FCBPBRec structure

#include <Files.h>

typedef struct **FCBPBRec {**		Size	Offset	Description
struct QElem * qLink;		4	0	Address of next queue element (0=last)
short	qType;	2	4	Always ioQType (2)
short	ioTrap;	2	6	(used internally by File Manager)
Ptr	ioCmdAddr;	4	8	(used internally by File Manager)
ProcPtr	ioCompletion;	4	12	Completion routine address (see Async I/O)
OSErr	ioResult;	2	16	Error code (0=no error, 1=not done yet)
StringPtr	ioNamePtr;	4	18	Address of p-string of current filename
short	ioVRefNum;	2	22	Volume, drive, or working directory (0=all)
short	ioRefNum;	2	24	File reference number (from **PBOpen**)
short	filler;	2	26	(unused)
short	ioFCBIndx;	2	28	Index (or 0 if not indexing)
short	Filler1;	2	30	(unused, error in IM IV)
long	ioFCBFlNm;	4	32	Unique 'hard' file number
short	ioFCBFlags;	2	36	Flags (see notes) bit 8=write permission granted bit 9=resource fork bit 15=dirty
short	ioFCBStBlk;	2	38	First allocation block
long	ioFCBEOF;	4	40	Logical EOF (file size, in bytes)
long	ioFCBPLen;	4	44	Physical EOF
long	ioFCBCrPs;	4	48	Current file position (mark)
short	ioFCBVRefNum;	2	52	'Hard' volume reference number
long	ioFCBClpSiz;	4	54	File clump size (minimum allocation unit)
long	ioFCBParID;	4	58	Parent's 'Hard' Directory ID
} FCBPBRec;		62		

typedef FCBPBRec ***FCBPBPtr**;

Notes: Use this FCBPBRec structure in calls to **PBGetFCBInfo** (or to examine data starting 2 bytes beyond the address in the global variable FCBSPtr).

Note that ioVRefNum has an unusual meaning when ioFCBIndx is non-zero (i.e., when you want to index through FCBs of open files). It identifies where to search and may be a volume or working directory number, a drive number, or 0 (which indicates to index through all open files, no matter where they are).

IM IV-180 defines ioFCBFlags masks as bits 0, 1, and 7, but that is based on a 1 byte value. If your compiler's headers lay this field out as an short, then use masks of 0x0100, 0x0200, and 0x8000, respectively.