The Contents of inode.i_block

Depending on the type of file an inode describes, the 60 bytes of storage in `inode.i_block` can be used in different ways. In general, regular files and directories will use it for file block indexing information, and special files will use it for special purposes.

Symbolic Links

The target of a symbolic link will be stored in this field if the target string is less than 60 bytes long. Otherwise, either extents or block maps will be used to allocate data blocks to store the link target.

Direct/Indirect Block Addressing

In ext2/3, file block numbers were mapped to logical block numbers by means of an (up to) three level 1-1 block map. To find the logical block that stores a particular file block, the code would navigate through this increasingly complicated structure. Notice that there is neither a magic number nor a checksum to provide any level of confidence that the block isn't full of garbage.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\filesystems\ext4\(linux-master)(Documentation)(filesystems)(ext4)ifork.rst`, line 28)

Unknown directive type "ifconfig".

```
.. ifconfig:: builder != 'latex'

   .. include:: blockmap.rst
```

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\linux-master\Documentation\filesystems\ext4\(linux-master)(Documentation)(filesystems)(ext4)ifork.rst`, line 32)

Unknown directive type "ifconfig".

```
.. ifconfig:: builder == 'latex'

   [Table omitted because LaTeX doesn't support nested tables.]
```

Note that with this block mapping scheme, it is necessary to fill out a lot of mapping data even for a large contiguous file! This inefficiency led to the creation of the extent mapping scheme, discussed below.

Notice also that a file using this mapping scheme cannot be placed higher than 2^32 blocks.

Extent Tree

In ext4, the file to logical block map has been replaced with an extent tree. Under the old scheme, allocating a contiguous run of 1,000 blocks requires an indirect block to map all 1,000 entries; with extents, the mapping is reduced to a single `struct ext4_extent` with `ee_len = 1000`. If flex_bg is enabled, it is possible to allocate very large files with a single extent, at a considerable reduction in metadata block use, and some improvement in disk efficiency. The inode must have the extents flag (0x80000) flag set for this feature to be in use.

Extents are arranged as a tree. Each node of the tree begins with a `struct ext4_extent_header`. If the node is an interior node (`eh.eh_depth` > 0), the header is followed by `eh.eh_entries` instances of `struct ext4_extent_idx`; each of these index entries points to a block containing more nodes in the extent tree. If the node is a leaf node (`eh.eh_depth == 0`), then the header is followed by `eh.eh_entries` instances of `struct ext4_extent`; these instances point to the file's data blocks. The root node of the extent tree is stored in `inode.i_block`, which allows for the first four extents to be recorded without the use of extra metadata blocks.

The extent tree header is recorded in `struct ext4_extent_header`, which is 12 bytes long:

Offset	Size	Name	Description
0x0	__le16	eh_magic	Magic number, 0xF30A.
0x2	__le16	eh_entries	Number of valid entries following the header.
0x4	__le16	eh_max	Maximum number of entries that could follow the header.

Offset	Size	Name	Description
0x6	__le16	eh_depth	Depth of this extent node in the extent tree. 0 = this extent node points to data blocks; otherwise, this extent node points to other extent nodes. The extent tree can be at most 5 levels deep: a logical block number can be at most `2^32`, and the smallest `n` that satisfies `4*(((blocksize - 12)/12)^n) >= 2^32` is 5.
0x8	__le32	eh_generation	Generation of the tree. (Used by Lustre, but not standard ext4).

Internal nodes of the extent tree, also known as index nodes, are recorded as `struct ext4_extent_idx`, and are 12 bytes long:

Offset	Size	Name	Description
0x0	__le32	ei_block	This index node covers file blocks from 'block' onward.
0x4	__le32	ei_leaf_lo	Lower 32-bits of the block number of the extent node that is the next level lower in the tree. The tree node pointed to can be either another internal node or a leaf node, described below.
0x8	__le16	ei_leaf_hi	Upper 16-bits of the previous field.
0xA	__u16	ei_unused	

Leaf nodes of the extent tree are recorded as `struct ext4_extent`, and are also 12 bytes long:

Offset	Size	Name	Description
0x0	__le32	ee_block	First file block number that this extent covers.
0x4	__le16	ee_len	Number of blocks covered by extent. If the value of this field is <= 32768, the extent is initialized. If the value of the field is > 32768, the extent is uninitialized and the actual extent length is `ee_len` - 32768. Therefore, the maximum length of a initialized extent is 32768 blocks, and the maximum length of an uninitialized extent is 32767.
0x6	__le16	ee_start_hi	Upper 16-bits of the block number to which this extent points.
0x8	__le32	ee_start_lo	Lower 32-bits of the block number to which this extent points.

Prior to the introduction of metadata checksums, the extent header + extent entries always left at least 4 bytes of unallocated space at the end of each extent tree data block (because (2^x % 12) >= 4). Therefore, the 32-bit checksum is inserted into this space. The 4 extents in the inode do not need checksumming, since the inode is already checksummed. The checksum is calculated against the FS UUID, the inode number, the inode generation, and the entire extent block leading up to (but not including) the checksum itself.

`struct ext4_extent_tail` is 4 bytes long:

Offset	Size	Name	Description
0x0	__le32	eb_checksum	Checksum of the extent block, crc32c(uuid+inum+igeneration+extentblock)

Inline Data

If the inline data feature is enabled for the filesystem and the flag is set for the inode, it is possible that the first 60 bytes of the file data are stored here.