The Video Flyer Developer's Handbook

Revision 10/19/95
Compiled and Published by Daniel Wolf
for the
NewTek Developer Support Program

The contents are separately paginated stand-alone documents.

I. Disclaimer

II. Flyer Programmer Documentation

A. Flyer Library AutoDocs
B. Flyer.i Include File
C. Flyer.h Include File
D. FlyerLib.i Include File
E. FlyerLib.h Include File
F. ClipAction Structure Detail
G. FlyerClipFmt.h Flyer File Format
H. Programming Example: Flyer Play Clip (Assembler)
I. Editor ARexx Documentation

III. Additional Documentation

A. Selecting and Configuring SCSI-2 Hard Drives for Flyer Systems
B. Using the Horita TRG-50PC with the Flyer

Disclaimer

NewTek, Inc. provides these materials at its sole discretion to aid developers of third party products which enhance and extend the capabilities of NewTek's own products. NewTek, Inc. maintains a policy of continuous improvement of its own products and therefore:

All documentation and specifications contained herein are subject to change without notice.

NewTek, Inc. provides these materials with no warranty or guarantee whatsoever.

To arrange for specific ongoing or updated support, please contact NewTek, Inc.

Flyer Library AutoDocs

by Marty Flickinger Rev 10/17/95

TABLE OF CONTENTS

flyer.library/MakeClipHead
flyer.library/MakeFlyerFile
flyer.library/ModeSelect
flyer.library/ModeSense
flyer.library/NewSequence
flyer.library/OpenReadField
flyer.library/OpenWriteField
flyer.library/PauseAction
flyer.library/PlayMode
flyer.library/PlaySequence
flyer.library/PutFrameHeader
flyer.library/Read10
flyer.library/ReadSize
flyer.library/ReadTest
flyer.library/RecordMode
flyer.library/ReqSense
flyer.library/ResetFlyer
flyer.library/SCSIinit
flyer.library/SCSIreset
flyer.library/SCSIseek
flyer.library/SetFillColor
flyer.library/SetFlooby
flyer.library/SetFlyerTime
flyer.library/SetSerDevice
flyer.library/SkipLines
flyer.library/StartClipCutList
flyer.library/StartHeadList
flyer.library/StillMode
flyer.library/TBCcontrol
flyer.library/ToasterMux
flyer.library/UnLockFlyVolList
flyer.library/VideoCompressModes
flyer.library/VideoParams
flyer.library/VoidAllHeads
flyer.library/VoidCardHeads
flyer.library/VoidClipHead
flyer.library/WaitAction
flyer.library/Write10
flyer.library/WriteTest

NAME
AbortAction - abort a previously started action

SYNOPSIS
error = AbortAction(action)
D0 A0

ULONG AbortAction(struct ClipAction *);

FUNCTION
Attempts to abort an action that was previously initiated. Does nothing if it has already finished.

If ClipAction ptr is NULL, aborts all pending operations to all Flyers.

INPUTS
action - ptr to ClipAction structure used to start the action
(or NULL to abort everything)

NAME
AddClipCut -- Add an entry to the ClipCut list

SYNOPSIS
error = AddClipCut(subclip)
D0 A0

ULONG AddClipCut(struct ClipAction *);

FUNCTION
Add another sub-clip definition to the currently open ClipCut list. The fields in this structure give specifics for this sub-clip, including:

Volume:Path/name for sub-clip
Beginning and ending field numbers
Contents (video and/or audio)

See StartClipCutList for a full description of this processing mechanism.

INPUTS
subclip - a ClipAction structure specifying the new sub-clip. The same structure may be used for each call, as all needed information is copied out of it before this function returns.

NOTES
The ClipAction fields AudStartField/VidStartField and AudFieldCount/VidFieldCount must each match, regardless of the type of clip specified to make.

SEE ALSO StartClipCutList, EndClipCutList

NAME
AddSeqClip - Add an entry to the Flyer sequence

SYNOPSIS
error = AddSeqClip(clip)
D0 A0

ULONG AddSeqClip(struct ClipAction *);

FUNCTION
Add another event for the Flyer's internal sequencer to play. Any combination of Video and Audio in/out points is supported properly, including split audio. See NewSequence for more info about the Flyer's sequencer.

INPUTS
clip - a ClipAction structure specifying the event. The same structure may be used for each call, as all needed information is copied out of it before this function returns.

