	0x12	A memory protection scheme is preventing acceptance of data. SD card may be WRITE_PROECTED etc.
FTC_RS_INVALID_DIRECTION	0x13	The Responder does not support the requested transfer direction
FTC_RS_OFFSET_ERROR	0x14	Unexpected or out of range file offset
FTC_RS_FILESIZE_ERROR	0x15	Transfer data beyond declared FileSize
FTC_RS_FTCVERSION_ERROR	0x16	The Responder cannot deal with FTCVERSION as offered by Controller
FTC_RS_UNRESOLVED_ERROR	0x7F	Default exit of any internal state machine with an invalid state.
FTC_RS_FILE_CRC_NOT_SUPPORTED	0x17	FileCRC has been requested but the Responder does not support FileCRC
FTC_RS_MANUFACTURER_SPECIFIC	0x80-0xFF	Any undefined Status should be displayed as Unknown Status Response, together with the Status and Data

Table A-4: Use of ResponseData field

The ResponseData field provides additional detail regarding various ResponseStatus entries.

ResponseStatus	Response Data Field use	Comment
FTC_RS_STATUSOK	Data Offset for next Transfer	GET:FTC_INITIATE
FTC_RS_STATUSOK	Data Offset for next Transfer	SET:FTC_TRANSFER_UPLOAD
FTC_RS_STATUSOK	0x00000000	GET:FTC_TRANSFER_DOWNLOAD
FTC_RS_STATUSOK	Communication Offline Delay ms	GET/SET:FTC_COMMIT
FTC_RS_STATUS_IN_PROGRESS	Time in ms to wait while busy	Any GET
FTC_RS_INITOK_UP	Data Offset for next Transfer	GET/SET:FTC_INITIATE
FTC_RS_INITOK_DN	Data Offset for next Transfer	GET/SET:FTC_INITIATE
FTC_RS_PACKET_CRC_ERROR	Upper 16 bits ~~Expected~~ Received CRC Lower 16 bits Calculated CRC	GET/SET:FTC_TRANSFER_UPLOAD
FTC_RS_FILE_CRC_ERROR	Upper 16 bits ~~Expected~~ Received CRC Lower 16bits Calculated CRC	
FTC_RS_TRANSFER_COMPLETE	0xFFFFFFFF	SET:FTC_COMMIT
FTC_RS_UNSUPPORTED_FILEID	The Expected FileID	GET/SET:FTC_TRANSFER_UPLOAD
FTC_RS_SESSIONID_MISMATCH	The Expected SessionID	GET/SET:FTC_INITIATE
FTC_RS_MODAL_ERROR	The Responder Modal State #	Any GET/SET
FTC_RS_UNSOLICITED_TRANSFER	0x00000000	Any GET/SET
FTC_RS_FILE_NOT_COMPATIBLE	0x00000000	Any GET/SET
FTC_RS_FILE_NOT_AVAILABLE	0x00000000	SET:FTC_INITIATE
FTC_RS_VALIDATION_ERROR	0x00000000	GET/SET:FTC_INITIATE
FTC_RS_E137_LOCKACTIVE	0x00000000	GET/SET:FTC_COMMIT
FTC_RS_OTHER_LOCKACTIVE	0x00000000	SET:FTC_INITIATE
FTC_RS_WRITE_PROTECT	0x00000000	SET:FTC_INITIATE
FTC_RS_INVALID_DIRECTION	0x00000000	SET:FTC_INITIATE
FTC_RS_OFFSET_ERROR	Expected Offset	SET:FTC_INITIATE
FTC_RS_FILESIZE_ERROR	Expected Filesize	GET/SET:FTC_COMMIT
FTC_RS_FTCVERSION_ERROR	Upper 16 bits Supported Major Lower 16 bits Supported Minor	SET:FTC_TRANSFER_UPLOAD
FTC_RS_UNRESOLVED_ERROR	0x00000000	SET:FTC_TRANSFER_UPLOAD
FTC_RS_MANUFACTURER_SPECIFIC	Manufacturer Code	GET/SET:FTC_INITIATE

