FOX-IT

YAFFS2 object headers

Identifying and parsing YAFFS2 object headers

22 July 2011

Ivo Pooters
pooters@fox-it.com

Pascal Arends
arends@fox-it.com

Steffen Moorrees
Moorrees@fox-it.com

1 About YAFFS2

Yet Another Flash File System 2 is a flash file system designed for NAND flash memory. It is the follow-up of YAFFS. YAFFS's primary strategies are:

- Single threaded model. YAFFS is locked on a per-partition basis at high level. YAFFS direct interface uses a single lock for all partitions.
- Log structured writing of data.
- Abstractions that build layered code.

YAFFS1 was originally designed for 512-byte page devices, but later also worked with 1k page devices. YAFFS2 was designed for 1k or larger page sizes, but can also be used with smaller pages using in-band tags. This document will focus on YAFFS2 and ignore the differences between YAFFS1 and YAFFS2.

In YAFFS everything that is stored in the file system is named an *object* and is uniquely identified by YAFFS by their *object ID.* An object can represent any of the following:

- File
- Directory
- Special (pipes, devices etc)
- Hard link
- Symbolic link

1.1 YAFFS2 NAND model

The memory in NAND flash is arranged in *pages*. A page is the unit of allocation and programming. In YAFFS the unit of allocation is a *chunk.* Typically a chunk will be equal to the underlying page.

A *block* is the unit of erasure. A block consists of many chunks, typically 32 to 128. Blocks can become corrupt of damaged. YAFFS implements a mechanism to detect bad blocks and mark these as bad.

YAFFS2 objectives to work with newer NAND types include:

- Zero overwrites. YAFFS2 never performs overwrites. Thus, no deletion markers or other markers are used.
- Sequential chunk writing within a block. Within a block YAFFS2 writes the chunk strictly sequential.

1.2 YAFFS file storage

YAFFS2 has a true log structure. This means, that chunks are written only sequentially and no chunk is ever written to twice.

Instead of writing data in locations specific to the files, the file system data is written in the form of a sequential log. The entries in the log are all one chunk in size and can hold one of two types of chunk:

- Data chunk: holds the actual file data.
- Object header: A descriptor for the object. This holds details (meta-data) of the object such as parent directory, object name, timestamps etc.

Each chunk has tag associated with it. The tags comprise the following relevant fields:

- Object Id: The identifier of the object the chunk belongs to.
- Chunk Id: Identifies where in the object this chunks belongs. A chunkId of zero indicates that this is an object header chunk. ChunkId == 1 indicates the first chunk.
- Byte count: The number of bytes used in this chunk. (only for data chunks)
- Sequence number: As each block is allocated, the file system's sequence number is incremented and each chunk in the block is marked with that sequence number. This provides a way to organize the chunks in chronological order.
- Shrink header marker: Used to mark object headers that are written to shrink the size of a data file.

1.3 Object header format

The YAFFS2 source code defines the struct *yaffs_obj_hdr* which tells us how the data is stored in the object header chunk.

```
struct yaffs_obj_hdr {
      enum yaffs_obj_type type;
      int parent_obj_id;
      u16 sum_no_longer_used; /* checksum of name. No longer used */
      YCHAR name[YAFFS_MAX_NAME_LENGTH + 1];
      u32 yst_mode;
      u32 yst_uid;
      u32 yst_gid;
      u32 yst_atime;
      u32 yst_mtime;
      u32 yst_ctime;
      int file_size; (object type: file)
      int equiv_id; (object type: symbolic)
      YCHAR alias[YAFFS_MAX_ALIAS_LENGTH + 1];
      u32 yst_rdev;
      u32 win_ctime[2];
      u32 win_atime[2];
      u32 win_mtime[2];
      u32 inband_shadowed_obj_id;
      u32 inband_is_shrink;
      u32 reserved[2];
      int shadows_obj;
      u32 is_shrink;};
```