SEE ALSO EndSequence, EndSequenceNew, NewSequence, PlaySequence

flyer.library/AppendFields **flyer.library/AppendFields**

NAME
AppendFields - Capture live video field(s) and append to Flyer clip

SYNOPSIS
error = AppendFields(clip)
D0 A0

ULONG AppendFields(struct ClipAction *);

FUNCTION
Captures live video field(s) and appends them to the specified Flyer video clip. Creates a new clip if it does not already exist. Grabs correct field(s) from the captured color frames so that any number of fields may be grabbed without concern for color phase.

Number of fields to record is specified in clip->ca_VidFieldCount.

This function always captures a new color frame. Also, if the number of fields specified spans more than one color frame, a new one is captured for every new field 1 needed. For example, if the current clip needs a field 4 to be appended next and this function is called with fields=3, a color frame is captured, and field 4 is appended. Then a NEW color frame is captured and fields 1 and 2 are appended. If this function is then called again with fields=3, a NEW color frame is captured and fields 3 and 4 are appended.

INPUTS
clip - a ClipAction structure specifying the clip.

NOTES
This function is not guaranteed to capture consecutive color frames, as the processing delays incurred may prohibit this. This may make capturing more than 4 fields at a time somewhat useless, yet perhaps interesting.

This function does not fully support the CAF_VIDEO and CAF_AUDIOL/R flags. It always captures video only without audio. It is also not currently capable of capturing audio only.

Be careful when appending fields onto a clip that was recorded "live", as no checking is done to see that the attributes of appended fields are correct for the rest of the clip (such as VIDEO and/or AUDIO flags). Since audio is not currently supported by this function, just be sure that fields are appended only to video-only clips (or just build new clips using this function).

flyer.library/BeginFindField **flyer.library/BeginFindField**

NAME
BeginFindField - Prepare to shuttle/jog a clip

SYNOPSIS
error = BeginFindField(clipaction)
D0 A0

ULONG BeginFindField(struct ClipAction *);

FUNCTION
Prepares Flyer for a shuttle/jog session for the named clip. Call this when the user brings up the control panel for a clip. You must prepare a ClipAction structure with the desired parameters then pass a pointer to it to this routine, which allows the Flyer to prepare itself internally. All calls to DoFindField, FindFieldAudio, and EndFindField must be passed this same structure pointer.

See also FindFieldAudio.

The fields which need setup prior to calling BeginFindField:

ca_Volume	Ptr to FlyerVolume structure
ca_Channel	Video channel to use during session
ca_Flags	CAB_VIDEO to see found frames CAB_AUDIO1 and/or 2 to hear found frames (here it is legal to set none)
ca_VolSust1	
ca_VolSust2	Volume for audio channels
fv_Path	Name of clip -- if volume name is prepended, then the next 3 fields can be left blank
fv_Board	Flyer board number
fv_SCSIchannel	SCSI channel on which clip resides (optional)
fv_SCSIdrive	Drive on SCSI channel on which clip resides (optional)

This call should always have a matching EndFindField call eventually. Do not call this twice without an intervening call to EndFindField. The result from this call should be checked. A 0 value indicates all went well, the Flyer is prepared for "DoFindField" calls. Any non-0 value indicates a failure, most likely that the named file could not be found on the specified drive (do not call EndFindField on it).

INPUTS
clipaction - specifies the name of the clip

NOTES
This call does not return until ready for DoFindField calls (???)

SEE ALSO FindFieldAudio

flyer.library/ChangeAudio **flyer.library/ChangeAudio**

NAME
ChangeAudio - Change audio parameters of a clip that is in progress

SYNOPSIS
error = ChangeAudio(clipaction)
D0 A0

ULONG ChangeAudio(struct ClipAction *);

FUNCTION
This routine allows a clip that is in progress to have its audio parameters adjusted. Simply modify the audio field(s) desired (or the CAB_AUDIO1/2 flags) in the structure used to initially start the clip, then call this function with a pointer to that structure.

INPUTS
clipaction - pointer to same structure clip was initiated with

flyer.library/CheckAction **flyer.library/CheckAction**

NAME
CheckAction - Check progress of an actions

SYNOPSIS
status = CheckAction(action)
D0 A0

ULONG CheckAction(struct ClipAction *);

FUNCTION
Checks if the operation associated with the provided ClipAction pointer has finished or not. Returns FERR_OKAY if the action has finished, or FERR_BUSY if it is still in progress. Also, starting with rev 4.08, updates the (new) ca_Status field with current status for the command. This data is only available with certain commands that use ClipAction structures.