Table A-5: Transfer Flags/Commit Flags

Transfer Flag / Commit Flag	Bit Value	Comment
FTC_TF_TESTMODE	0x0001	1 if this is a test
FTC_TF_DOWNLOAD	0x0002	1 if this is a download
FTC_TF_RESERVED1	0x0004	
FTC_TF_RESERVED2	0x0008	
FTC_TF_RESERVED3	0x0010	
FTC_TF_RESERVED4	0x0020	
FTC_TF_RESERVED5	0x0040	
FTC_TF_RESERVED6	0x0080	
FTC_TF_RESERVED7	0x0100	
FTC_TF_RESERVED8	0x0200	
FTC_TF_RESERVED9	0x0400	
FTC_TF_RESERVED10	0x0800	
FTC_TF_RESERVED11	0x1000	
FTC_TF_RESERVED12	0x2000	
FTC_TF_RESERVED13	0x4000	
FTC_TF_RESERVED14	0x8000	

Table A-6: Transfer Download Commands

Transfer Download Command	Value	Comment
RESERVED	0x00	
FTC_CR_GET_NEXT_PACKET	0x01	Responder replies with file data at next offset (Controller offset ignored)
FTC_CR_RESEND_LAST_PACKET	0x02	Responder replies with file data at the LAST Offset returned. (Controller offset ignored)
RESERVED	0x03-0xFE	
FTC_CR_GET_FILE_CRC	0xFF	Controller sends after responder provides a response status of transfer complete. Responder replies with FileCRC for the last transferred file

Table A-7: Responder Capabilities

Responder Capabilities Defines	Bit Value	Comments
Accept_Upload_Flag	0x00000001	This shall be set if responder is able to accept offered FileID for upload.
Accept_Download_Flag	0x00000002	This shall be set if responder is able to accept offered FileID for download.
Process_FileCRC	0x00000004	Set if responder processes FileCRC on upload
Process_PacketCRC	0x00000008	Set if responder processes PacketCRC.
Generate_FileCRC	0x00000010	Set if responder generates FileCRC on download.
Generate_PacketCRC	0x00000020	Set if responder generates PacketCRC on download.
Reserve1	0x00000040	Reserved.
Reserve2	0x00000080	Reserved.
NSC_No_Interleave	0x00000100	Set if interleaving of NULL START Code DMX and/or non-FTC RDM Data during FTC transfer may result in responder failure.
NSC_Ignore	0x00000200	Set if NULL START Code DMX packets are ignored during transfers
Functional_Limit	0x00000400	Set if the responder limits normal functions during transfers.
Reserved3	0x00000800	
E137_Lock	0x00001000	Set if E1.37-1 PID Unlock required before upload accepted.
Other_Lock	0x00002000	Set if Proprietary Unlock required before upload accepted.
Reserved4	0x00004000	Reserved.
Broadcast Accepted	0x00008000	Set if responder does accept E1.37-4 Parameter Messages as Broadcast
Fail_May_Brick	0x00010000	Set if failure to successfully complete the transfer may cause the Responder to become inoperative.
Alternate_Error_Recovery	0x00020000	Set if Manufacturer is declaring support for some alternate mechanism of fatal error recovery. Such methods may involve processes not described by this standard such as USB uploads, media replacement, power cycling or other software solutions.
Reserved5	0x00040000	Reserved.
Reserved6	0x00080000	Reserved.
Reserved7	0x00100000	Reserved.
Reserved8	0x00200000	Reserved.
Reserved9	0x00400000	Reserved.
Testmode Supported	0x00800000	Responder supports use of Testmode
Reserved10	0x01000000	Reserved.
Reserved11	0x02000000	Reserved.
Reserved12	0x04000000	Reserved.
Reserved13	0x08000000	Reserved.
Reserved14	0x10000000	Reserved.
Reserved15	0x20000000	Reserved.
Reserved16	0x40000000	Reserved.
Reserved17	0x80000000	Reserved.

Appendix B: Control Flow – Firmware Upload

Example B-1 : File Upload Controller to Responder (Single File Id supported)

Controller sends SET:FTC_INITIATE Responder replies with FTC_RS_INITOK_UL and the Transfer Requirements
Controller waits Initial Delay Time
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 2)
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 2)
Repeat as required
Controller sends SET:FTC_TRANSFER_UPLOAD for final packet (Note 1) Responder replies with FTC_RS_TRANSFER_COMPLETE
Controller waits Validation Delay Time (Note 3)
Controller sends SET:FTC_COMMIT Responder replies with FTC_RS_STATUS_OK and Commit time
Controller waits returned Commit time
Controllers may assume transfer is complete after the expiry of the commit time (Note 4)

Note 1 : The TransferBlock Size is as determined in the response to the FTC_INITIATE request.

