Flexible & Interoperable Data Transfer (FIT) Protocol

Revision History

Revision	Effective Date	Description
1.0	May 2010	Initial Release
1.1	March 2011	Updated License
1.2	April 2011	Updated File Header Information Corrected Compressed Timestamp header description
1.3	April 2012	Corrected minor errors in documentation (omissions, typos, incorrect data)

Table of Contents

List of Figures

List of Tables

1 Introduction

The Flexible and Interoperable Data Transfer (FIT) protocol is a format designed specifically for the storing and sharing of data that originates from sport, fitness and health devices. It is specifically designed to be compact, interoperable and extensible. This document will describe the FIT file structure and interpretation.

The FIT protocol defines a set of data storage templates (FIT messages) that can be used to store information such as user profiles and activity data in files. Any FIT-compliant device can interpret a FIT file from any other FIT-compliant device. A software development kit (SDK) is provided to generate code and libraries specific to a product's requirements. The SDK enables efficient use of a binary format at the embedded level, to significantly reduce the development effort and allow for rapid product development.

The following example use case illustrates how the FIT protocol is used to transfer personal monitoring information acquired during exercise to an internet database (Figure 1-1):

1. ANT+ Sensors measure parameters such as heart rate and running speed
2. Data is broadcast in real time, using interoperable ANT+ data formats
3. Session events and real time activity data is collected and saved into a FIT file on a display device
4. The FIT file is transferred to the PC using ANT File Share (ANT-FS)
5. The FIT data may be used directly on the PC or transferred to internet applications

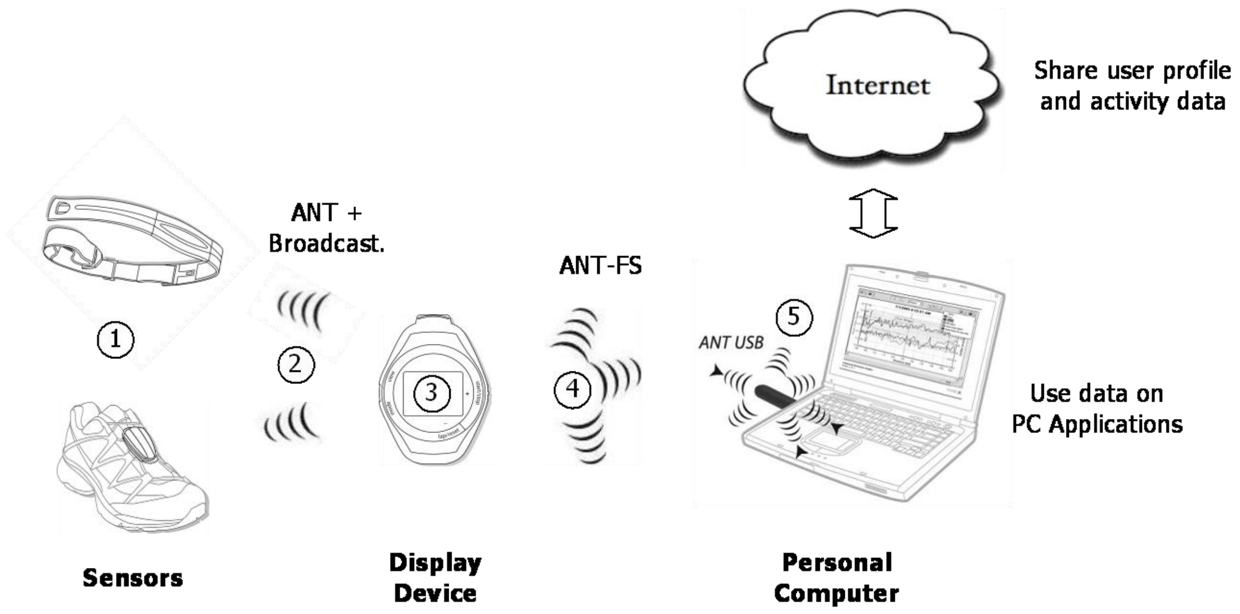


Figure 1-1. Data flow between devices using FIT protocol

After the initial wireless sensor data is collected, the FIT protocol provides a consistent format allowing all devices in the subsequent chain to share and use the data.

The FIT file protocol was designed to provide:

- Interoperability of device data across various device platforms
- Scalability from small embedded targets to large web databases
- Forward compatibility, allowing the protocol to grow and retain existing functionality
- Automated compatibility across platforms of different native endianness

The FIT file protocol consists of:

- A file structure
- A global list of FIT messages and FIT, fields together with their defined data types
- Software Development Kit (SDK) to configure target products and generate the necessary FIT code and libraries

2 Related Documents

The following supplementary documentation and files are provided in the SDK:

- FIT File Types Document
- FIT Global Messages and Fields (Profile .xls)
- FIT code generator
- FIT to CSV Conversion Tool
- Reference code examples
- Example FIT files

Many FIT applications will involve the ANT-FS protocol to facilitate the wireless transfer of FIT files. For further information regarding ANT-FS and related details for transferring FIT files specifically, refer to the following documents:

- ANT File Share (ANT-FS) Technology
- ANT-FS Reference Design and User Manual

3 Overview of the FIT File Protocol

A FIT file contains a series of records that, in turn, contain a header and content. The record content is either a definition message that is used to specify upcoming data, or a data message that contains a series of data-filled fields (Figure 3-1). The FIT protocol defines the type and content of messages, the data format of each message's field, and methods of compressing data (if applicable).

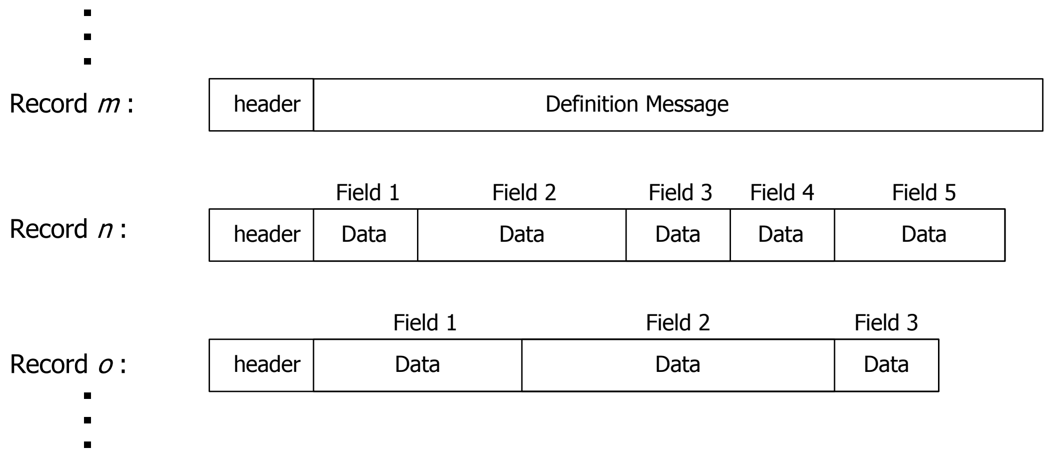


Figure 3-1. Basic FIT File components

3.1 FIT Profiles

There are two types of FIT profiles: global and product. All available FIT messages are outlined in the *Global FIT Profile*. This is then broken down into smaller, subset, *Product Profiles* outlining product-specific FIT messages (Figure 3-2).

