

# The Tabix index file format

Heng Li

Field	Description	Type	Value
<b>magic</b>	Magic string	char[4]	TBI\1
<b>n_ref</b>	# sequences	int32_t	
<b>format</b>	Format (0: generic; 1: SAM; 2: VCF)	int32_t	
<b>col_seq</b>	Column for the sequence name	int32_t	
<b>col_beg</b>	Column for the start of a region	int32_t	
<b>col_end</b>	Column for the end of a region	int32_t	
<b>meta</b>	Leading character for comment lines	int32_t	
<b>skip</b>	# lines to skip at the beginning	int32_t	
<b>l_nm</b>	Length of concatenated sequence names	int32_t	
<b>names</b>	Concatenated names, each zero terminated	char[l_nm]	
<i>List of indices (n=n_ref)</i>			
<b>n_bin</b>	# distinct bins (for the binning index)	int32_t	
<i>List of distinct bins (n=n_bin)</i>			
<b>bin</b>	Distinct bin number	uint32_t	
<b>n_chunk</b>	# chunks	int32_t	
<i>List of chunks (n=n_chunk)</i>			
<b>cnk_beg</b>	Virtual file offset of the start of the chunk	uint64_t	
<b>cnk_end</b>	Virtual file offset of the end of the chunk	uint64_t	
<b>n_intv</b>	# 16kb intervals (for the linear index)	int32_t	
<i>List of distinct intervals (n=n_intv)</i>			
<b>ioff</b>	File offset of the first record in the interval	uint64_t	
<b>n_no_coor</b> (optional)	# unmapped reads without coordinates set	uint64_t	

## Notes:

- The index file is BGZF compressed.
- All integers are little-endian.
- When (format&0x10000) is true, the coordinate follows the BED rule (i.e. half-closed-half-open and zero based); otherwise, the coordinate follows the GFF rule (closed and one based).
- For the SAM format, the end of a region equals POS plus the reference length in the alignment, inferred from CIGAR. For the VCF format, the end of a region equals POS plus the size of the deletion.
- Field col\_beg may equal col\_end, and in this case, the end of a region is end=beg+1.
- Example. For GFF, format=0, col\_seq=1, col\_beg=4, col\_end=5, meta='#' and skip=0. For BED, format=0x10000, col\_seq=1, col\_beg=2, col\_end=3, meta='#' and skip=0.
- Given a zero-based, half-closed and half-open region [beg,end), the bin number is calculated with the following C function:

```

int reg2bin(int beg, int end) {
    --end;
    if (beg>>14 == end>>14) return ((1<<15)-1)/7 + (beg>>14);
    if (beg>>17 == end>>17) return ((1<<12)-1)/7 + (beg>>17);
    if (beg>>20 == end>>20) return ((1<<9)-1)/7 + (beg>>20);
    if (beg>>23 == end>>23) return ((1<<6)-1)/7 + (beg>>23);
    if (beg>>26 == end>>26) return ((1<<3)-1)/7 + (beg>>26);
    return 0;
}

```

- The list of bins that may overlap a region [beg,end) can be obtained with the following C function.

```

#define MAX_BIN (((1<<18)-1)/7)
int reg2bins(int rbeg, int rend, uint16_t list[MAX_BIN])
{
    int i = 0, k;
    --rend;
    list[i++] = 0;
    for (k = 1 + (rbeg>>26); k <= 1 + (rend>>26); ++k) list[i++] = k;
    for (k = 9 + (rbeg>>23); k <= 9 + (rend>>23); ++k) list[i++] = k;
    for (k = 73 + (rbeg>>20); k <= 73 + (rend>>20); ++k) list[i++] = k;
    for (k = 585 + (rbeg>>17); k <= 585 + (rend>>17); ++k) list[i++] = k;
    for (k = 4681 + (rbeg>>14); k <= 4681 + (rend>>14); ++k) list[i++] = k;
    return i; // #elements in list[]
}

```