FIT SDK

Introductory Guide

Copyright Information and Usage Notice

Revision History

Revision	Effective Date	Description
1.0	May 2010	Initial release
1.1	March 2011	Updated License
1.2	November 2011	Updated format Added "alternate message configurations" section
1.3	October 2012	Add C# Section
1.4	December 2012	Update FitCSVTool Usage
1.5	February 2013	Updated Template Updated Usage Notice
1.6	May 2014	Added Monitoring Reader
1.7	August 2015	Updated FitCSVTool Usage
2.0	May 2016	Release for FIT 2.0
2.1	June 2016	Update FitCSVTool Usage
2.2	September 2016	Added "alternate field configurations" section
2.3	November 2016	Added SDK Requirements

Table of Contents

List of Figures

List of Tables

1 Overview of the FIT SDK

1.1 SDK Package Contents

The FIT SDK includes:

- FitGen.exe -
- config.csv -
- c\ - C code
    - examples\ - Example encode and decode applications
- cpp\ - C++ code
    - examples\ - Example encode and decode applications
- cs\ - C# code
    - Fit.dll & Fit.xml – SDK Library (must recompile if you customize the profile)
    - Dynastream\ - Source files for the C# SDK objects
        - Fit\ - General FIT SDK Objects
            - Profile\Mesgs\ - Profile Message Objects
            - Profile\Types\ - Profile Type Objects
        - Utility\ - Extensions
    - Examples\ -Example encode and decode applications, DLL Project
- java\ - Java code
    - fit.jar - .FIT java package
    - FitCSVTool.jar - Executable CSV conversion tool.
    - FitTestTool.jar - Executable test tool.
    - com\garmin\fit\ - Source code.
    - doc\ - Java API documentation.

1.2 SDK Requirements

1.2.1 Java

The Java SDK Requires:

- JDK 6
    - Must be on PATH

1.2.2 C#

The C# SDK targets .NET framework 4.5

2 Customizing C Code for a Specific Product

The C code can be customized to a product specific set of .FIT messages and fields. This allows the code to be optimized for the product to minimize RAM and ROM resource requirements and can be achieved by following these steps:

1. Modify the compile options in fit_config.h to suit the product requirements.
2. Modify config.csv to customize the .FIT profile for a product(s).
3. Run FitGen.exe to regenerate the C code with the new configuration.

2.1 Generating Code for Multiple Products

Additional columns can be added to config.csv to specify configurations for multiple products. The default product is 'SDK' which includes all possible messages and fields. Fit_product.c/h allow the product configuration to be selected at compile time by defining FIT_PRODUCT_<PRODUCT_NAME> in fit_config.h.

2.2 Data Structure Alignment

Message data is accessed directly through auto generated C structures. See FIT_*_MESG type definitions. The order of fields is optimized to minimize the structure padding requirements. The default alignment is 4 bytes and can be configured in config.csv for each product.

2.3 Selecting Messages and Fields

Messages and fields are selected by specifying the number of field elements in the product column of config.csv. A field is not included if the cell is 0 or blank. If no fields are included, then the message is also not included in the source.

For example, the following configuration includes a user profile message with a friendly name with maximum length of 16, gender, age and weight. Language is not included in the message because it is set to 0.

Table 2-1. Message Configuration

Message Name	Field Name	Product
user_profile		
	friendly_name	16
	gender	1
	age	1
	weight	1
	language	0

2.4 File Structures

Access to some files such as settings can be setup as a fixed structure of message definitions and data. Defining a fixed file structure allows efficient random access of message data. A message can be included in a file structure by adding the following option to the message row of config.csv:

f=<file type>,<number of messages in file>

If file type is "all", then the message is included in all file types. The file_id message is required in every file so it must be configured as f=all,1.

For example, the configuration shown in Table 2-2 includes 3 user profile messages in a settings file. Each of these 3 messages includes the fields: friendly name, gender, age and weight.

Table 2-2. File Configuration

Message Name	Field Name	Product
user_profile		f=settings,3
	friendly_name	16
	gender	1
	age	1
	weight	1
	language	0

The FIT_MESG_OFFSET macro can be used to compute the byte offset of a message in a file. For example, the offset of the 3rd user profile message in a settings file will be returned by:

FIT_MESG_OFFSET(user_profile_mesg, 2, USER_PROFILE_MESG_SIZE, FIT_SETTINGS_FILE)

The number of files can be specified in the "Files:" row of config.csv.

2.5 Capabilities

Message and field capabilities can be auto generated. See device file type for more information on device capabilities.

A capability is configured by added the following option to the message or field row of config.csv:

c=<file type>,<capability type>,<number of messages>

The options for capability type are: num_per_file, max_per_file, or max_per_file_type. Each message and field can have multiple capability options (for different file types).