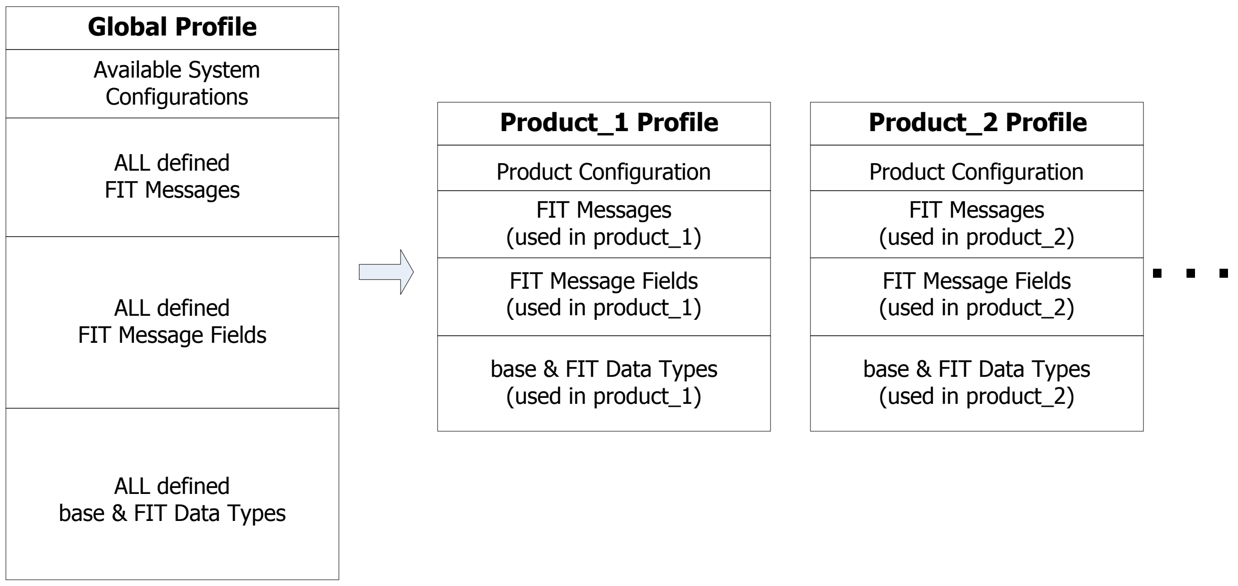

Global Profile
Available System Configurations
ALL defined FIT Messages
ALL defined FIT Message Fields
ALL defined base & FIT Data Types
Product_1 Profile
Product Configuration
FIT Messages (used in product_1)
FIT Message Fields (used in product_1)
base & FIT Data Types (used in product_1)
Product_2 Profile
Product Configuration
FIT Messages (used in product_2)
FIT Message Fields (used in product_2)
base & FIT Data Types (used in product_2)

Figure 3-2. Components of Global and Product FIT Profiles

Each profile consists of system configuration information, defined FIT messages and fields, base data types, and FIT data types (Figure 3-2).

3.1.1 Global Profile

The *Global FIT Profile* consists of the complete collection of available system configurations, FIT messages, fields and data types as described below:

- System Configuration: Describes system parameters such as byte endianness and alignment. The FIT protocol supports multiple system configurations
- FIT Messages: define the FIT fields contained within each FIT message
- FIT Message Fields: define the base type and format of data within each FIT field
- FIT Types: describe the FIT field as a specific type of FIT variable (unsigned char, signed short, etc)

New configurations, messages and data types may be added at any time. The relationships between FIT messages, fields and base types are illustrated in Figure 3-3.

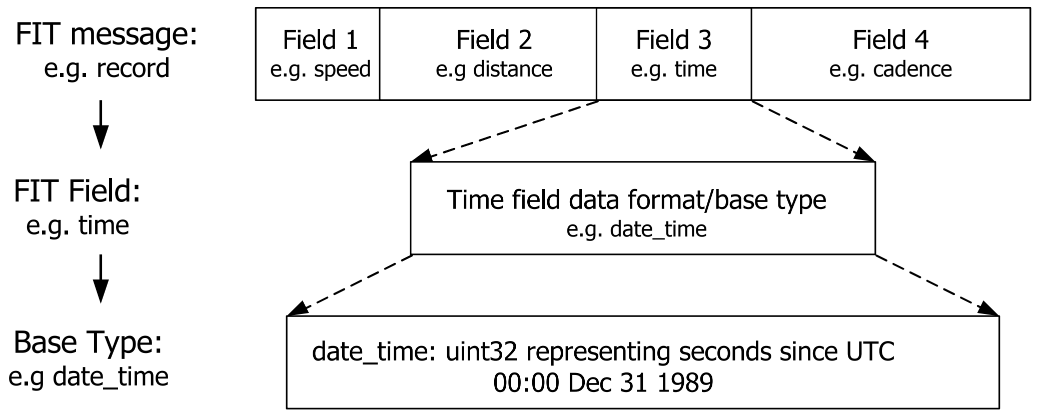


Figure 3-3. FIT message, field and base type structure

3.1.2 Product Profile

Not all messages defined in the global FIT profile will be relevant to a particular application. A *Product Profile* is an application specific subset of the Global Profile that defines only the necessary data messages in the configuration of the product's architecture (Figure 3-2). An SDK is provided to allow the developer to select the desired system configuration and relevant data messages and then generate application specific FIT code.

Two different FIT devices may use different product profiles or versions of the complete Global Profile. This may result in one device receiving a FIT message that it does not recognize. When this occurs, the FIT file is maintained in its entirety and any unrecognized messages are simply ignored by the decoder without interrupting the operation of the receiving device, or causing any errors. Similarly, if a device does not receive data that it may expect, it will simply fill those fields with an invalid value rather than creating errors. In this way, the FIT protocol will ensure compatibility across devices that may not have the exact same profiles implemented. These compatibility processes are discussed in more detail in later sections.

3.2 FIT File Protocol

The FIT protocol defines the process for which profiles are implemented and files are transferred. Figure 3-4 provides an overview of the FIT process. Typically, ANT+ broadcast data is collected by a display device. The display device would then encode the data into the FIT file format according to its product profile (i.e. product profile 1). The FIT file is then transferred to another device which would then decode the received files according to its own implemented product profile (i.e. product profile 2).

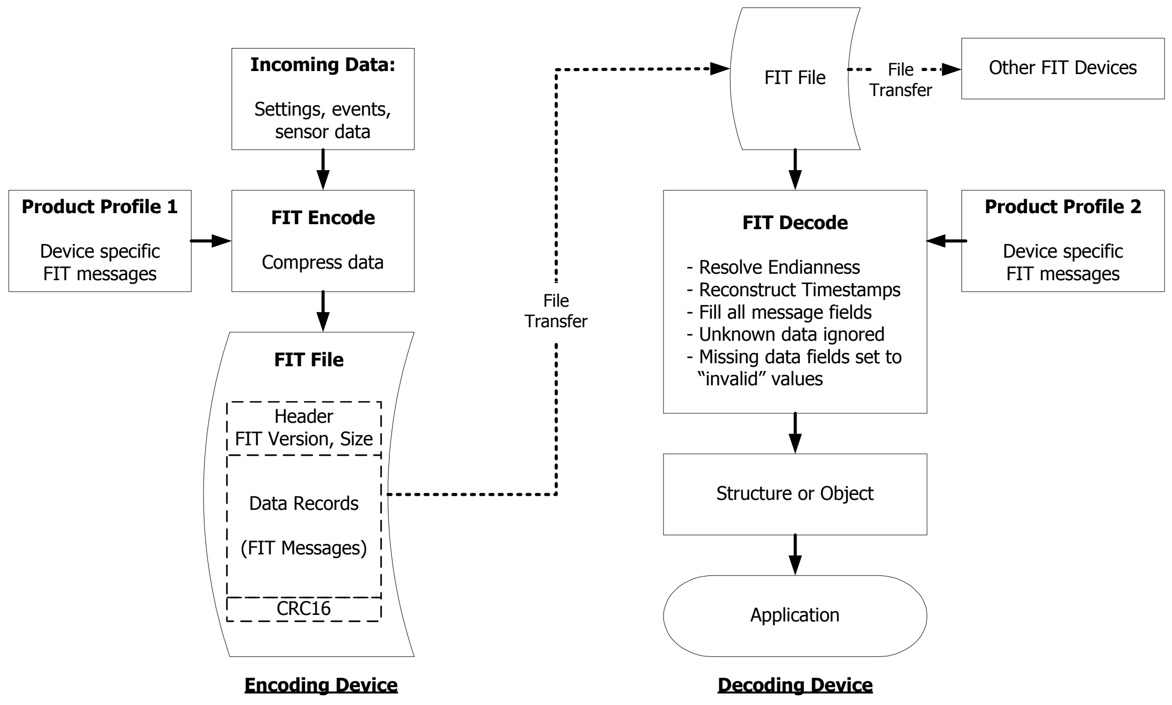


Figure 3-4. Overview of the FIT File protocol

Incoming data such as settings, events and sensor data are written into FIT message fields according to the formats defined by the device's product profiles. The FIT encoding process is optimized, such that only valid fields are written to the file. The file can then be transferred to another FIT device. When the data is used by the receiving device, it is decoded according to its implemented product profiles, which relate the received FIT messages to the global FIT message list. The decoded values will then be passed as structures or objects to the application.

