FIT SDK

Introductory Guide

D00001384 Rev 1.2

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

Table of Contents

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
- 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.

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.

1. Modify compile options in fit_config.h to suit 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 a 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 by 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.

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 is always configured as f=all,1.

For example, the following configuration includes 3 user profile messages in a settings file:

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 byte offset of a message in a file. For example, the offset of the 3rd user profile in a settings file:

```
FIT_MESG_OFFSET(user_profile_mesg, 2, USER_PROFILE_MESG_SIZE, FIT_SETTINGS_FILE)
```

The number of files can 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.

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.

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.

3 Using FitCSVTool

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

```
.FIT CSV Tool 1.0.0.70
Usage: java -jar FitCSVTool.jar -b|-c <INPUT FILE> <OUTPUT 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 (also enables file verification tests).
```

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.