Listing 1. Struct Yaffs_obj_hdr definition

Following the struct definition we can locate the information on the following locations:

	Description	Offset	size	Interpret as
0	Object type	0x0000	4 bytes	Int
1	Parent object id	0x0004	4 bytes	Int
2	Not used	0x0008	2 bytes 0xFFFF	Don't
3	Name	0x000A	256 bytes (UTF-16 mode)	String
4	Not used	0x010A	2 bytes 0xFFFF	Don't
5	File mode	0x010C	4 bytes	16 LSB are permissions (4,3,3,3)
6	User ID	0x0110	4 bytes	Int
7	Group id	0x0114	4 bytes	Int

8	Accessed timestamp	0x0118	4 bytes	Int
9	Modified timestamp	0x011C	4 bytes	Int
10	Created timestamp	0x0120	4 bytes	Int
11	File size	0x0124	4 bytes	Int
12	Equivalent ID	0x0128	4 bytes	Int
13	Alias name	0x012C	128 bytes	String
14	Rdev mode	0x01AC	4 bytes	Int
15	Win created time	0x01B0	8 bytes	Long
16	Win accessed time	0x01B8	8 bytes	Long
17	Win modified time	0x01C0	8 bytes	Long
18	Shadowed object ID	0x01C8	4 bytes	Int
19	Is shrink	0x01CC	4 bytes	Int
20	Reserved	0x01D0	8 bytes	
21	Shadowed object ID	0x01D8	4 bytes	Int
22	Is shrink	0x01DC	4 bytes	Int

2 Methodology

The goal is to extract object information from a YAFFS2 image file where the OOB bytes are missing. YAFFS2 stores the tag information such as the chunk Id and object Id in the OOB (or spare) area of NAND pages. We are presented with an image where the OOB areas are missing and thus the chunk tags are not available.

Extracting object information can be split into the following sub problems:

1. Identifying the object header chunks
2. Interpreting the object header chunks

The method described is implemented in a python program named *extractObjectHeaders.py*.

2.1 Identifying the object header chunks

A chunk can be identified as an object header by examining the chunk tags and looking for chunk Id value. If it is zero, the chunk is an object header. However, if the tags are not available, another approach is needed.

From the source code we know that the first four bytes of an object header identify the type of the object. The bytes should be interpreted as a little-endian integer and has one of the values:

- 0x00: Unknown object type
- 0x01: file
- 0x02: symbolic link
- 0x03: directory
- 0x04: hard link
- 0x05: special

The chunk is identified as object header when it meets the requirements:

1. The size of the chunk must be larger than 512 bytes.
2. Bytes 0-3 should be equal to one of the object type values
3. Bytes 8-9 should be equal to 0xFFFF (not used)
4. All bytes from offset 512 onward should be equal to 0xFF

2.2 Interpreting the object header chunks

The integers in the object header should be read little-endian. The file mode value is stored in unix f_mode format.

The result is a list of all object headers on a YAFFS2 partition. This provides an accurate view of the files and directories that were or are present in the file system. Further it provides a view of changes that have occurred to the files or directories. Every time a file is edited, read, deleted etc, a new object header is written. The object headers of a file/directory provide versioning information on that file/directory. By looking at the timestamps, information is gained on the file system activity.

It may be interesting to analyze the last object headers of each object. The last object headers can be found by examining the timestamps. We assume that each action on a file/directory will change one or more of the timestamps.

For a given object name *ON*, the last object header can be retrieved by the following algorithm:

```
Newest = 1st object header
For each object header:
  last_ts= max(mod_time, acc_time, cre_time)
  If last_ts > prev_ts:
     Newest = object header
 Prev_ts = last_ts
```

3 References

http://www.yaffs.net/

http://www.yaffs.net/files/yaffs.net/HowYaffsWorks.pdf

http://www.yaffs.net/files/yaffs.net/YaffsTuning.pdf

YAFFS2 source code