The SDK code will resolve native endianness, reconstruct timestamp information and fill all message fields appropriately. If there is a difference in profile version between the two devices, any missing data will be set to invalid or default values as defined in the FIT protocol, and any unknown messages or data will be ignored. The FIT file is maintained in its original form for transfer to other devices, if desired.

3.3 FIT File Structure

All FIT files have the same structure which consists of a File Header, a main Data Records section that contains the encoded FIT messages, followed by a 2 byte CRC (Figure 3-5.a).

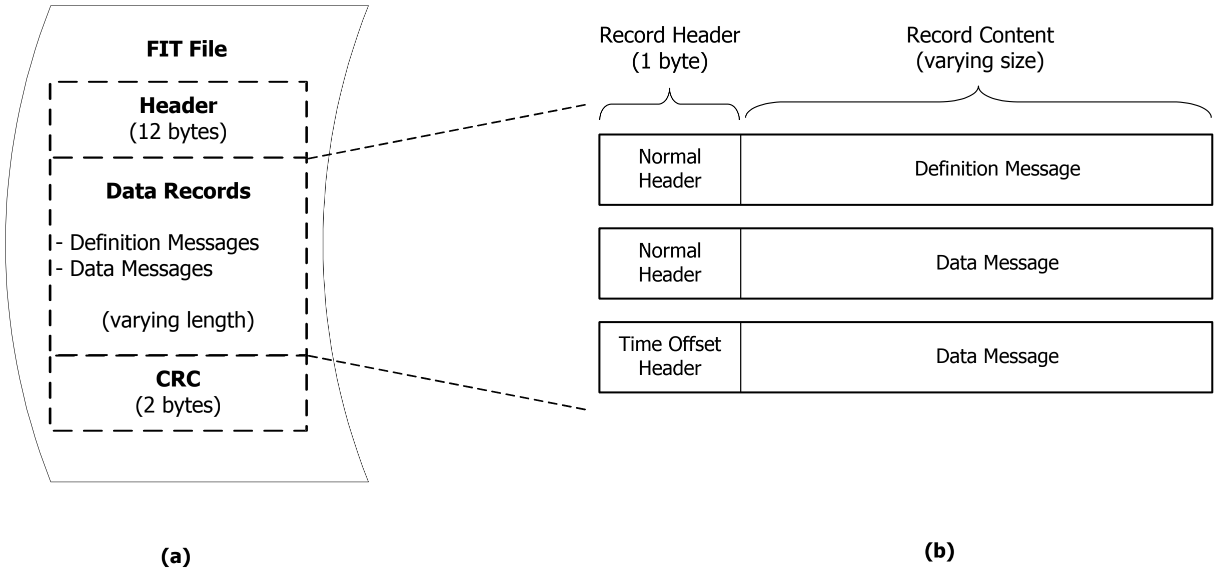


Figure 3-5. (a) The FIT file structure (b) Data Record types

3.3.1 File Header

The file header provides information about the FIT File. The minimum size of the file header is 12 bytes including protocol and profile version numbers, the amount of data contained in the file and data type signature. The size of the file header may be increased to add additional information. The CRC is optional with a 14 byte file header. The CRC in the file header allows the CRC of the file to be computed as the file is being written when the amount of data to be contained in the file is not known.

Table 3-1 outlines the FIT file header formats.

Table 3-1. Byte Description of File Header

Byte	Parameter	Description	Size (Bytes)
0	Header Size	Indicates the length of this file header including header size. Minimum size is 12. This may be increased in future to add additional optional information.	1
1	Protocol Version	Protocol version number as provided in SDK	1
2	Profile Version LSB	Profile version number as provided in SDK	2
3	Profile Version MSB		
4	Data Size LSB	Length of the Data Records section in bytes Does not include Header or CRC	4
5	Data Size		
6	Data Size		
7	Data Size MSB		
8-11	Data Type LSB	**ASCII values for ".FIT".** A FIT binary file opened with a text editor will contain a readable ".FIT" in the first line.	4
9	Data Type		
10	Data Type		
11	Data Type MSB		
12	CRC LSB	Contains the value of the CRC (section 0) of Bytes 0 through 11, or may be set to 0x0000. This field is optional.	2
13	CRC MSB		

3.3.2 CRC

The final 2 bytes of a FIT file contain a 16 bit CRC in little endian format. The CRC is computed as follows:

```
FIT_UINT16 FitCRC_Get16(FIT_UINT16 crc, FIT_UINT8 byte)
{
   static const FIT_UINT16 crc_table[16] =
   {
      0x0000, 0xCC01, 0xD801, 0x1400, 0xF001, 0x3C00, 0x2800, 0xE401,
      0xA001, 0x6C00, 0x7800, 0xB401, 0x5000, 0x9C01, 0x8801, 0x4400
   };
   FIT_UINT16 tmp;

   // compute checksum of lower four bits of byte
   tmp = crc_table[crc & 0xF];
   crc  = (crc >> 4) & 0x0FFF;
   crc  = crc ^ tmp ^ crc_table[byte & 0xF];

   // now compute checksum of upper four bits of byte
   tmp = crc_table[crc & 0xF];
   crc  = (crc >> 4) & 0x0FFF;
   crc  = crc ^ tmp ^ crc_table[(byte >> 4) & 0xF];

   return crc;
}
```

3.3.3 Data Records

The data records in the FIT file are the main content and purpose of the FIT protocol. There are two kinds of data records:

- **Definition Messages** – define the upcoming data messages. A definition message will define a local message type and associate it to a specific FIT message, and then designate the byte alignment and field contents of the upcoming data message.
- **Data Messages** – contain a local message type and populated data fields in the format described by the preceding definition message. The definition message and its associated data messages will have matching local message types. There are two types of data message:
    - Normal Data Message
    - Compressed Timestamp Data Message

These messages will be further explained in section 4. All records contain a 1 byte Record Header that indicates whether the Record Content is a definition message, a normal data message or a compressed timestamp data message (Figure 3-5.b). The lengths of the records vary in size depending on the number and size of fields within them.

All data messages are handled locally, and the definition messages are used to associate local data message types to the global FIT message profile. For example, a definition message may specify that data messages of local message type 0 are Global FIT "lap" messages (Figure 3-6). The definition message also specifies which of the "lap" fields are included in the data messages (start_time, start_position_lat, start_position_long, end_position_lat, end_position_long), and the format of the data in those fields. As a result, data messages can be optimized to contain only data, and the local message type is referenced in the header.

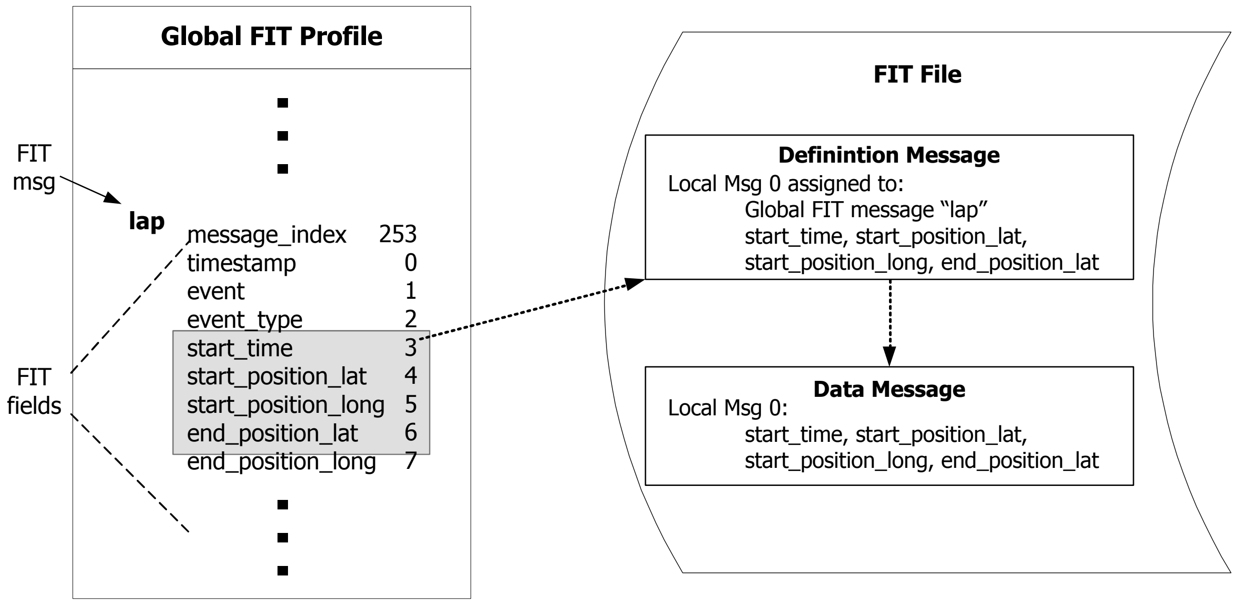


Figure 3-6. Definition message assigns Global FIT message to local message type 0. Data messages are referenced to local message type.

