Field	Description	Type	Value
magic	Magic string	char[4]	CSI\1
min_shift	# bits for the minimal interval	int32_t	[14]
depth	Depth of the binning index	int32_t	[5]
l_aux	Length of auxiliary data	int32_t	[0]
aux	Auxiliary data	uint8_t[l_aux]	
n_ref	# reference sequences	int32_t	
List of indices (n=n_ref)			
n_bin	# distinct bins	int32_t	
List of distinct bins (n=n_bin)			
bin	Distinct bin	uint32_t	
loffset	(Virtual) file offset of the first overlapping record	uint64_t	
n_chunk	# chunks	int32_t	
List of chunks (n=n_chunk)			
chunk_beg	(Virtual) file offset of the start of the chunk	uint64_t	
chunk_end	(Virtual) file offset of the end of the chunk	uint64_t	
n_no_coor (optional)	# unmapped unplaced reads (RNAME *)	uint64_t	

The following functions generalise those given in the SAM specification for a BAI-style binning scheme. Note that in CSI *depth* refers to the scheme's maximal depth, i.e., the level number of the scheme's smallest bins, and the single-bin level spanning the entire coordinate range is level 0. Hence the BAI-style binning scheme, with six levels in total, is represented by $\mathsf{min\_shift} = 14, \mathsf{depth} = 5$.

CSI index files may contain metadata pseudo-bins for each reference sequence, with the same contents as BAI pseudo-bins. In CSI, the pseudo-bins have bin number $\mathsf{bin\_limit}(\mathsf{min\_shift}, \mathsf{depth}) + 1$.

```
/* calculate bin given an alignment covering [beg,end) (zero-based, half-close-half-open) */
int reg2bin(int64_t beg, int64_t end, int min_shift, int depth)
{
    int l, s = min_shift, t = ((1<<depth*3) - 1) / 7;
    for (--end, l = depth; l > 0; --l, s += 3, t -= 1<<l*3)
        if (beg>>s == end>>s) return t + (beg>>s);
    return 0;
}

/* calculate the list of bins that may overlap with region [beg,end) (zero-based) */
int reg2bins(int64_t beg, int64_t end, int min_shift, int depth, int *bins)
{
    int l, t, n, s = min_shift + depth*3;
    for (--end, l = n = t = 0; l <= depth; s -= 3, t += 1<<l*3, ++l) {
        int b = t + (beg>>s), e = t + (end>>s), i;
        for (i = b; i <= e; ++i) bins[n++] = i;
    }
    return n;
}

/* calculate maximum bin number -- valid bin numbers range within [0,bin_limit) */
int bin_limit(int min_shift, int depth)
{
    return ((1 << (depth+1)*3) - 1) / 7;
}
```