redfoxengineering

discferret

DFI (DiscFerret Image)
File Format Manual
and Reference Guide

Revision 0.0 (2012-03-31)

License and Copyright Information

Our rationale for this is that:

1. We would like to keep the DiscFerret magnetic disc image format as an open, publicly-documented standard.
2. We wish to encourage users of the format to suggest improvements (via the DiscFerret IRC channel, mailing list and private email), rather than create their own incompatible variations of the format.

The file format itself is provided for use for any purpose, in any product or software application, irrespective of whether the end product is distributed under an open-source, public-domain, freeware, shareware or commercial license. No royalties are charged for its use, nor are any license fees (up-front or after-the-fact) required in any way shape or form.

In the event that any intellectual property owned by Red Fox Engineering is required to implement this specification, that intellectual property will be licensed under Fair, Reasonable and Non-Discriminatory terms, free of charge to the implementer, provided that the implementation meets the requirements set out in this specification.

In the event that a third party submits a Proposed Amendment to the DiscFerret format, they shall also license any required patents, copyrights or other intellectual property required for its implementation under Fair, Reasonable and Non-Discriminatory terms, free of charge to any user of this file format.

Users should refer to the file format as “the DiscFerret Image File Format”, or alternatively “the DiscFerret DFI File Format”. In file-type drop-down selection widgets, the form “DiscFerret Image File (*.dfi)” is acceptable.

The intended file extension for DiscFerret Image Files is ***dfi*** (e.g. *myFile.dfi*).

Table of Contents

1. Terminology used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. This document is available to download from http://www.ietf.org/rfc/rfc2119.txt or any site which mirrors the IETF RFC archive.

Other terminology used in this document is listed below.

Captured image
A disc image which was created by reading an existing magnetic disc.

Mastered image
A disc image which was created from an existing binary data stream, using a format description.

Remastered image
A disc image which was created by reading an existing magnetic disc, decoding the data, then re-encoding the data using knowledge of the on-disc format.

2. Introduction

This document is intended to provide software developers with the information required to create, interpret and decode DiscFerret Image (DFI) files in a manner which allows for seamless interoperability with other tools capable of reading and writing DFI image files.

The DFI image file format allows for high-efficiency storage of magnetic track images. These may have been acquired from an existing disc using a DiscFerret or similar magnetic disc analyser (*Captured* images), or they may have been created with the intention of being written back to disc (*Mastered* or *Remastered* images).

The DFI file format is based on the Electronic Arts *EA IFF 1985* standard, but with several critical modifications. Most notable of these is the replacement of the *FORM* chunk with a dedicated header derived from the *Portable Network Graphics* file format. Other extensions include:

- The four-byte *identifier* (also known as a *FOURCC*) has been extended to eight bytes to allow for future expansion. Any ASCII alphanumeric character may be used in an identifier (other ASCII character codes may also be used, but this is discouraged).
- The *chunk length* field has been extended from 32 to to 64 total bits. This means the maximum block size is around *sixteen exbibytes* (16 EiB), or 2^{64} total bytes.
- The total file size is limited only by the storage capability of the *chunk length* flag.
- Storage of meta-data inside the disc image. This includes scanned copies of disc labels (as either PNG or JPEG image data), textual descriptions of disc content and optionally data to identify the tools used to image the disc.

It is hoped that the DFI image format can be extended to suit the requirements of many disc archiving use cases. If for some reason you require a feature to be added to the DFI Standard, please feel free to contact us.

3. Primitive data types

The various abstract data types, chunks and headers which form a DiscFerret Image file are themselves composed of a number of primitive data types. These data types are listed in the table below.

Data type	Definition
BYTE	8-bit byte, unsigned
SBYTE	8-bit byte, signed
WORD	16-bit word, unsigned
SWORD	16-bit word, signed
DWORD	32-bit double word, unsigned
SDWORD	32-bit double word, signed
QWORD	64-bit quad word, unsigned
SQWORD	64-bit quad word, signed

All of the multi-byte data formats listed are stored in *big endian* (also known as *Motorola*) byte order. This means that the most significant byte is stored first, e.g. the DWORD 0x12345678 would be serialised as the four-byte sequence: *0x12*, *0x34*, *0x56*, *0x78*.

All signed types are stored in two's complement form. For example, a signed two-byte word (SWORD) would be stored as:

- Value -65536 – 0x80 0x00 (negative 65536)
- Value -1 – 0xFF 0xFF (negative one)
- Value 0 – 0x00 0x00 (zero)
- Value 1 – 0x00 0x01 (positive one)
- Value 65535 – 0x7F 0xFF (positive 65535)

Array types are denoted with an asterisk suffix. For example, *BYTE*4* would be a sequence of four unsigned bytes.

