Upgrading EcFAT from version 2.2 to 3.0

Version 3.0.4

Table of contents

1 Introduction

EcFAT 3.0 adds a lot of new features to EcFAT 2.2 like journaling, wear-leveling and bad block management.

It is EmbCode's policy to only introduce API changes that requires code rewrite with new major versions.

The API changes included in EcFAT 3.0 are both to support new features and to make EcFAT easier to use. Most of these changes could have been done in a way to be compatible with EcFAT 2.2 but clarity and simplicity of the code are more important. We also believe that most users that upgrade to EcFAT 3.0 plan to use the new features and would change their code anyway.

This document describes how to rewrite EcFAT 2.2 code to work with EcFAT 3.0.

2 Upgrading

2.1 Change Include header files

The naming structure has been changed and includes need to change.

Change includes from:

```
#include "ECF/ECF.h"
```

to:

```
#include "EcFAT/EcFAT.h"
```

All functions still have the "ECF_" prefix.

2.2 Change included source files

Files now have the EcFAT_ prefix instead of ECF_. Also make sure you add the two new files EcFAT_Journal.c and EcFAT_WearLevel.c.

2.3 Change types from BYTE, WORD, DWORD to uint8_t, uint16_t and uint32_t where necessary

The old types BYTE, WORD and DWORD have been replaced with uint8_t, uint16_t and uint32_t. These are normally defined in stdint.h but will be defined by EcFAT if not present.

You will need to change the types in your block driver functions.

If you have any casts to BYTE, WORD and/or DWORD in your code, you need to change these as well.

2.4 Add a 4th parameter to all your ECF_ReadFile() calls

Previsouly, you had to supply the exact number of bytes to read to ECF_ReadFile(). It now supports a second mode where you specify the size of your buffer and ECF_ReadFile() reads as many bytes as possible. For this to work, an extra parameter is necessary.

To convert your old code, search for ECF_ReadFile() and add a NULL parameter in the end. If you previously had:

```
if(ECF_ReadFile(&fileHandle, data, 32) != ECFERR_SUCCESS)
   halt("Can't read file");
```

Replace it with:

```
if(ECF_ReadFile(&fileHandle, data, 32, NULL) != ECFERR_SUCCESS)
   halt("Can't read file");
```

If you supply NULL as the 4th parameter, ECF_ReadFile() will act exactly like in EcFAT 2.2 and previous versions.

See ECF_ReadFile in *EcFAT API Reference* to see how to use the 4th parameter.

2.5 Rename fnReadSector, fnWriteSector and fnGetVolumeInformation

The function pointers in the struct ECF_BlockDriver has the *m_* prefix added for consistency.

If you previously had:

```
bd.fnReadSector           = MyDriver_ReadSector;
bd.fnWriteSector          = MyDriver_WriteSector;
bd.fnGetVolumeInformation = MyDriver_GetVolumeInformation;
```

Change it to:

```
bd.m_fnReadSector           = MyDriver_ReadSector;
bd.m_fnWriteSector          = MyDriver_WriteSector;
bd.m_fnGetVolumeInformation = MyDriver_GetVolumeInformation;
```

2.6 Add *flags* parameter to block driver m_fnReadSector and m_fnWriteSector functions

The functions m_fnReadSector and m_fnWriteSector now take an extra parameter of type uint8_t called flags.

If you previously had (after changing the types):

```
ECF_ErrorCode MyDriver_ReadSector(
   struct ECF_BlockDriver *pBlockDriver,
   uint32_t sector,
   uint8_t *pData)
{
   ...
}

ECF_ErrorCode MyDriver_WriteSector(
   struct ECF_BlockDriver *pBlockDriver,
    uint32_t sector,
    uint8_t *pData)
{
   ...
}
```

You should change it to:

```
ECF_ErrorCode MyDriver_ReadSector(
   struct ECF_BlockDriver *pBlockDriver,
   uint32 t sector,
   uint8_t *pData,
   uint8_t flags)
{
   ...
}

ECF_ErrorCode MyDriver_WriteSector(
   struct ECF_BlockDriver *pBlockDriver,
    uint32_t sector,
    uint8_t *pData,
    uint8_t flags)
{
   ...
}
```

If you are not using trim, wear-leveling or bad block management, you can safely ignore the flags parameter. But if you plan to use these features, see *EcFAT Getting started guide* on what flags you need to handle.

2.7 If using TRIM: Replace fnWriteTrimmedSector

struct ECF_BlockDriver no longer contains fnWriteTrimmedSector.

EcFAT will now notify the block driver that the sector it is writing is trimmed by calling m_fnWriteSector() with the flag ECF_WRITESECTOR_IS_TRIMMED set.

2.8 Read EcFAT Getting started guide.pdf

If you wish to implement journaling. wear-leveling or bad block management support, continue reading in *EcFAT Getting started guide.pdf*.