2.6 Alternate Message Configurations

A message can be configured to have multiple definitions with different numbers of fields included. Alternate definitions must first be named by adding the following option to the message row of config.csv:

d=<alt-def 1 name>,<alt-def 2 name>,<alt-def 3 name>,…

If n names are specified, then n alternate message definitions are created **in addition** to the main message definition. Names can only include letters, numbers and the underscore character. To specify the number of elements for a particular field, the field row of config.csv must be formatted as follows:

<main-def # elements> d=<alt-def 1 # elements>,<alt-def 2 # elements>,<alt-def 3 # elements>,…

For example, the following configuration includes three user profile message definitions. The main definition includes a friendly name with maximum length of 16, gender, age and weight. The alternate definition "long_name" includes a friendly name with a maximum length of 32. The alternate definition "language_only" includes only the language field.

Table 2-3. Alternate Message Definitions

Message Name	Field Name	Product
user_profile		d=long_name,language_only
	friendly_name	16 d=32,0
	gender	1 d=0,0
	age	1 d= 0,0
	weight	1 d=0,0
	language	0 d=0,1

Alternate messages can be selected when setting up files and RAM. To set up a file with an alternate message, add the following option to the message row of config.csv:

f=<file_type>,<number of messages in file>,<alt-def name>

To set up a static copy of a message in RAM, add the following option to the message row of config.csv:

r=<number of messages in RAM>, <alt-def name>

If the alternate definition name is not specified, the main definition is used. For example, the following configuration selects the "long_name" definition for the settings file but the main definition in the device file.

Table 2-4. File Definitions

Message Name	Field Name	Product
user_profile		d=long_name,language_only f=settings,3,long_name f=device,3
	friendly_name	16 d=32,0
	gender	1 d=0,0
	age	1 d=0,0
	weight	1 d=0,0
	language	0 d=0,1

Note that functions in the SDK that access definitions and sizes through the global message number (for example, Fit_GetMesgDef) will have return values corresponding only to the *main* definition. To access alternate message definitions, use the fit_mesg_defs array using the appropriate FIT_MESG enumerator as the index.

2.7 Alternate Field Configurations

Fields can be configured to set initial values or retrieve data from RAM. To setup fields to use a constant initialization value add the following option to the field row of config.csv:

f=<file_type>,<count>,<init_value>

When fields should be pulled from a variable available in RAM add the following option to the field row of config.csv:

f=<file_type>,<count>,<init_value>,<ram_variable>

If there is a constant pointer to the RAM data instead of an accessible variable add the following option to the field row of config.csv:

f=<file_type>,<count>,<init_value>,<ram_pointer>,p

3 C# SDK

The FIT SDK provides access to the Microsoft .NET framework via a managed C# class library. The SDK is developed using Visual Studio C# Express 2008 and uses version 3.5 of the .NET framework. Earlier versions are not supported.

The SDK may be used by C# applications in the following ways:

- By including the autogenerated source files
- By referencing the FIT.DLL library

If the second option is used, the FIT.DLL must be included with the end user application.

To acquaint users with the SDK three example projects are provide in the Examples.sln:

- ClassLib: Recompiles the FIT.DLL library from the autogenerated source files. The DLL will support custom user messages but must be recompiled after any modifications are made to the profile via the FITGen tool.
- Decode: Demonstrates decoding of a FIT file and use of the Decode and Broadcaster objects and OnMesg event interfaces. This application includes the source files directly for a single .exe solution.
- Encode: Demonstrates encoding a FIT file and programmatic generation of messages. This application references the FIT.DLL.

These example projects serve as a useful starting point for working with the C# SDK. The interface is similar to the Java SDK implementation.

The SDK files are autogenerated using the FitGen tool so custom user messages can be supported.

4 Using FitCSVTool

The FitCSVTool is a java command line utility that converts FIT information stored in csv files, to binary FIT files, and vice versa. The usage is described below:

```
FIT CSV Tool - Protocol 2.0
Usage: java -jar FitCSVTool.jar <options> <file>
      -b <FIT FILE> <CSV FILE>  FIT binary to CSV.
      -c <CSV FILE> <FIT FILE>  CSV to FIT binary.
      -t Enable file verification tests.
      -d Enable debug output.
      -i Check integrity of FIT file before decoding.
      -s Show invalid fields in the CSV file.
      -se Show invalid fields in the CSV file as empty cells.
      -u Hide unknown data and report statistics on how much is hidden.
      -x Print byte values as hexadecimal.
      -pN Encode file using Protocol Version <N>. Default: 1
      --defn <MESSAGE_STRING_0,MESSAGE_STRING_1,...> Narrows down the
          definitions output to CSV. Use 'none' for no definitions
          When this option is used only the message definitions
          in the comma separated list will be written to the CSV.
          eg. --defn file_capabilities,record,file_creator
          Note: This option is only compatible with the -b option.
      --data <MESSAGE_STRING_0,MESSAGE_STRING_1,...> Narrows down the
          data output to CSV. When this option is used only the data
          in the comma separated list will be written to the csv.
          eg. --data file_capabilities,record,file_creator
          Note: This option is only compatible with the -b option.
```

Refer to the \examples directory in the FIT SDK for examples of how to correctly define the .csv file. Note, the .csv files may be created with or without specifying definition messages and local message types. If the local message type is not defined in the .csv file, the FitCSVTool will redefine each message with local message type 0. Refer to the workout .csv files for an example of defining csv's without defining the local message type; and refer to the blood pressure multiuser .csv file for an example of defining FIT messages and their respective local message types.