INPUTS
action - pointer to structure that was used to issue the original command

RESULT
status = FERR_OKAY or FERR_BUSY

SEE ALSO WaitAction

flyer.library/CloseField **flyer.library/CloseField**

NAME
CloseField - Closes an OpenReadField or (Easy)OpenWriteField

SYNOPSIS
error = CloseField(action)
D0 A0

ULONG CloseField(struct ClipAction *);

FUNCTION
Closes the field which was previously opened using an OpenReadField, OpenWriteField, or EasyOpenWriteField call. In the case of a write session being closed, any unwritten data is written to the clip. Also, if less than a full field of scan lines was written, fills in remainder with fill color (usually black).

Returns FERR_FULL if not enough room left in current field to write any remaining data.

The only structure field which needs setup prior to calling CloseField: ca_FldHandle Field handle returned from successful OpenReadField or (Easy)OpenWriteField call

INPUTS
action - pointer to structure which contains field handle to close

SEE ALSO EasyOpenWriteField, OpenReadField, OpenWriteField, SetFillColor

flyer.library/CopyData **flyer.library/CopyData**

NAME
CopyData - Copy data from one location to another

SYNOPSIS
error = CopyData(srcvolume,destvolume,srcaddr,blocks,destaddr)
D0 A0 A1 D0 D1 D2

ULONG CopyData(struct FlyerVolume *,struct FlyerVolume *,ULONG,ULONG,
ULONG);

FUNCTION
Copies a range of data from one drive to another. This currently works with a start block number and a block count. The start locations may be different on the src and dest drives. This function may also be used to move data on the same drive. Handles making a copy which overlaps original on same drive.

Can also read/write to/from a tape drive by simply using -1 for the appropriate address (srcaddr or destaddr).

INPUTS
srcvolume - pointer to structure which describes a volume (used to pick specific Flyer card).
destvolume - pointer to structure which describes a volume (used to pick specific Flyer card).
srcaddr - SCSI block address on source drive
blocks - number of SCSI blocks to copy
destaddr - SCSI block address on destination drive

NOTES
May copy slower than "real-time" playback rate if copying to and from the same drive.

NAME
CPUDMA -- Transfer data between DMA memory and CPU memory

SYNOPSIS
error = CPUDMA(board,cpuptr,dmaptr,length,readflag)
D0 D0 A0 A1 D1 D2

ULONG CPUDMA(UBYTE,ULONG,ULONG,UWORD,UBYTE);

FUNCTION
This function transfers blocks of data between the Flyer's DRAM and SRAM areas. All pointers and sizes are in blocks (512 bytes). "Writes" are TO SRAM, "reads" are FROM SRAM.

CAUTION! These memory areas are highly private and dangerous to access. Use only under advisement or based on sample code.

INPUTS
board - specifies the Flyer board (0-3)
cpuptr - CPU address (block)
dmaptr - DMA address (block)
length - length of transfer (in blocks)
readflag - 0=write, 1=read

NAME
Defaults - clear given ClipAction structure(s) to default values

SYNOPSIS
error = Defaults(clipaction)
D0 A0

VOID Defaults(struct ClipAction *);

FUNCTION
Clears the given ClipAction structure to default values, as well as the attached FlyerVolume structure. See structure documentation for default values.

INPUTS
clipaction - pointer to a ClipAction structure. Will also setup FlyerVolume structure, if attached.

SEE ALSO flyer.h

NAME
DoFindField - find a specific field in clip (and view/hear it)

SYNOPSIS
error = DoFindField(clipaction)
D0 A0

ULONG DoFindField(struct ClipAction *);

FUNCTION
Finds the color frame that contains the field number specified in ca_VidStartField. If the CAB_VIDEO flag was set, the frame's video will loop on the output channel. Also, if the CAB_AUDIO1/2 flag(s) were set, the frame's audio will be heard. Currently, when the user stops in a particular spot, the color frame loops repeatedly, but the audio (if on) is heard once per new frame only. If the return value is non-0, something went wrong (such as the requested field number is out of range for the clip).