4 Record Format

A FIT record consists of two parts: a Record Header and the Record Content. The record header indicates whether the record content contains a definition message, a normal data message or a compressed timestamp data message. The record header also has a Local Message Type field that references the local message in the data record to its global FIT message.

4.1 Record Header Byte

The Record Header is a one byte bit field. There are actually two types of record header: *normal header* and *compressed timestamp header*. The header type is indicated in the most significant bit (msb) of the record header. The normal header identifies whether the record is a definition or data message, and identifies the local message type. A compressed timestamp header is a special compressed header that may also be used with some local data messages to allow a compressed time format.

4.1.1 Normal Header

A value of 0 in Bit 7 of the record header, indicates that this is a Normal Header. The bit field description for a normal header is shown below in Table 4-1.

Table 4-1. Normal Header Bit Field Description

Bit	Value	Description
7	0	Normal Header
6	0 or 1	Message Type 1: Definition Message 0: Data Message
5	0	Reserved
4	0	Reserved
0 - 3	0 - 15	Local Message Type

Message Type

The message type indicates whether the record contains a definition or data message.

Local Message Type

The Local Message Type is used to create an association between the definition message, data message and the FIT message in the Global FIT Profile.

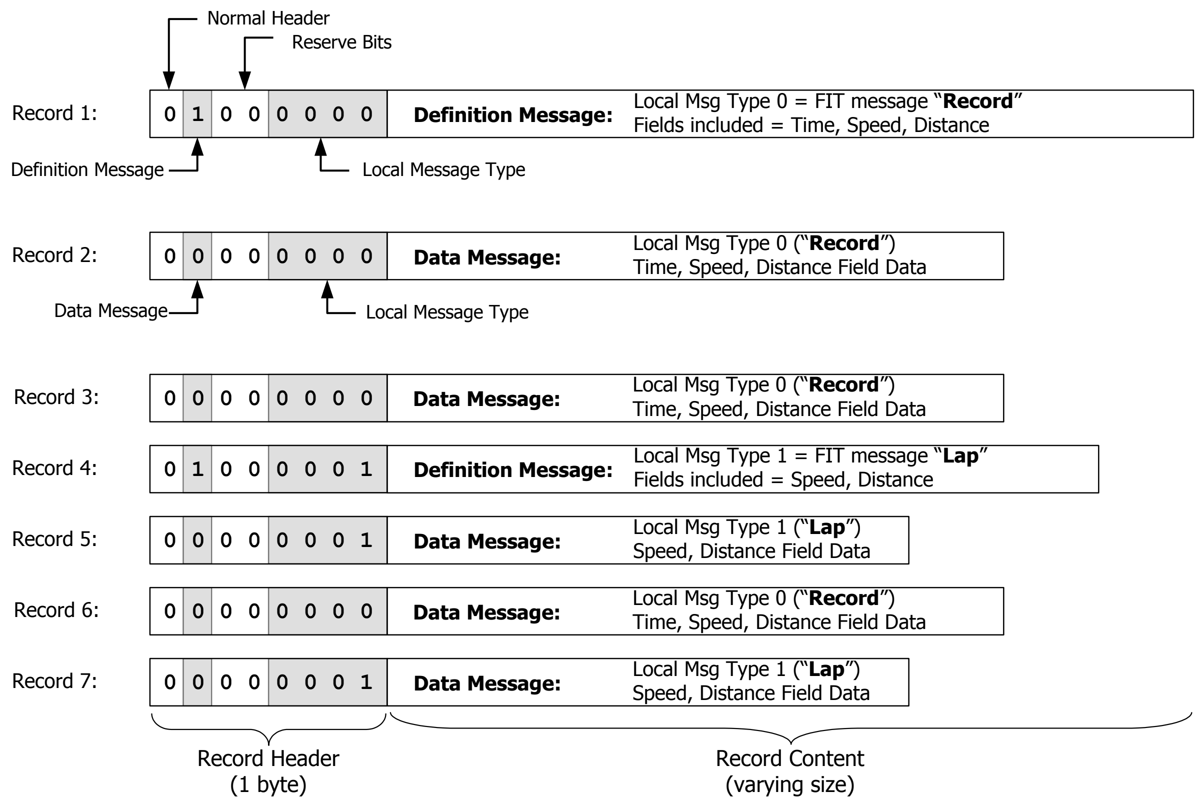
- **Definition Message**: In a definition message, the local message type is assigned to a Global FIT Message Number (mesg_num) relating the local messages to their respective FIT messages

- **Data Message**: The local message type associates a data message to its respective definition message, and hence, its global FIT message. A data message will follow the format as specified in its definition message of matching local message type.

Local Message Types can be redefined within a single FIT file, please refer to section 4.6.3 for best practices when using a single local message type for different records.

Example:

Figure 4-1 shows an example of using data records with normal headers to designate definition and respective data messages for recording FIT "record" and "lap" messages.



Normal Header
Reserve Bits
Record 1: 0 1 0 0 0 0 0 0 Definition Message: Local Msg Type 0 = FIT message "Record" Fields included = Time, Speed, Distance
Definition Message
Local Message Type
Record 2: 0 0 0 0 0 0 0 0 Data Message: Local Msg Type 0 ("Record") Time, Speed, Distance Field Data
Data Message
Local Message Type
Record 3: 0 0 0 0 0 0 0 0 Data Message: Local Msg Type 0 ("Record") Time, Speed, Distance Field Data
Record 4: 0 1 0 0 0 0 0 1 Definition Message: Local Msg Type 1 = FIT message "Lap" Fields included = Speed, Distance
Record 5: 0 0 0 0 0 0 0 1 Data Message: Local Msg Type 1 ("Lap") Speed, Distance Field Data
Record 6: 0 0 0 0 0 0 0 0 Data Message: Local Msg Type 0 ("Record") Time, Speed, Distance Field Data
Record 7: 0 0 0 0 0 0 0 1 Data Message: Local Msg Type 1 ("Lap") Speed, Distance Field Data
Record Header (1 byte)
Record Content (varying size)

Figure 4-1. Normal Headers: example definition and data messages

4.1.2 Compressed Timestamp Header

The compressed timestamp header is a special form of record header that allows some timestamp information to be placed within the record header, rather than within the record content. In applicable use cases, this allows data to be recorded without the need of a 4 byte timestamp in every data record. The bit field description of a compressed timestamp header is shown in Table 4-2 below.

Table 4-2. Compressed Timestamp Header Bit Field Description

Bit	Value	Description
7	1	Compressed Timestamp Header
5 – 6	0 - 3	Local Message Type
0 - 4	0 - 31	Time Offset (seconds)

Note, this type of record header is used for a data message only.

Local Message Type

In order to compress the header, only 2 bits are allocated for the local message type. As a result, the use of this special header is restricted to local message types 0-3, and cannot be used for local message types 4-15. The local message type can be redefined within a single FIT file, please refer to section 4.6.3 for more details.

Time Offset

The five least significant bits (lsb) of the header represent the compressed timestamp, in seconds, from a fixed time reference. The time resolution is not configurable. The fixed time reference is provided in the form of any FIT FIT message containing a full, four byte timestamp recorded prior to the use of the compressed timestamp header (see example). The 5-bit time offset rolls over every 32 seconds; hence, it is necessary that any two consecutive compressed timestamp records be measured less than 32 seconds apart.

The actual timestamp value is determined by concatenating the most significant 27 bits of the previous timestamp value and the 5 bit value of the time offset field. Rollover must be taken into account such that:

If Time Offset >= (Previous Timestamp)&0x0000001F
(i.e. offset value is greater than least significant 5 bits of previous timestamp):

Timestamp = (Previous timestamp)&0xFFFFFFE0 + Time Offset

If Time Offset < (Previous Timestamp)&0x0000001F
(i.e. offset is less than least significant 5 bits of previous timestamp):

Timestamp = (Previous timestamp)&0xFFFFFFE0 + Time Offset + 0x20
The addition of 0x20 accounts for the rollover event

Refer to Figure 4-2 for an example of using compressed timestamp headers. In this example, local message type 0 is used to define a message containing a both a timestamp field and multiple data fields (Record 1). Local message type 1 defines a message containing only data fields (Record 2).

