leveldb Log format

The log file contents are a sequence of 32KB blocks. The only exception is that the tail of the file may contain a partial block.

Each block consists of a sequence of records:

```
block := record* trailer?
record :=
  checksum: uint32     // crc32c of type and data[] ; little-endian
  length: uint16       // little-endian
  type: uint8          // One of FULL, FIRST, MIDDLE, LAST
  data: uint8[length]
```

A record never starts within the last six bytes of a block (since it won't fit). Any leftover bytes here form the trailer, which must consist entirely of zero bytes and must be skipped by readers.

Aside: if exactly seven bytes are left in the current block, and a new non-zero length record is added, the writer must emit a FIRST record (which contains zero bytes of user data) to fill up the trailing seven bytes of the block and then emit all of the user data in subsequent blocks.

More types may be added in the future. Some Readers may skip record types they do not understand, others may report that some data was skipped.

```
FULL == 1
FIRST == 2
MIDDLE == 3
LAST == 4
```

The FULL record contains the contents of an entire user record.

FIRST, MIDDLE, LAST are types used for user records that have been split into multiple fragments (typically because of block boundaries). FIRST is the type of the first fragment of a user record, LAST is the type of the last fragment of a user record, and MIDDLE is the type of all interior fragments of a user record.

Example: consider a sequence of user records:

```
A: length 1000
B: length 97270
C: length 8000
```

A will be stored as a FULL record in the first block.

B will be split into three fragments: first fragment occupies the rest of the first block, second fragment occupies the entirety of the second block, and the third fragment occupies a prefix of the third block. This will leave six bytes free in the third block, which will be left empty as the trailer.

C will be stored as a FULL record in the fourth block.

Some benefits over the recordio format:

1. We do not need any heuristics for resyncing - just go to next block boundary and scan. If there is a corruption, skip to the next block. As a side-benefit, we do not get confused when part of the contents of

one log file are embedded as a record inside another log file.

2. Splitting at approximate boundaries (e.g., for mapreduce) is simple: find the next block boundary and skip records until we hit a FULL or FIRST record.

3. We do not need extra buffering for large records.

Some downsides compared to recordio format:

1. No packing of tiny records. This could be fixed by adding a new record type, so it is a shortcoming of the current implementation, not necessarily the format.

2. No compression. Again, this could be fixed by adding new record types.