IOParam (aka **HIOParam)** structure

#include <Files.h>

This structure is used in **PB**xxx and **PBH**xxx and Device Manager calls which open, close, or perform I/O on open files.

typedef struct **IOParam** {		Size	Offset	Description
ParamBlockHeader		24	0	common fields of ParamBlock types
short	ioRefNum;	2	24	File reference or device driver reference
char	ioVersNum;	1	26	Version (use 0 for HFS) (unused by device manager)
char	ioPermssn;	1	27	Read/write permission bit (see below)
Ptr	ioMisc;	4	28	Unused
Ptr	ioBuffer;	4	32	Address of data buffer
long	ioReqCount;	4	36	Number of bytes requested
long	ioActCount;	4	40	Number of bytes read or written
short	ioPosMode;	2	44	Positioning Mode (1=absolute,2=from EOF,...) (bit 7=newline mode; bits 8-15=delimiter)
long	ioPosOffset;	4	46	Positioning delta (bytes from start,EOF,...)
short	filler1;	2	48	occurs only in HIOParam
} **IOParam**, **HIOParam**;		50		

Notes: This structure is used in Device Manager and File Manager I/O calls:

OpenSlot	**PBHOpen**	**PBRead**
PBAllocate	**PBHOpenRF**	**PBRename**
PBAllocContig	**PBHRename**	**PBSetEOF**
PBClose	**PBKillIO**	**PBSetFPos**
PBFlushFile	**PBLockRange**	**PBSetFVers**
PBGetEOF	**PBOpen**	**PBUnlockRange**
PBGetFPos	**PBOpenRF**	**PBWrite**

The most common way to use this structure is to allocate a union which is an aggregate and create and initialize a pointer to the desired data type. See ParamBlockRec for examples.

IOPermissn contains one of the following values:

fsCurPerm	(0)	currently allowed read/write permission
fsRdPerm	(1)	request for read-only permission
fsWrPerm	(2)	request for write-only permission
fsRdWrPerm	(3)	request to both read and write

IOPosMode sets bits 0 and 1 to indicate where an operation begins relative to the beginning of the physical disk media and will be one of:

fsAtMark	(0)	operation begins at present position
fsFromStart	(1)	operation begins at point offset from beginning of disk
fsFromMark	(3)	operation begins a point offset from current position

IOPosOffset gives the byte offset relative to the location indicated in IOPosMode except in cases where IOPosMode is fsAtMark (in which case the offset indicator is ignored) IOPosOffset is always a multiple of 512 bytes.

Functions vary as to which fields are required on entry and which fields are defined upon return. Some fields take on different meanings or even data types in certain cases. Refer to the function in question for full information on fields.

The most common way to use this structure is to allocate a union which is an aggregate and create and initialize a pointer to the desired data type. See ParamBlockRec or HParamBlockRec for an example.