INPUTS
clipaction - same pointer as was used with BeginFindField

flyer.library/EasyOpenWriteField **flyer.library/EasyOpenWriteField**

NAME
EasyOpenWriteField - Open a clip field for writing (easy version)

SYNOPSIS
error = EasyOpenWriteField(action,field,modes,quality)
D0 A0 D0 D1 D2

ULONG EasyOpenWriteField(struct ClipAction *,ULONG,UBYTE,UBYTE);

FUNCTION
Provides an easier front-end for the more complicated OpenWriteField call.

See the description under OpenWriteField for a full description of field writing and the "action", "field", and "modes" arguments.

INPUTS
action - pointer to structure which describes a volume and the name of the clip to operate on.
field - field number of clip (starts at 0). Is a don't care with some open modes
modes - flags describing how to handle writing field
quality - a number representing the video quality
Currently supported modes, in order of decreasing video quality:
0 (D2) Best quality, worst compression
1 (D2)
2 (SN)
3 (SN)
4 (SN) Worst quality, best compression

SEE ALSO CloseField, OpenReadField, OpenWriteField, FlyerWriteLine

flyer.library/EndClipCutList **flyer.library/EndClipCutList**

NAME
EndClipCutList - Finalizes ClipCut list

SYNOPSIS
error = EndClipCutList(doit)
D0 D0

ULONG EndClipCutList(UBYTE);

FUNCTION
Finalizes a ClipCut list that was opened with StartClipCutList. If the "doit" flag is set, the processing will begin. Otherwise, the list is thrown away and the original clip remains unchanged. See StartClipCutList for a full description of this processing mechanism.

INPUTS
doit - flag: 0 aborts and discards list, 1 starts clip processing

SEE ALSO StartClipCutList, AddClipCut

flyer.library/EndFindField **flyer.library/EndFindField**

NAME
EndFindField - Cleanup after a shuttle/jog session

SYNOPSIS
error = EndFindField(clipaction)
D0 A0

ULONG EndFindField(struct ClipAction *);

FUNCTION
This call frees up resources allocated with a BeginFindField call. Call when the control panel for a clip is put away. You must pass a pointer to the same structure as was passed to BeginFindField. If CAF_USEMATTE flag is true in the ClipAction structure, this call will also put up the specified matte color on the video channel. A return value of 0 indicates all went well.

INPUTS
clipaction - same pointer as was used for entire shuttle/jog session.

NOTES
Only matte black is currently supported for CAF_USEMATTE

flyer.library/EndHeadList **flyer.library/EndHeadList**

NAME
EndHeadList - Completes list of A/B heads

SYNOPSIS
error = EndHeadList(board,makeit)
D0 D0 D1

ULONG EndHeadList(UBYTE,UBYTE);

FUNCTION
Completes list of A/B heads. If 'makeit' is 0, the list is thrown away (aborted). Otherwise, the Flyer then begins creating heads. It may use old clip heads that already exist or create new ones. Any old heads that are not used in the list are deleted.

INPUTS
board - specifies the Flyer board (0-3)
makeit - flag

flyer.library/EndSequence flyer.library/EndSequence

NAME
EndSequence - Finalizes the Flyer's internal sequence

SYNOPSIS
error = EndSequence(board,doit)
D0 D0 D1

ULONG EndSequence(UBYTE, UBYTE);

FUNCTION
Finalizes the sequence definition that was downloaded. Post processing occurs at this time, such as sequence optimization and temporary data movement. This call, therefore, may take a while to complete. When it does, the sequence is ready to play (using PlaySequence). If the "doit" flag is FALSE (0), no post processing is done, but the sequence is closed (this is required as an "abort" during sequence downloading). See NewSequence for more info on Flyer sequencing.

INPUTS
board - specifies the Flyer board (0-3)
doit - flag: 0 aborts and discards sequence, 1 starts post-processing

SEE ALSO AddSeqClip, EndSequenceNew, NewSequence, PlaySequence

flyer.library/EndSequenceNew **flyer.library/EndSequenceNew**

NAME
EndSequenceNew -- Finalizes the Flyer's internal sequence (extra features)

SYNOPSIS
error = EndSequenceNew(action, doit)
D0 A0 D0

ULONG EndSequenceNew(struct ClipAction *, UBYTE);

FUNCTION
Identical to EndSequence function, except that it uses a ClipAction structure, which specifies the Flyer board number. This allows some enhanced things during the sometimes lengthy sequence processing phase, such as the ability to be run asynchronously, ability to be aborted, and the ability for the application to obtain status during this phase.