Record 3 is a data message which includes a timestamp value of 0xXXXXXX3B. For the purpose of this example, the values of the upper 3 bytes do not change and are set as 0xXX “don’t care” values.

Record 4 is a data message using a compressed timestamp header. As the Time Offset value is the same as the 5 least significant bits of the previous timestamp, the calculated timestamp for record 4 is 0xXXXXXX3B.

Record 5 is another data message using a compressed timestamp header. The Time Offset value is greater than the 5 least significant bits of the previous timestamp by 2 seconds, so the calculated timestamp for record 5 becomes 0xXXXXXX3D.

Record 6 Time Offset value is 00010, which is smaller than the 5 least significant bits of the previous timestamp (5 lsb of 0xXXXXXX3D is 11101), indicating a rollover event has occurred. Therefore, the timestamp becomes 0xXXXXXX42.

Similarly, record 7 shows an increase in time of 3 seconds and the timestamp becomes 0xXXXXXX45, and record 8 shows a rollover event resulting in a timestamp of 0xXXXXXX61.

Finally, if a new data message containing a timestamp is recorded (i.e. record 9), then this becomes the new time reference for any subsequent compressed timestamp data records, such as records 10 and 11.

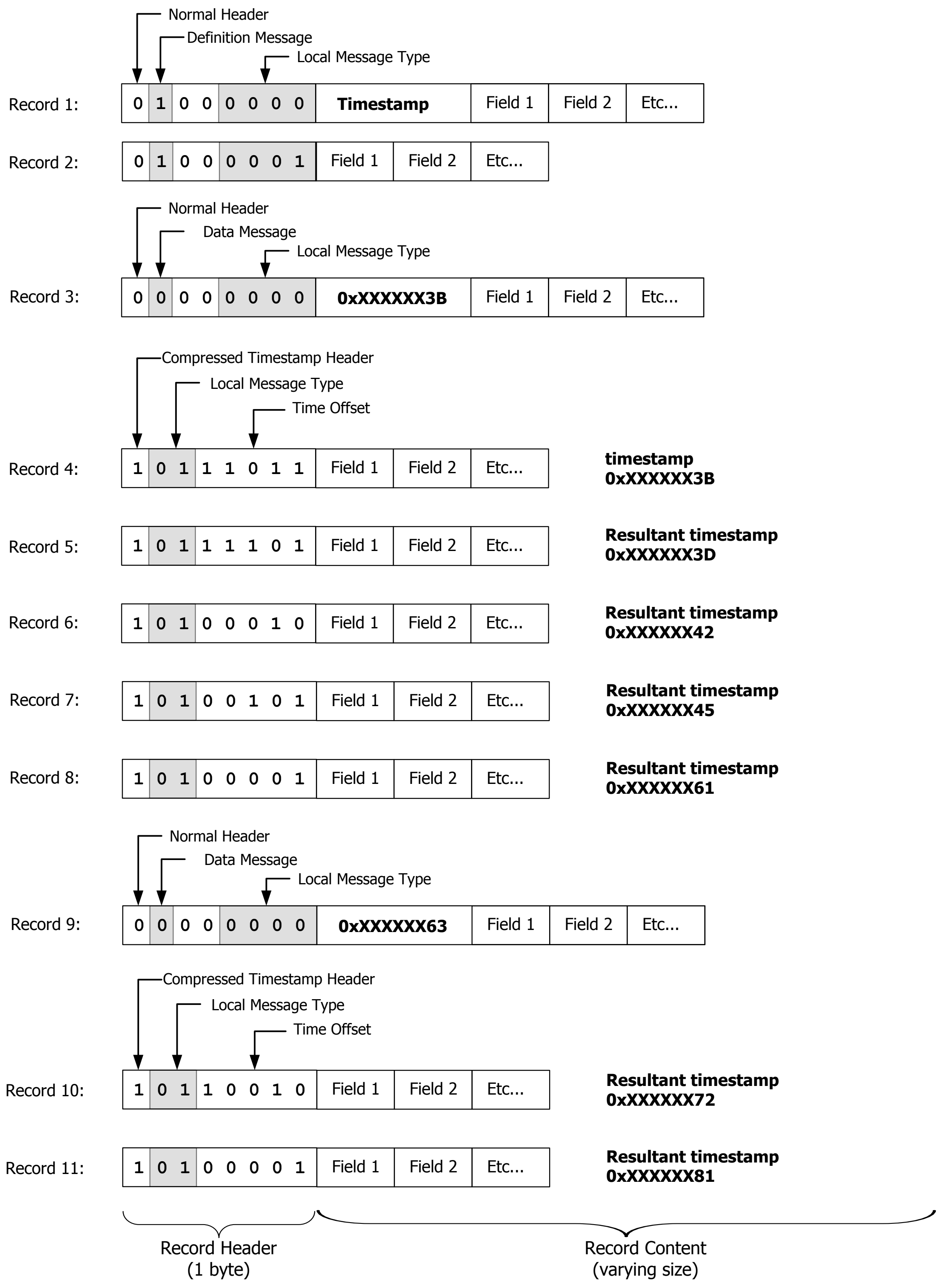


Figure 4-2. Compressed Timestamp Headers: example definition, normal data, and compressed timestamp data messages

4.2 Record Content

The record content contains one of two messages:

- Definition Message: describes the architecture, format, and fields of upcoming data messages
- Data Message: contains data that is formatted according to a preceding definition message

Definition and data messages are associated through the local message type. A data message must always be specified by a definition message before it can be used in a FIT file. If a data message is sent without first being defined, it will cause a decode error and the data will not be interpreted. Definition message are used by the conversion tools to interpret subsequent data messages contained in a FIT file. For more details see best practices in section 4.4.

4.2.1 Definition Message

The definition message is used to create an association between the local message type contained in the record header, and a Global Message Number (mesg_num) that relates to the global FIT message.

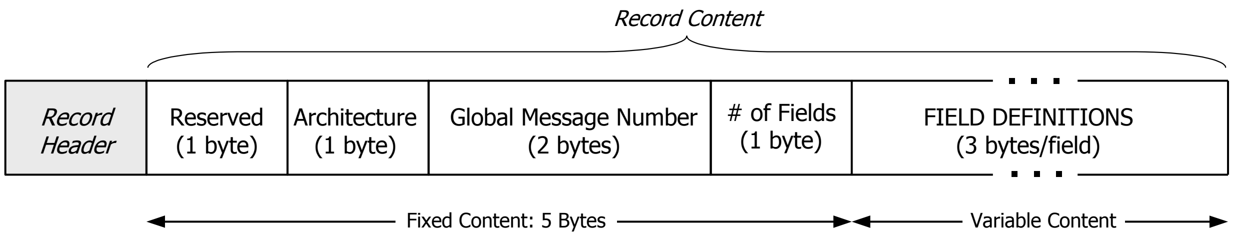


Figure 4-3. Definition Message Structure

The record contents of a definition message are outlined in Table 4-3 below.

Table 4-3. Definition Message Contents

Byte	Description	Length	Value
0	Reserved	1 Byte	0
1	Architecture	1 Byte	Architecture Type 0: Definition and Data Messages are Little Endian 1: Definition and Data Message are Big Endian
2-3	Global Message Number	2 Bytes	0:65535 – Unique to each message *Endianness of this 2 Byte value is defined in the Architecture byte
4	Fields	1 Byte	Number of fields in the Data Message
5 - END	Field Definition	3 Bytes (per Field)	See Field Definition Contents (Table 4-4 and)

Architecture Type

The Architecture Type describes whether the system architecture is big or little endian. All data in the related definition and upcoming data message will follow this format.

Global Message Number

The Global Message Number relates to the Global FIT Message. For example, the Global FIT Message "Record" has the global message number "20". All Global Message Numbers are found in the mesg_num base type defined in the SDK.

Fields

Fields defines the number of FIT fields that will be included in the data message. For example, if a given FIT message had 10 defined FIT fields, the application may only choose to send 4 of those FIT fields in the data message. In this case, the Fields byte would be set to "4". All FIT messages and their respective fields are listed in the global FIT profile.

