

google / leveldb Public[Code](#) [Issues](#) 168 [Pull requests](#) 68 [Actions](#) [Security](#) [Insights](#)[main](#) ▾

...

[leveldb](#) / [doc](#) / table_format.md**cmumford** Convert documentation to markdown. ... ✓[History](#)

1 contributor

107 lines (83 sloc) | 3.7 KB

...

leveldb File format

```
<beginning_of_file>
[data block 1]
[data block 2]
...
[data block N]
[meta block 1]
...
[meta block K]
[metaindex block]
[index block]
[Footer]          (fixed size; starts at file_size - sizeof(Footer))
<end_of_file>
```

The file contains internal pointers. Each such pointer is called a BlockHandle and contains the following information:

```
offset:  varint64
size:    varint64
```

See [varints](#) for an explanation of varint64 format.

1. The sequence of key/value pairs in the file are stored in sorted order and partitioned into a sequence of data blocks. These blocks come one after another at the beginning of the file. Each data block is formatted according to the code in `block_builder.cc`, and then optionally compressed.
2. After the data blocks we store a bunch of meta blocks. The supported meta block types are described below. More meta block types may be added in the future. Each meta block is again formatted using `block_builder.cc` and then optionally compressed.
3. A "metaindex" block. It contains one entry for every other meta block where the key is the name of the meta block and the value is a `BlockHandle` pointing to that meta block.
4. An "index" block. This block contains one entry per data block, where the key is a string \geq last key in that data block and before the first key in the successive data block. The value is the `BlockHandle` for the data block.
5. At the very end of the file is a fixed length footer that contains the `BlockHandle` of the metaindex and index blocks as well as a magic number.

```

metaindex_handle: char[p];      // Block handle for metaindex
index_handle:     char[q];      // Block handle for index
padding:          char[40-p-q]; // zeroed bytes to make fixed length
                                   // (40==2*BlockHandle::kMaxEncodedLength)
magic:            fixed64;      // == 0xdb4775248b80fb57 (little-endian)

```

"filter" Meta Block

If a `FilterPolicy` was specified when the database was opened, a filter block is stored in each table. The "metaindex" block contains an entry that maps from `filter.<N>` to the `BlockHandle` for the filter block where `<N>` is the string returned by the filter policy's `Name()` method.

The filter block stores a sequence of filters, where filter `i` contains the output of `FilterPolicy::CreateFilter()` on all keys that are stored in a block whose file offset falls within the range

```
[ i*base ... (i+1)*base-1 ]
```

Currently, "base" is 2KB. So for example, if blocks X and Y start in the range [0KB .. 2KB-1], all of the keys in X and Y will be converted to a filter by calling `FilterPolicy::CreateFilter()`, and the resulting filter will be stored as the first filter in the filter block.

The filter block is formatted as follows:

```
[filter 0]
[filter 1]
[filter 2]
...
[filter N-1]

[offset of filter 0]           : 4 bytes
[offset of filter 1]           : 4 bytes
[offset of filter 2]           : 4 bytes
...
[offset of filter N-1]         : 4 bytes

[offset of beginning of offset array] : 4 bytes
lg(base)                        : 1 byte
```

The offset array at the end of the filter block allows efficient mapping from a data block offset to the corresponding filter.

"stats" Meta Block

This meta block contains a bunch of stats. The key is the name of the statistic. The value contains the statistic.

TODO(postrelease): record following stats.

```
data size
index size
key size (uncompressed)
value size (uncompressed)
number of entries
number of data blocks
```