Note 2 : Inter-packet Delay Time is potentially extended after a series of packets has been transferred as defined by the Accumulated Byte Count and Accumulated Byte Delay Time.

Note 3 : The Validation Delay Time is as determined in the response to the FTC:INITIATE request.

Note 4 : The Controller should assume that the responder may have new features. Controller may wish to attempt a Device Info to determine the new software version.

Example B-2 : File Upload Controller to Responder (Multiple File IDs)

Controller sends GET:FTC_INITIATE or SET:FTC_INITIATE Responder replies with FTC_RS_STATUSOK and File ID of FTC_DEF_MULTIPLE_FILEID to indicate that multiple files are supported
Controller sends GET:FTC_FILELIST Responder returns list of File IDs, names and capabilities
Controller sends SET:FTC_INITIATE for required File ID Responder replies with FTC_RS_INITOK_UL and the Transfer Requirements
Controller waits Initial Delay Time
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 2)
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 2)
Repeat as required
Controller sends SET:FTC_TRANSFER_UPLOAD for final packet Responder replies with FTC_RS_TRANSFER_COMPLETE
Controller waits Validation Delay Time (Note 3)
Controller sends SET:FTC_COMMIT Responder replies with FTC_RS_STATUS_OK and Commit time
Controller waits returned Commit time
Controllers may assume transfer is complete after the expiry of the commit time (Note 4).

Note 1 : The TransferBlock Size is as determined in the response to the FTC:INITIATE request.

Note 2 : Inter-packet Delay Time is potentially extended after a series of packets has been transferred as defined by the Accumulated Byte Count and Accumulated Byte Delay Time.

Note 3 : The Validation Delay Time is as determined in the response to the FTC:INITIATE request.

Note 4: The Controller should assume that the responder may have new features. Controller may wish to attempt a Device Info to determine the new software version.

Example B-3 : File Upload Controller to Responder : Responder needs more time to check Transfer packets

Controller sends SET:FTC_INITIATE Responder replies with FTC_RS_INITOK_UL and the Transfer Requirements
Controller waits Initial Delay Time
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_IN_PROGRESS and Time required
Controller waits Time required (Note 2)
Controller sends SET:FTC_TRANSFER_UPLOAD (Note 1) Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 3)
Repeat as required

Note 1 : The TransferBlock Size is as determined in the response to the FTC:INITIATE request.

Note 2 : The Data Offset field in the reply with the FTC_RS_IN_PROGRESS indicates the next expected Data Offset.

Note 3 : Inter-packet Delay Time is potentially extended after a series of packets has been transferred as defined by the Accumulated Byte Count and Accumulated Byte Delay Time..

Example B-4 : Controller to Responder : Negotiate Fails

Controller sends SET:FTC_INITIATE with FT_TF_DOWNLOAD Responder replies with FTC_RS_INITOK_DL and the Transfer Requirements
Controller waits Initial Delay Time (Note 1)
Controller sends GET:FTC_TRANSFER_DOWNLOAD Responder replies with FTC_RS_STATUSOK and the packet data
Controller waits declared Inter-packet Delay Time (Note 1)
Controller sends GET:FTC_TRANSFER_DOWNLOAD Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 1)
Repeat as required
Controller sends GET:FTC_TRANSFER_DOWNLOAD for final packet Responder replies with FTC_RS_TRANSFER_COMPLETE and the packet data
Controller waits Validation Delay Time (Note 3)
Controller sends GET:FTC_COMMIT Responder replies with FTC_RS_STATUS_OK and File CRC
Controller calculates the CRC on the received data and verifies with received File CRC (Note 4)

Note 1 : A Responder performing File Download may require an Initial Delay Time due to internal processing to access the required data file.

Note 2 : A Responder performing a File Download may require an Inter-packet Delay in order for it to pre-calculate the Packet CRC or for other internal processing..

Note 3 : The Validation Delay Time is as determined in the response to the FTC:INITIATE request.

.

Note 4 : If the calculated File CRC result fails or the Controller never receives a Commit response the Controller shall discard the file data and restart the download from the beginning as the Responder is no longer in the FTC state.