4. Overall view of a typical DFI file

- File header (see section 5)
- METADATA chunk, containing (all optional):
  - File title
  - File description
  - Item number
  - Imaging hardware description (hardware type and optional serial number)
  - Format script
  - Drive script
  - Images of e.g. disc label and packaging
- One or more TrACK chunks, each containing:
  - One TrAKINFO chunk, containing metadata specific to this track
    - Contains details about how this track was imaged, e.g. sample rate, synchronisation method and number of disc reads.
    - Contains details about this track's physical location (cylinder, head and optionally sector).
    - Also contains details of this track's *provenance*, i.e. whether it originated from a disc read or a mastering operation.
  - One coMPRESS chunk (*compression ID*), optional
    - Identifies the compression format used to store the track data.
    - If omitted, track timing data is assumed to be uncompressed.
    - Valid IDs are four characters long, defined formats are:
      - `NONE` – no compression
      - `GZIP` – gzip Deflate compression
      - `BZ2` – Bzip2 compression
      - `LZMA` – LZMA compression
      - `LZO` – LZO compression
    - Target applications are not required to support all these formats, but must support (as a minimum), `NONE`, `GZIP` and `BZ2`.
  - One trACKDAT chunk, containing track data
    - Track timing data, in the form of a sequence of 32-bit unsigned timing deltas.
  - One inDEXPOS chunk, containing positions of index pulses (optional)
    - Stores the timing positions of the index pulses present in the input stream

5. File header

The file header is placed at the beginning of the file (offset zero), and serves to identify that the file being read is a DiscFerret Image file. It also contains several blocks which allow common file transmission errors to be detected.

This section has been derived from the *Portable Network Graphics* (*PNG*) file format header, but has been modified in order to prevent DFI files from being mistaken for PNG files.

Offset	Length	Data Type	Value / Function
0	1	BYTE	**0x89** A byte with the high bit set. Used to detect transmission over 7-bit data paths.
1	3	BYTE*3	**“DFI”** DiscFerret Image
4	2	BYTE*2	**0x0D 0x0A** A DOS-style line ending. Used to detect DOS-to-UNIX line ending conversion.
6	1	BYTE	**0x1A** ASCII EOF. Stops the remainder of the file from being displayed by the TYPE command in DOS or Windows command prompts.
7	1	BYTE	**0x0A** A UNIX-style line ending. Used to detect UNIX-to-DOS line ending conversion.

Note that the header is exactly eight bytes in length. In the event that a DFI file is transferred over a 7-bit data path (e.g. a modem or USENET) without suitable encapsulation, the first byte's most significant bit will be cleared (changing the byte to *0x09*).

The next three characters are an ASCII string: “DFI”. These signify that the file is a DiscFerret Image.

A two-byte DOS-style (CR LF) line ending follows this. In the event that a DFI image file is incorrectly processed with a DOS-to-UNIX line ending conversion tool (e.g. *dos2unix* or *hd2u*), these two bytes will be converted to a single *0x0A* (LF) byte.

A single *EOF* byte follows these. This will cause the MS-DOS and Windows *TYPE* command to cease outputting data. This is intended, and allows the *TYPE* command to be used (in a rudimentary fashion) to identify that the file is a DFI image. On UNIX systems, the *file* command should be used instead.

Finally, the header is terminated with a single *0x0A* byte (ASCII LF). In the event that the file is processed by a UNIX-to-DOS line ending conversion tool (e.g. *unix2dos*), this byte will be converted into a two-byte DOS line ending (CR LF).

Applications are permitted to detect any of the above corruption, however this should only be used for the purposes of informing the user that an error has occurred. An application **MUST NOT** attempt to correct any of the above types of corruption, as this is impossible to do in a repeatable and provably-correct manner.

6. Data chunks

Each block of data stored in a DFI file is encapsulated within a Data Chunk. Each data chunk consists of a header and a payload, which in some cases may be one or more other chunks.

The header contains an EIGHTCC (Eight Character Code) which uniquely identifies the data contained within the chunk, whether it contains further *child* chunks, and whether the chunk may safely be copied into another file verbatim if the writing application does not recognise it.

6.1. EIGHTCC code descriptor

An EIGHTCC code is an eight-character ASCII string, which must be composed solely of printable characters (ideally upper and lower case letters and numbers).

The initial two characters of an EIGHTCC code must be alphabetic (their case encodes the *safe to copy* and *contains children* state flags), however other characters may be alphabetic, numeric or contain shifted symbols (e.g. %, #, ! or space).

The minimal length of an EIGHTCC is two characters, while the maximal length is eight characters. Unused bytes shall be padded with spaces. For example, “TRACK” shall be padded to create the EIGHTCC identifier “TRACK “.

The first character of an EIGHTCC code encodes the state of the *contains children* flag. If this character is upper-case, then the chunk contains children. If this character is lower-case, then the chunk does not contain children.

The second character of an EIGHTCC code encode the state of the *safe to copy* flag. If this character is upper-case, then the *safe to copy* flag is set. If the application does not recognise a chunk identifier (EIGHTCC), and the *safe to copy* flag is set, then the chunk should be copied verbatim into the output file. Otherwise, the chunk should be removed from the output file.

6.2. Chunk header format

This section has been derived from the *Portable Network Graphics* (*PNG*) file format header, but has been modified in order to prevent DFI files from being mistaken for PNG files.

Offset	Length	Data Type	Value / Function
0	8	BYTE*8	**EIGHTCC identifier code** Eight-character chunk identifier, as specified in the above section, “EIGHTCC code descriptorabove“.
7	8	QWORD	**Length** Length of the chunk payload, in bytes.
15	*n*	BYTE**n*	**Data payload** This chunk's data payload.

7. Defined chunk identifiers and payload formats

TODO