INPUTS
clipaction - specifies the board number in the attached Volume structure
doit - flag: 0 aborts and discards sequence, 1 starts post-processing

SEE ALSO AddSeqClip, EndSequence, NewSequence, PlaySequence

flyer.library/Error2String **flyer.library/Error2String**

NAME
Error2String - Convert a Flyer error code into an error string

SYNOPSIS
desc = Error2String(error)
D0

char * Error2String(UBYTE);

FUNCTION
Gives an descriptive string for the supplied Flyer error code. Simply returns a pointer to the string. DO NOT MODIFY THE DATA IN THIS STRING.

INPUTS
error - an error code returned by a Flyer call

RESULT
desc - pointer to a static string which describes the error condition

NOTES
Does not currently convert some of the more "internal" Flyer errors, but just gives "???".

flyer.library/FindDrives **flyer.library/FindDrives**

NAME
FindDrives -- Find responding drives on SCSI bus

SYNOPSIS
error = FindDrives(flyervolume,buffer)
D0 A0 A1

ULONG FindDrives(struct FlyerVolume *,APTR);

FUNCTION
This function scans one of the Flyer's SCSI busses, looking for drives at each of the possible unit numbers. An array of data is returned which gives some rudimentary information about which unit numbers correspond to a present drive, as well as some info which is helpful in getting more detailed data (with the Inquiry command).

INPUTS
volume - pointer to structure which specifies bus to scan for drives
buffer - pointer to an 18 byte buffer which receives results

RESULT
Format of data array:

```
UBYTE DriveFlags;      // '1' bit at (1<<unit) for each drive present
UBYTE pad;
UBYTE Versions[8];     // SCSI versions of each drive found, [x] = unit
UBYTE InqLens[8];      // Inquiry lengths of each drive found, [x] = unit
```

SEE ALSO Inquiry

flyer.library/FindFieldAudio **flyer.library/FindFieldAudio**

NAME
FindFieldAudio - change audio parameters during shuttle/jog session

SYNOPSIS

```
error = FindFieldAudio(clipaction)
D0                     A0

ULONG FindFieldAudio(struct ClipAction *);
```

FUNCTION
This call allows you to change the status of the audio flag while in the middle of a shuttle session (overrides the initial audioflag specified in BeginFindField). To effect the change, modify the CAB_AUDIO flags in ca_Flags and pass the structure to this routine. Call this as many times as needed (whenever the user clicks the audio button on/off), but do not call it outside of the Begin/EndFindField pair. A return value of 0 indicates all went well.

INPUTS
clipaction - same pointer as was used for entire shuttle/jog session.

flyer.library/Firmware **flyer.library/Firmware**

NAME
Firmware - Download and run software on Flyer CPU

SYNOPSIS

```
error = Firmware(board,length,data,offset)
D0               D0    D1     A0   D2

ULONG Firmware(UBYTE,ULONG,APTR,ULONG);
```

FUNCTION
Downloads the provided binary file to the Flyer and executes it as the controlling software.

INPUTS

```
board     - specifies the Flyer board (0-3)
length    - length of data provided
offset    - offset address in shared SRAM
data      - pointer to binary data
```

flyer.library/FlyerAudioCtrl **flyer.library/FlyerAudioCtrl**

NAME
FlyerAudioCtrl - Sense/control audio rec level/aux input functions

SYNOPSIS
error = FlyerAudioCtrl(board,FlyAudCtrl,oper)
D0 D0 A0 D1

ULONG FlyerAudioCtrl(UBYTE,struct FlyAudCtrl *,UBYTE);

FUNCTION
Provides access to the Flyer's audio subsystem. This provides a means of smartly setting the input gain on record, as well as control over the Flyer's auxilliary audio inputs.

The "oper" flags describe which portions of the FlyAudCtrl structure to apply. This allows modification of individual values, and the ability to sense input levels without changing any values.

INPUTS
board - specifies the Flyer board (0-3)
FlyAudCtrl - pointer to FlyAudCtrl structure
oper - various flags indicating what kind of operation(s) to perform:
FACOF_SENSE -- update LeftSense/RightSense values
FACOF_SENSE8 -- update LeftSense/RightSense with 8-bit values
FACOF_SETGAIN -- set input gain (for recording)
FACOF_SETSRC -- set the input selector mux
FACOF_SETMIX -- set auxilliary channel mixing values
Any combination of these operations can be specified.