Field Definition

The Field Definition bytes are used to specify which FIT fields of the global FIT message are to be included in the upcoming data message in this instance. Any subsequent data messages of a particular local message type are considered to be using the format described by the definition message of matching local message type. All FIT messages and their respective FIT fields are listed in the global FIT profile. Each Field Definition consists of 3 bytes as detailed in Table 4-4. Refer to Figure 4-5 for an example definition message.

Table 4-4. Field Definition Contents

Byte	Name	Description
0	Field Definition Number	Defined in the Global FIT profile for the specified FIT message
1	Size	Size (in bytes) of the specified FIT message's field
2	Base Type	Base type of the specified FIT message's field

Field Definition Number

The Field Definition Number uniquely identifies a specific FIT field of the given FIT message. The field definition numbers for each global FIT message are provided in the SDK.

Base Type

Base Type describes the FIT field as a specific type of FIT variable (unsigned char, signed short, etc). This allows the FIT decoder to appropriately handle invalid or unknown data of this type. The format of the base type bit field is shown below in Table 4-5. All available Base Types are fully defined in the fit.h file included in the SDK and as listed in Table 4-6.

Table 4-5. Base Type Bit Field

Bit	Name	Description
7	Endian Ability	**0** - for single byte data **1** - if base type has endianness (i.e. base type is 2 or more bytes)
5-6	Reserved	Reserved
0-4	Base Type Number	Number assigned to Base Type (provided in SDK)

When the decoder encounters unknown or invalid data, it will assign an invalid value according to the designated base type. Base type numbers (bits 0:4) and their invalid values can also be found in the fit.h file provided in the SDK and as listed in Table 4-6 below.

Table 4-6. FIT Base Types and Invalid Values

Base Type #	Endian Ability	Base Type Field	Type Name	Invalid Value	Size (Bytes)	Comment
0	0	0x00	enum	0xFF	1	
1	0	0x01	sint8	0x7F	1	2's complement format
2	0	0x02	uint8	0xFF	1	
3	1	0x83	sint16	0x7FFF	2	2's complement format
4	1	0x84	uint16	0xFFFF	2	
5	1	0x85	sint32	0x7FFFFFFF	4	2's complement format
6	1	0x86	uint32	0xFFFFFFFF	4	
7	0	0x07	string	0x00	1	Null terminated string
8	1	0x88	float32	0xFFFFFFFF	4	
9	1	0x89	float64	0xFFFFFFFFFFFFFFFF	8	
10	0	0x0A	uint8z	0x00	1	
11	1	0x8B	uint16z	0x0000	2	
12	1	0x8C	uint32z	0x00000000	4	
13	0	0x0D	byte	0xFF	1	Array of bytes. Field is invalid if all bytes are invalid.

4.2.2 Data Message

Once a global FIT message has been associated to a local message type, and the format of the FIT fields defined, data messages may be written to the FIT file. Definition messages have a minimum length of 8 bytes, excluding the record header; however, data messages can be very compact.

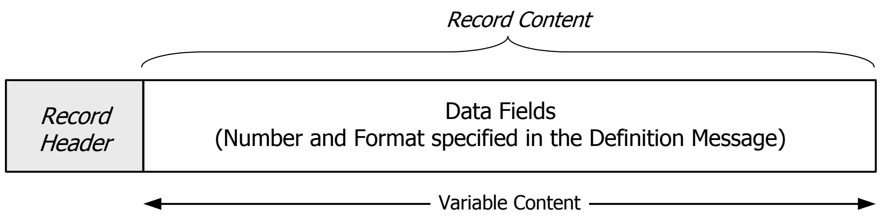


Figure 4-4. Data Message Structure

A data message must start with a normal or compressed timestamp header indicating its local message type, and the record content must be formatted according to the definition message of matching local message type.

4.3 FIT File Example

The example in Figure 4-5 shows a simple FIT activity file containing the 12-byte file header, data records and 2 byte CRC.

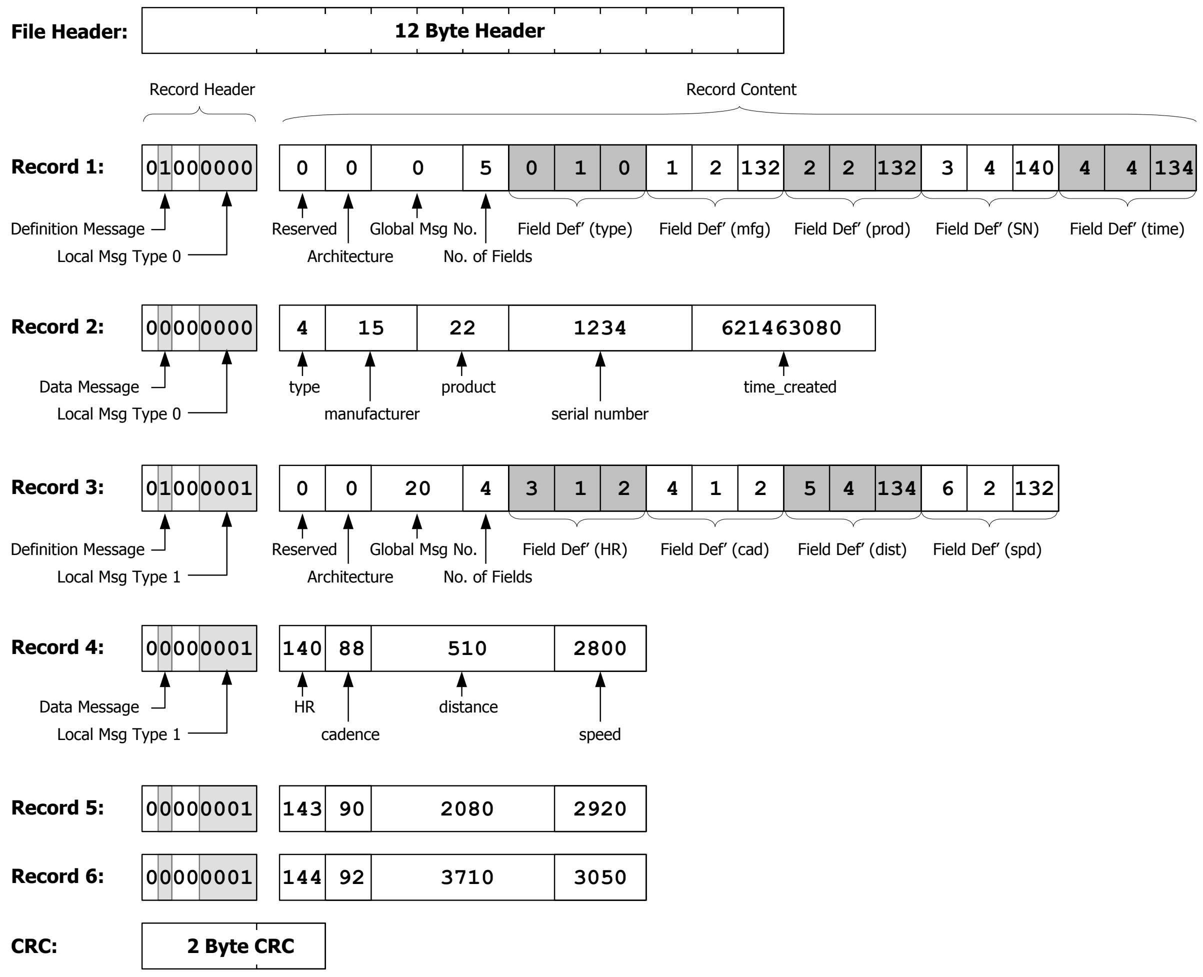


Figure 4-5. Definition and data message example

Record 1 (definition message: "file_id" (mesg_num = 0x00))

Indicates the record is a definition message specifying the upcoming data message (of local message type 0) are:

- Little Endian
- Global Message Number 0 identifies FIT "file_id" message
- The FIT file_id fields that will be included in the associated data message are:
    - Field Definition Number: 0 (type); Size: 1 byte; Base Type: 0 (enum)
    - Field Definition Number: 1 (manufacturer); Size: 2 bytes; Base Type: 132 (uint16)
    - Field Definition Number: 2 (product); Size: 2 bytes; Base Type: 132 (uint16)
    - Field Definition Number: 3 (serial number); Size: 4 bytes; Base Type: 140 (uint32z)
    - Field Definition Number: 4 (time_created); Size: 4 bytes; Base Type: 134 (uint32)

Global Message Number is found in the mesg_num type of the FIT protocol

Field Definition Numbers for each FIT message are found in the FIT profile provided in the SDK. *Size* and *Base Type* definitions are located in the fit.h file in the SDK, or as listed in Table 4-6.