Example B-5 : File Download Responder to Controller (Multiple File Ids)

Controller sends GET:FTC_INITIATE or SET:FTC_INITIATE with FT_TF_DOWNLOAD Responder replies with FTC_RS_STATUSOK and File ID of FTC_DEF_MULTIPLE_FILEID to indicate that multiple files are supported
Controller sends GET:FTC_FILELIST Responder returns list of File IDs, names and capabilities
Controller sends GET:FTC_INITIATE for required File ID with FT_TF_DOWNLOAD Responder replies with FTC_RS_INITOK_DL and the Transfer Requirements
Controller waits Initial Delay Time (Note 1)
Controller sends GET:FTC_TRANSFER_DOWNLOAD Responder replies with FTC_RS_STATUSOK and the packet data
Controller waits declared Inter-packet Delay Time (Note 1)
Controller sends GET:FTC_TRANSFER_DOWNLOAD Responder replies with FTC_RS_STATUSOK
Controller waits declared Inter-packet Delay Time (Note 1)
Repeat as required
Controller sends GET:FTC_TRANSFER_DOWNLOAD for final packet Responder replies with FTC_RS_TRANSFER_COMPLETE and the packet data
Controller waits Validation Delay Time (Note 3)
Controller sends GET:FTC_TRANSFER_DOWNLOAD command FTC_CR_GET_FILE_CRC Responder replies with FTC_RS_STATUS_OK and File CRC
Controller calculates the CRC on the received data and verifies with received File CRC (Note 4)

Note 1 : A Responder performing File Download may require an Initial Delay Time due to internal processing to access the required data file.

Note 2 : A Responder performing a File Download may require an Inter-packet Delay in order for it to pre-calculate the Packet CRC or for other internal processing..

Note 3 : The Validation Delay Time is as determined in the response to the FTC:INITIATE request.

Note 4 : If the calculated File CRC result fails or the Controller never receives a Commit response the Controller shall discard the file data and restart the download from the beginning as the Responder is no longer in the FTC state.

Appendix C: Transfer Time Guidance

This analysis is provided as informative. All calculations are approximate.

It is intended to provide guidance as to the suitability of the transfer process, especially as it might relate to larger files.

To transfer a file of 64Kbytes of data will require [65,535 ÷ 224 = 293] packets to be sent.

Assuming the best case of Break, MAB and inter-character time in the outgoing transmission, each transmitted packet takes:

*176 µs Break + 12us MAB+((25 + 231) * 44)µs [11.452ms].*

Assuming each packet gets an ACK, with a PDL of 5 bytes, the response takes:

*176 µs Break + 12us MAB+((25 + 5) * 44)µs [1.51ms].*

This assumes the best case of Break, MAB and inter-character time in the return (response) transmission.

*Assuming best case turn-around of 176 µs there is 293 * 176 µs [approximately 51.6ms] of system delay:*

*Best case Total transfer time estimated at (293 * 11.45ms Tx Data) [3,354.85ms] + (293 * 1.420ms Rx Data) [416ms]+ System delay [51.6ms] which gives approximately 3,822.51ms.*

*Assuming worst case turnaround of 2.8ms there is 292 * 2.8ms [817.6ms] of system delay, so the transfer time would increase to 4,640.11ms.*

Further delays may occur depending on the time taken by the responder to validate transfers and reprogram memory.

If we take the above values and assume a transfer file size of 1Mbyte, the amount of packets to be transmitted would be 4,688 and the expected duration would be 61,160.16 ms.

	64kb	**1024kb**	**2048kb**
Packets	293	4,688	9,376
Duration [ms]	3,822.51	61,160.16	122,320.32

A file size of 4GB would, however, take almost 3 days using this protocol. The use of this standard to transfer files of this size may not be appropriate for some applications.

Appendix D: CRC Algorithm and Example

A CRC is called an *n*-bit CRC when its check value is *n*-bits. For a given *n*, multiple CRCs are possible, each with a different polynomial. This standard uses a 16-bit CRC commonly known as CRC-16-ANSI, CRC-16-IBM, or Modbus CRC using the polynomial $x^{16}+x^{15}+x^2+1$.

The Cyclical Redundancy Check (CRC) field is two bytes, containing a 16bit binary value.

The CRC value is calculated by the Controller.

The Responder recalculates a CRC during receipt of the message and compares the calculated value against the actual value it received in the CRC field. If the two values are not equal, an error results.