NOTES
The LeftSense and RightSense values from FACOF_SENSE are interpreted:
0 -- over -1.0 dB underrange
1 -- 0 to -1.0 dB underrange
2 -- 0 to 1.0 dB overrange
3 -- over 1.0 dB overrange
FACOF_SENSE8 causes Left/RightSense to contain 8 bit peak-reading values (low 8 bits truncated off)

SEE ALSO flyer.h

flyer.library/FlyerCopyClip **flyer.library/FlyerCopyClip**

NAME
FlyerCopyClip - Fast copy a flyer clip

SYNOPSIS
error = FlyerCopyClip(srcvolume,destvolume)
D0 A0 A1

ULONG FlyerCopyClip(struct FlyerVolume *,struct FlyerVolume *);

FUNCTION
Makes a copy of a Flyer clip using high speed copying (independent of Amiga host operating system). Will fail if filename is not found on the source volume or if the destination filename already exists on the destination volume. Will not create subdirectories for the destination name, so ensure entire path exists before starting copy. Source and destination volumes may be the same drive, but copying will be slower.

INPUTS
srcvolume - pointer to structure which describes the source clip's path/name and the volume on which it is found.

destvolume - pointer to structure which describes the destination path/name and the volume on which to create it. Must always contain the path/name, even if not renaming clip during copy.

NOTES
Both source and destination volumes must be attached to the same Flyer card.

SEE ALSO FlyerCopyClipNew

NAME
FlyerCopyClipNew -- Fast copy a flyer clip (w/status & abort capabilities)

SYNOPSIS
error = FlyerCopyClipNew(srcaction,destvolume)
D0 A0 A1

ULONG FlyerCopyClipNew(struct ClipAction *,struct FlyerVolume *);

FUNCTION
Identical to FlyerCopyClip function, except that it uses a ClipAction structure to specify the source, which adds the ability to run it asynchronously, ability to be aborted, and the ability to obtain status during a copy.

INPUTS
srcaction - pointer to structure which describes the source volume and clip name. "ReturnTime" and "Status" fields are also used.
destvolume - pointer to structure which describes the destination path/name and the volume on which to create it. Must always contain the path/name, even if not renaming clip during copy.

NOTES
Both source and destination volumes must be attached to the same Flyer card.

SEE ALSO FlyerCopyClip

flyer.library/FlyerCreateDir **flyer.library/FlyerCreateDir**

NAME
FlyerCreateDir -- create a sub-directory on a Flyer drive

SYNOPSIS
error = FlyerCreateDir(clipaction)
D0 A0

ULONG FlyerCreateDir(struct ClipAction *);

INPUTS
clipaction - specifies volume/path/name of the directory to create

flyer.library/FlyerDeFrag **flyer.library/FlyerDeFrag**

NAME
FlyerDeFrag - De-fragment hard drive

SYNOPSIS
error = FlyerDeFrag(volume)
D0 A0

ULONG FlyerDeFrag(struct FlyerVolume *);

FUNCTION
Begins defragmentation process on specified drive.

INPUTS
volume - pointer to structure which describes the volume to defrag

NOTES
Currently accepts no parameters and cannot be interrupted

BUGS
Reports of bugs. Unable to reproduce to date...

SEE ALSO FlyerDeFragNew

flyer.library/FlyerDeFragNew **flyer.library/FlyerDeFragNew**

NAME
FlyerDeFragNew -- De-fragment hard drive (extra features)

SYNOPSIS
error = FlyerDeFragNew(clipaction)
D0 A0

ULONG FlyerDeFragNew(struct ClipAction *);

FUNCTION
Begins defragmentation process on specified drive. This is identical to FlyerDeFrag function, except that this one uses a ClipAction structure to specifies the Flyer drive. This allows some enhanced things during defragmentation, such as the ability to be run asynchronously, ability to be aborted, and the ability for the application to obtain status while it's occurring.