4.1 Message Filtering

Message filtering in the output ASCII file may be accomplished using the --defn or --data arguments and passing in a list of message names. In this way only messages of interest will be written to the .csv file simplifying analysis.

To suppress the generation of ALL data messages the special argument 'none' may be used with the –data argument. To suppress the generation of ALL definition messages the special argument 'none' may be used with the –defn argument.

In addition to the usual .csv output a 'data' file is created (*_data.csv). The header row of the data file contains a column for each message/field combination encountered in the file (messages that have been filtered are not considered). Each row contains the most recent value encountered for that field. Care should be taken to differentiate values that are repeated in the source file from values added when rows are generated as other messages are encountered.

4.2 Batch Files

The FitCSVTool can be invoked from the command line as described above or by using the provided batch files found in the Java folder. Users can "drag and drop" .FIT binary files onto the FitToCSV.bat to quickly convert them to human readable .csv format.

4.3 Invalid Fields

The FitCSVTool defaults to excluding invalid fields from the resulting CSV. It can be configured to show invalid fields with their corresponding invalid values by using –s, or to show invalid fields in the CSV but substitute an empty cell for the invalid value by using –se. These options can be helpful for maintaining consistent formatting when analyzing .csv output. The following are the differences in output when a field is invalid (in this case, the battery_status field):

Type	Local Number	Message	Field 1	Value 1	Units 1	Field 2	Value 2
Definition	0	file_id	time_created	1		type	1
Data	0	file_id	time_created	809977435		type	32
Definition	0	device_info	timestamp	1		battery_status	1
Data	0	device_info	timestamp	809977435	s		
Definition	1	monitoring	timestamp	1		cycles	1

Figure 4-1. Default CSVTool behavior

Type	Local Number	Message	Field 1	Value 1	Units 1	Field 2	Value 2
Definition	0	file_id	time_created	1		type	1
Data	0	file_id	time_created	809977435		type	32
Definition	0	device_info	timestamp	1		battery_status	1
Data	0	device_info	timestamp	809977435	s	battery_status	255
Definition	1	monitoring	timestamp	1		cycles	1

Figure 4-2. CSVTool behavior with –s option

Type	Local Number	Message	Field 1	Value 1	Units 1	Field 2	Value 2
Definition	0	file_id	time_created	1		type	1
Data	0	file_id	time_created	809977435		type	32
Definition	0	device_info	timestamp	1		battery_status	1
Data	0	device_info	timestamp	809977435	s	battery_status	
Definition	1	monitoring	timestamp	1		cycles	1

Figure 4-3. CSVTool behavior with –se option

5 MonitoringReader

The MonitoringReader class provides enhanced support for decoding monitoring messages. It offers support for differentiating accumulated values based on the activity_type, computing and populating the local_timestamp and modifying the message period.

The MonitoringReader implements the MonitoringMesgListener interface and is used in much the same way as the MesgBroadcaster. Rather than connecting directly to the MesgBroadcaster, user applications will subscribe to MonitoringMesg events from the MonitoringReader. The MonitoringReader is connected to the MesgBroadcaster by adding it as a listener. See the MonitoringReaderExample sample application for further details.

5.1 Activity Type Grouping

The MonitoringReader will differentiate accumulated values by activity type and track totals independently for cycles, distance, active calories, (total) calories and active time. An aggregate total will be calculated by summing the contributions of each activity type encountered and generate a monitoring message where activity type = 'all'. See the Monitoring A&B section in the FIT File Types document for further examples.

5.2 Local Timestamp Support

In FIT files, timestamps can be of different forms. Before the UTC time is set, system time may be logged. After the timezone is set local time can computed. If the offsets from UTC time are known, system time can be converted to UTC time and UTC times can be converted to local times.

If the utc_offset (offset between system time and UTC) is known the MonitoringReader will replace any system times encountered with the corresponding UTC time. If the timezone is known the local_timestamp field will be generated and added to monitoring messages containing a valid UTC time.

The MonitoringReader also implements the DeviceSettingsMesgListener and MonitoringInfoMesgListener interfaces. These listeners may be used to establish the system time offset or local time offset from data found in either of these messages.

5.3 Adjustable Message Period

The Monitoring Reader may be configured to generate its monitoring message event stream at a particular rate (between 1 mesg/s and 1 mesg/day), which may be different than the source stream. The output frequency may be more or less frequent than the original message stream.