CRC Generation

The CRC is created by preloading a 16-bit register to all 1's. Then a process begins of applying successive 8-bit bytes of the message to the current contents of the register. Only the eight bits of data in each character are used for generating the CRC.

During generation of the CRC, each 8-bit character is exclusive ORed with the register contents. The result is then shifted in the direction of the least significant bit (LSB), with a zero filled into the most significant bit (MSB) position. The LSB is extracted and examined. If the LSB was a 1, the register is then exclusive ORed with a preset, fixed value. If the LSB was a 0, no exclusive OR takes place.

This process is repeated until eight shifts have been performed. After the last (eighth) shift, the next 8-bit character is exclusive ORed with the register's current value, and the process repeats for eight more shifts as described above. The final contents of the register, after all the characters of the message have been applied, is the CRC value.

A procedure for generating a CRC is:

1) Load a 16-bit register with 0xFFFF (all 1's). Call this the CRC register.
2) Exclusive OR the first 8-bit byte of the message with the low–order byte of the 16-bit CRC register, putting the result in the CRC register.
3) Shift the CRC register one bit to the right (toward the LSB), setting the MSB to zero. Extract and examine the LSB.
4) If the LSB is 0, repeat Step 3 (another shift). If the LSB is 1, exclusive OR the CRC register with the polynomial value 0xA001 (1010 0000 0000 0001).
5) Repeat Steps 3 and 4 until 8 shifts have been performed. When this is done, a complete 8-bit byte will have been processed.
6) Repeat Steps 2 through 5 for the next 8-bit byte of the message. Continue doing this until all bytes have been processed.
7) The CRC register now contains the CRC value.

An example of a C language function performing CRC generation is shown below. All of the possible CRC values are preloaded into two arrays, which are simply indexed as the function increments through the message buffer.

One array contains all of the 256 possible CRC values for the high byte of the 16bit CRC field, and the other array contains all of the values for the low byte.

Indexing the CRC in this way provides faster execution than would be achieved by calculating a new CRC value with each new character from the message buffer.

CRC Generation Function

The function takes two arguments:

```
unsigned char *puchMsg ;     //pointer to the message buffer containing binary data to be used for generating the CRC

unsigned short usDataLen ;  //The quantity of bytes in the message buffer.
```

The function returns the CRC as a type unsigned short.

```
unsigned short CRC16(*puchMsg, usDataLen)

{
unsigned char uchCRCHi = 0xFF ; /* high byte of CRC initialized */
unsigned char uchCRCLo = 0xFF ; /* low byte of CRC initialized */
unsigned uIndex ;               /* will index into CRC lookup table */

while (usDataLen--) /* pass through message buffer */
{
      uIndex   = uchCRCHi ^ *puchMsgg++ ; /* calculate the CRC */
      uchCRCHi = uchCRCLo ^ auchCRCHi[uIndex] ;
      uchCRCLo = auchCRCLo[uIndex] ;
}
return (uchCRCHi << 8 | uchCRCLo) ;
}
```

High-Order Byte Table

/* Table of CRC values for high–order byte */

```
static unsigned char auchCRCHi[] = {
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01,
0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0,
0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01,
0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40, 0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41, 0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81,
0x40
} ;
```

Low-Order Byte Table

/* Table of CRC values for low–order byte */

```
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2, 0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4,
0x04, 0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E, 0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09,
0x08, 0xC8, 0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A, 0x1E, 0xDE, 0xDF, 0x1F, 0xDD,
0x1D, 0x1C, 0xDC, 0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6, 0xD2, 0x12, 0x13, 0xD3,
0x11, 0xD1, 0xD0, 0x10, 0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32, 0x36, 0xF6, 0xF7,
0x37, 0xF5, 0x35, 0x34, 0xF4, 0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE, 0xFA, 0x3A,
0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38, 0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA, 0xEE,
0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C, 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0, 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2,
0x62, 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4, 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F,
0x6E, 0xAE, 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68, 0x78, 0xB8, 0xB9, 0x79, 0xBB,
0x7B, 0x7A, 0xBA, 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C, 0xB4, 0x74, 0x75, 0xB5,
0x77, 0xB7, 0xB6, 0x76, 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0, 0x50, 0x90, 0x91,
0x51, 0x93, 0x53, 0x52, 0x92, 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54, 0x9C, 0x5C,
0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E, 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98, 0x88,
0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A, 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86, 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80,
0x40
} ;
```