INPUTS
clipaction - specifies the drive using an attached FlyerVolume structure

SEE ALSO FlyerDeFrag

flyer.library/FlyerDelete **flyer.library/FlyerDelete**

NAME
FlyerDelete -- Delete a file from a Flyer drive

SYNOPSIS
error = FlyerDelete(clipaction)
D0 A0

ULONG FlyerDelete(struct ClipAction *);

INPUTS
clipaction - specifies the path/name of the file to delete

flyer.library/FlyerDirList **flyer.library/FlyerDirList**

NAME
FlyerDirList -- return first/next entry in a directory

SYNOPSIS
error = FlyerDirList(flyervolume,grip,objinfoptr,firstflag,fsonly)
D0 A0 D0 A1 D1 D2

ULONG FlyerDirList(struct FlyerVolume *,ULONG,struct ClipInfo *,UBYTE,UBYTE);

INPUTS
volume - pointer to structure which specifies drive
grip - grip of directory
objinfoptr - Pointer to ClipInfo structure to receive info
firstflag - 0 if first call, 1 for each additional
fsonly - 0 for full information, 1 for just FileSys info

flyer.library/FlyerDriveCheck **flyer.library/FlyerDriveCheck**

NAME
FlyerDriveCheck - check if the specified drive has anything in it

SYNOPSIS
error = FlyerDriveCheck(volume)
D0 A0

ULONG FlyerDriveCheck(struct FlyerVolume *);

FUNCTION
Checks to see if the specified board/channel/drive has media loaded. Note that you must use the FVF_USENUMS flags, since the use of a volume name is not logical here.

INPUTS
volume - pointer to structure which describes a volume

RESULT
error - FERR_OKAY or FERR_VOLNOTFOUND

flyer.library/FlyerDriveInfo **flyer.library/FlyerDriveInfo**

NAME
FlyerDriveInfo - Return general information about a drive

SYNOPSIS
error = FlyerDriveInfo(volume,volinfo)
D0 A0 A1

ULONG FlyerDriveInfo(struct FlyerVolume *,struct FlyerVolInfo *);

FUNCTION
This returns general information about the drive, including the volume name, total number of blocks, number of blocks free, size of largest contiguous block, and free block size if DeFrag would be performed. If volptr is NULL, just fills in info in FlyerVolume structure only.

INPUTS
volume - pointer to structure which specifies volume
volinfo - pointer to structure to receive information about volume

flyer.library/FlyerFileClose **flyer.library/FlyerFileClose**

NAME
FlyerFileClose -- close a file

SYNOPSIS
error = FlyerFileClose(flyervolume,fileID)
D0 A0 D0

ULONG FlyerFileClose(struct FlyerVolume *,ULONG);

INPUTS
volume - pointer to same structure as passed to FlyerFileOpen
fileID - ID returned from FlyerFileOpen call

SEE ALSO FlyerFileOpen

flyer.library/FlyerFileOpen **flyer.library/FlyerFileOpen**

NAME
FlyerFileOpen -- open a file on a Flyer drive for reading/writing

SYNOPSIS
error = FlyerFileOpen(clipaction)
D0 A0

ULONG FlyerFileOpen(struct ClipAction *);

INPUTS
clipaction - specifies the name of the file to open

flyer.library/FlyerFileRead **flyer.library/FlyerFileRead**

NAME
FlyerFileRead -- read from an open Flyer file

SYNOPSIS
error = FlyerFileRead(flyervolume,fileID,size,buffer,actual)
D0 A0 D0 D1 A1 A2

ULONG FlyerFileRead(struct FlyerVolume *,ULONG,ULONG,UBYTE *,ULONG *);

INPUTS
volume - pointer to same structure as passed to FlyerFileOpen
fileID - ID returned from FlyerFileOpen call
size - number of bytes to read
buffer - pointer to buffer to receive data
actual - pointer to variable to receive count of actual bytes read

flyer.library/FlyerFileSeek **flyer.library/FlyerFileSeek**

NAME
FlyerFileSeek -- seek to a given position in a file

SYNOPSIS
error = FlyerFileSeek(flyervolume,fileID,pos,mode,posptr,oldposptr)
D0 A0 D0 D1 D2 A1 A2

ULONG FlyerFileSeek(struct FlyerVolume *,ULONG,LONG,UBYTE,LONG *,LONG *);

INPUTS
volume - pointer to same structure as passed to FlyerFileOpen
fileID - ID returned from FlyerFileOpen call
pos - requested new file position (per 'mode' below)
mode - seek mode to apply to 'pos' (FLYER_POS_xxx)
posptr - pointer to variable to receive new file position
oldposptr - pointer to variable to receive old file position