Record 2 (data message: "file_id" (local msg type = 0))

Indicates the record is a data message of local message type 0. Data is formatted according to the definition message of local message type 0:

- Little Endian, FIT "file_id" message
- Included Fields and Data:
    - type: 4* (activity file)
    - manufacturer: 15* (Dynastream)
    - product: 22
    - serial number: 1234
    - time_created: 621463080 (~14 Aug 2009)

  * These values are defined in the FIT protocol

Record 3 (definition message: "record" (mesg_num = 0x14))

Indicates the record is a definition message specifying the upcoming data message (of local message type 1) are:

- Little Endian
- Global Message Number 20 identifies FIT "record" message
- The FIT record fields that will be included in the associated data message are:
    - Field Definition Number: 3 (heart_rate); Size: 1 byte; Base Type: 2 (uint8)
    - Field Definition Number: 4 (cadence); Size: 1 bytes; Base Type: 2 (uint8)
    - Field Definition Number: 5 (distance); Size: 4 bytes; Base Type: 134 (uint32)
    - Field Definition Number: 6 (speed); Size: 2 bytes; Base Type: 132 (uint16)

Global Message Number is found in the mesg_num type of the FIT protocol

Field Definition Numbers for each FIT message are found in the FIT profile provided in the SDK. *Size* and *Base Type* definitions are located in the fit.h file in the SDK, or as listed in or as listed in Table 4-6.

Record 4 (data message: "record" (local msg type = 1))

Indicates the record is a data message of local message type 1. Data is formatted according to the definition message of local message type 1:

- Little Endian, FIT "record" message
- Included Fields and Data:
    - heart_rate: 140 (bpm)
    - cadence: 88 (rpm)
    - distance: 510 (cm)
    - speed: 2800 (mm/s)

Record 5 (data message: "record" (local msg type = 1))

Indicates the record is a data message of local message type 1. Data is formatted according to the definition message of local message type 1:

- Little Endian, FIT "record" message
- Included Fields and Data:
    - heart_rate: 143 (bpm)
    - cadence: 90 (rpm)
    - distance: 2080 (cm)
    - speed: 2920 (mm/s)

Record 6 (data message: "record" (local msg type = 1))

Indicates the record is a data message of local message type 1. Data is formatted according to the definition message of local message type 1:

- Little Endian, FIT “record” message
- Included Fields and Data:

heart_rate:	144 (bpm)
cadence:	92 (rpm)
distance:	3710 (cm)
speed:	3050 (mm/s)

Note that in this example, the fields are defined in the order of increasing field number. This does not have to be the case. Field definitions do not need to be in the order of increasing field number, however, the order the fields are recorded in data message MUST follow the order they are defined in the definition message.

4.4 Dynamic Fields

The interpretation of some message fields will depend on the value of another field. This is called a Dynamic Field. For example, Field Definition #3 of the event message is data. If the event field is equal to fitness_equipment then data is interpreted as fitness_equipment_state. Similarly, if event is battery then data is interpreted as battery level.

The advantage of dynamic fields is it allows the interpretation of a field to change without writing a new message definition. This optimizes the size of a file.

4.5 Common Fields

There are some select fields that are common across all FIT messages. These fields have a reserved field number that is also common across all FIT messages.

4.5.1 Message_Index (Field # = 254)

This field allows messages to be indexed with a common method.

For example, a FIT_LookupMessage C function is provided in the SDK that returns the location of a message in a file by global message number and message index. The SDK C code provides a FIT_LookupMessage function that returns the location of a message in a file by specifying the global message number and message index. The message index field also contains a bit to indicate a selected message. For example, the active user profile could be selected by setting the selected bit in the message index. Note, message_index fields must be recorded sequentially (i.e. numbered starting from 0 and incremented in steps of 1).

Message index can be used to refer to a previously defined record. For example, the user_profile message has a message_index field. Multiple user_profile messages may be recorded using the message_index field. The blood_pressure message has a user_profile_index field that relates back to the user_profile_message. For example, if the blood_pressure message has a user_profile_index =1, this will correspond to the user_profile message that has message_index=1.

4.5.2 Timestamp(Field # = 253)

Common timestamp field for all FIT messages. This field may be used in combination with the compressed timestamp header.

4.6 Best Practices

To properly encode/decode FIT files, the following MUST be included:

- 12 byte header
- Data Record 1: file_id Definition Message
- Data Record 2: file_id Data Message
- Data Records: Ensure appropriate definition messages are included in the FIT file prior to recording any associated data messages. Note field definitions do not need to be in the order of increasing field number, however, the order the fields are recorded in data message MUST follow the order they are defined in the definition message
- 2 Byte CRC

4.6.1 File ID Messages

The purpose of the "file_id" message is to uniquely identify the file in a global system. The fields in the data message may include file type, manufacturer, product, serial number, time created and file number depending on the FIT file type.

4.6.2 Defining Data Messages

A data message must always be specified by a definition message prior to recording any data. Once a data message has been properly defined, the FIT file can be properly decoded. Even if a device's implemented profile does not include all of the FIT messages or fields contained in the FIT file, it will be decoded without error: unrecognized data will be ignored, and any expected values not included will be assigned invalid values. If a data message is recorded without an appropriate definition message, an error will occur.

Often, multiple data messages of the exact same format are recorded. In this case, it is best practice to use a single definition message for all data messages; rather than recording a definition message for each data message. For example, in Figure 4-6, two types of data messages are being recorded: lap and record messages. The FIT file on the left contains a definition message for each data message. Although technically correct, this method of recording data is sub optimal; instead, define the lap and record messages once at the beginning of the file, followed by all lap and record messages as shown in the FIT file on the right.

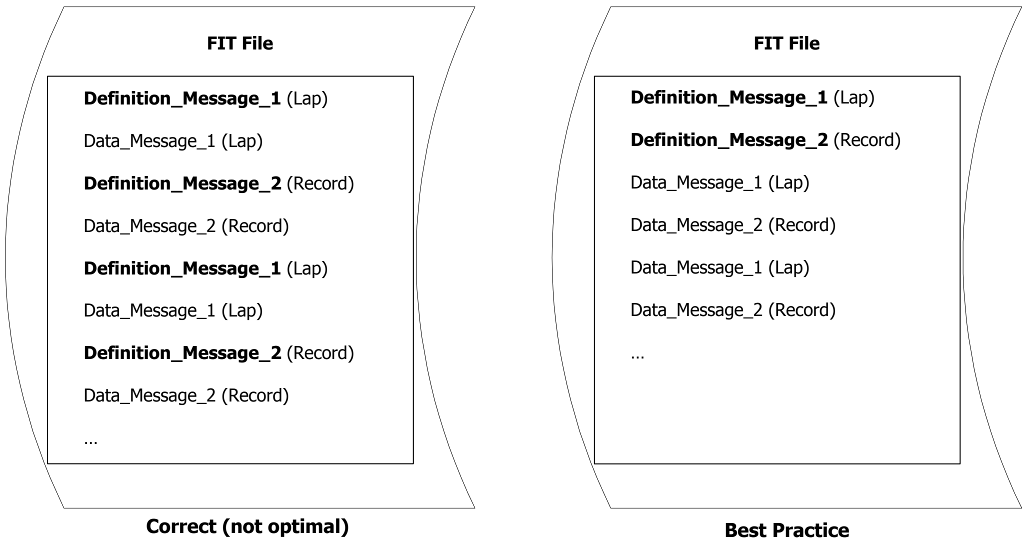


Figure 4-6. Best Practice for Defining Data Messages.

4.6.3 Re-defining Local Message Types

Local message types can be redefined within a single FIT file. Figure 4-5, for example, shows a FIT file using a single local message type (i.e. 0) to record both the "file_id" and "record" data. Note that this FIT file contains the same data that was recorded in section 4.3. The number of local message types used in a file should be minimized in order to minimize the RAM required to decode the file. For example, embedded devices may only support decoding data from local message type 0. The advantage of using multiple local message types is the file size is optimized because new definition messages are not required to interleave different message types. Multiple local message types should be avoided in file types such as settings where messages of the same type can be grouped together.

Care must be taken when redefining local message types. If data message formats are recorded without the new definition message, unpredictable results will occur and may cause the decoder to fail.

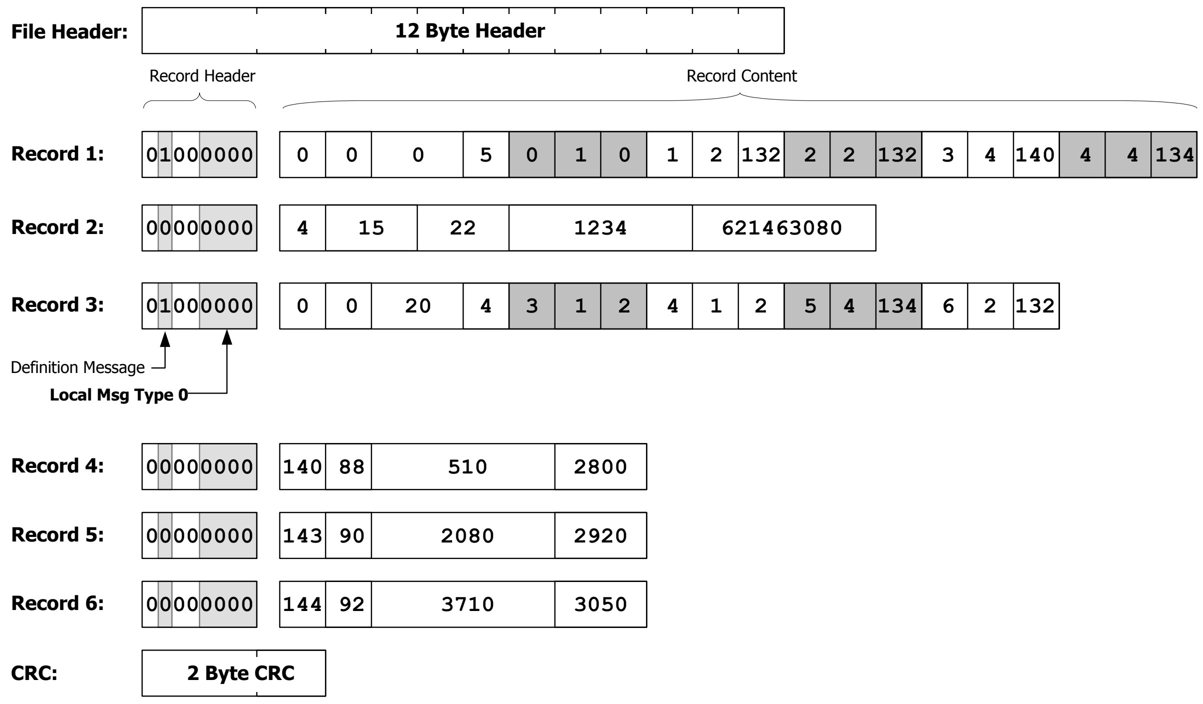


Figure 4-7. Redefining local message type within a single FIT file

5 FIT Message Conversion

Reference C, C++ and Java code for both embedded and PC conversions of FIT files are available in the provided SDK. The FIT protocol is fully backwards compatible, ensuring that devices with different versions of the FIT protocol can share files. The conversion tool handles all conversion-related issues such as differences in device architecture (big endian vs. little endian), and differences in messages between devices which have different versions of the FIT protocol.

Figure 5-1 takes the example given in section 4.3 and shows how an incoming message is encoded according to the device's implemented *Product Profile* and added to the FIT file. In this case, the data corresponding to Record 5 of the previous example, is used.

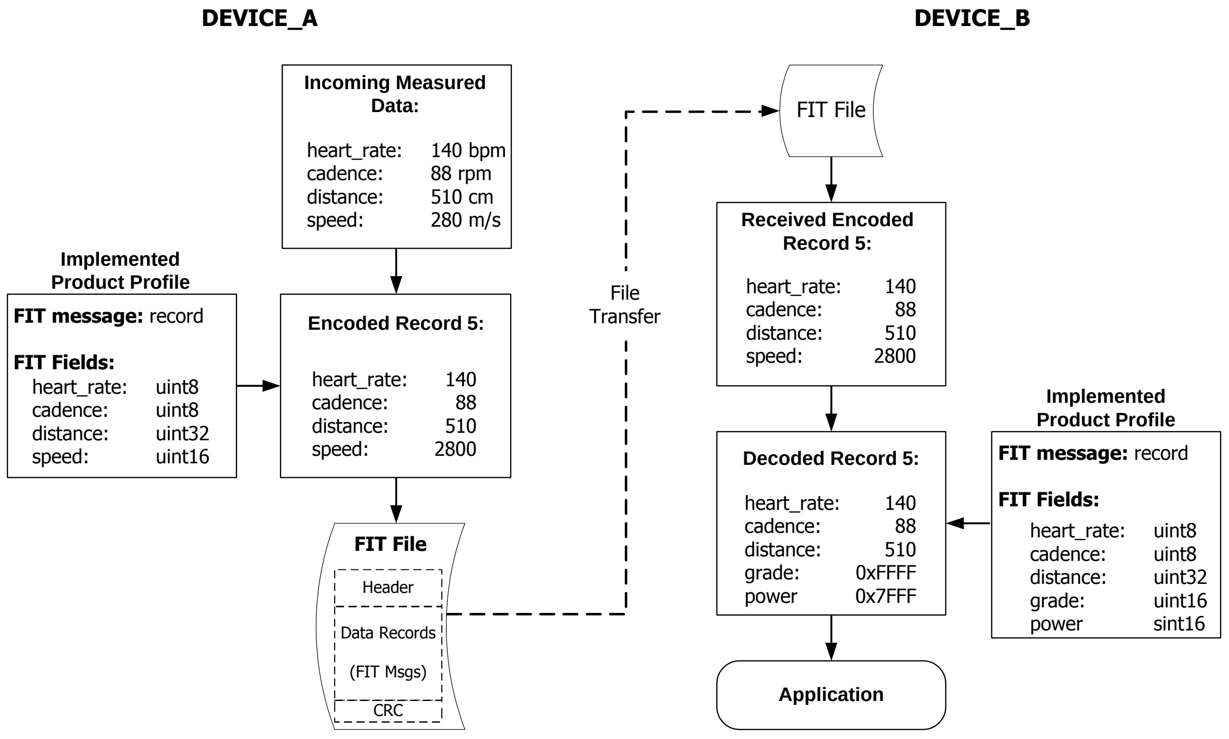


Figure 5-1 Conversion of a FIT message

In this simplified example, Device_A's implemented product profile includes the FIT "record" message and its heart_rate, cadence, distance and speed FIT fields. The incoming data is formatted and encoded according to the product profile and added to the FIT file. When all records have been added and the FIT file is complete, it is ready for transfer.

Device_B, on the other hand, has a slightly different implemented product profile that still includes the FIT "record" message; however, this profile has a different set of FIT fields defined. Device_A and Device_B both have the heart_rate, cadence and distance FIT fields, but Device_A includes speed, whereas Device_B includes grade and power data. As FIT is fully compatible across different versions of global and product FIT profiles, the protocol will automatically account for these differences.

As illustrated in Figure 5-1, Device_B receives the FIT file, and the decoder will interpret and decode the information it recognizes (i.e. heart_rate, cadence, distance), ignore data it does not recognize (i.e. speed), and populate the remaining FIT fields with invalid values according to its base type (i.e. grade and power). In this way, the FIT file is maintained and can be transferred again in its original form, unrecognized or missing data is processed by Device_B without causing errors, and the resultant information is passed in the form of a C structure or object to Device_B's application for further use.

5.1 Compatibility

The FIT protocol is designed for extensibility. The software development code provided is designed to maintain compatibility as FIT files are transferred between systems. For compatibility between systems to be maintained, the FIT profile must be strictly adhered to. There is built in flexibility for system architectures. Endian architecture is described in each message definition and automatically handled within the FIT SDK.

5.2 Common FIT File Applications

Certain applications of FIT files lead to a natural grouping of messages based on purpose. Refer to the FIT File Types Description document for more details on the common message groupings and methods for best practice of the following file types:

Table 5-1. Common FIT File Types

FIT File Type	Purpose
Settings	Describes a user’s parameters such as Age, Weight, and Height
Activity	Records data and events from an active session
Workout	Records data describing a workout’s parameters such as target rates and durations
Blood Pressure	Provides summary data from a blood pressure device
Weight	Provides summary data from a weight scale device