SEE ALSO flyer.h

flyer.library/FlyerFileWrite **flyer.library/FlyerFileWrite**

NAME
FlyerFileWrite -- write to an open Flyer file

SYNOPSIS
error = FlyerFileWrite(flyervolume,fileID,size,buffer,actual)
D0 A0 D0 D1 A1 A2

ULONG FlyerFileWrite(struct FlyerVolume *,ULONG,ULONG,UBYTE *,ULONG *);

INPUTS
volume - pointer to same structure as passed to FlyerFileOpen
fileID - ID returned from FlyerFileOpen call
size - number of bytes to write
buffer - pointer to buffer which contains data
actual - pointer to variable to receive count of actual bytes written

NAME
FlyerFormat - High-level format a drive with the Flyer's filesystem

SYNOPSIS
```
error = FlyerFormat(volume,name,datestamp,blocks,flags)
D0                  A0   A1       A2        D0    D1

ULONG FlyerFormat(struct FlyerVolume *,char *,struct DateStamp *,ULONG,UBYTE);
```

FUNCTION
This function does a high-level format on a drive connected to the Flyer. Not all sectors are read/write tested, so this is a "quick" format. The format procedure normally uses the entire drive, but this can be reduced to avoid using slower parts of the drive.

INPUTS
volume - NULL string, specifies drive to format
name - pointer to a null-terminated string to use for the volume name
datestamp - pointer to an AmigaDOS DateStamp structure to use as the drive's creation date
blocks - NULL for entire drive, or the number of sectors to use for video data. WARNING! This does not actually prohibit the Flyer from using the remaining space, but gives a cutoff point beyond which no video clips may be placed. Non time-critical data may eventually be placed in this "slow" region.
flags - FVIF_xxx flags to request to be applied to this drive (specifically FVIF_VIDEOREADY and FVIF_AUDIOREADY). FlyerFormat will test the speed of the drive and clear the video flag if it does not find it capable. This allows drives to be targeted as data only, data/audio, or data/audio/video.

NOTES
Do not use an fv_Path string to specify the drive. Specify a NULL string and specify the exact drive specifically with fv_SCSIdrive. This will prevent formatting the wrong drive if two exist with identical volume names!

SEE ALSO flyer.h

NAME
FlyerInputSel - Select Flyer video input sources

SYNOPSIS
```
error = FlyerInputSel(board,video,sync)
D0                    D0    D1    D2

ULONG FlyerInputSel(UBYTE,UBYTE,UBYTE);
```

FUNCTION
Specifies what video channel to use for recording to the Flyer and where to get sync.

INPUTS
board - specifies the Flyer board (0-3)
video - video source to record
- FI_Camcorder = Flyer camcorder input (TBC required)
- FI_SVHS = Flyer SVHS input (TBC required)
- FI_Toaster1 = Toaster input 1
- FI_Toaster2 = Toaster input 2
- FI_ToasterMain = Toaster Main bus output
- FI_ToasterPV = Toaster Preview bus output

sync - video source to use as a reference
- FS_ToasterMain = Toaster Main output
- FS_Toaster1 = Toaster input 1

NOTES
Changes to video or sync source using this command should be allowed to "settle" before beginning to record.

SEE ALSO FlyerTermination, ToasterMux

NAME
FlyerPlay - Play a video/audio clip

SYNOPSIS
error = FlyerPlay(clipaction)
D0 A0

ULONG FlyerPlay(struct ClipAction *);

FUNCTION
Plays a video/audio clip as specified in the structure whose pointer is given. The definition of this structure is in "Flyer.h".

This call can be made to return at different times, depending on the value of ReturnTime.

If the video channel or SCSI channel needed to accomplish this action are in use when this clip needs to begin, the PermissFlags indicate what actions the Flyer can take to free up the necessary resource(s). CAPB_STEALOURVIDEO allows the Flyer to stop a clip on the video channel specified for this new play. CAPB_KILLOTHERVIDEO allows the Flyer to stop clips on other video channels if needed to gain access to the SCSI drive for this new clip.

If the CAPB_ERRIFBUSY flag is set, this call will return with error FERR_CHANINUSE if the clip cannot be played without waiting for other resources. If this flag is not set, the Flyer will delay playback if needed while waiting for resources it needs.

When using a ReturnTime of RTT_STOPPED, you may modify/recycle the ClipAction structure once this call returns. For RTT_IMMED and RTT_STARTED, you must not modify the ClipAction structure until the clip stops or is aborted. Use AbortAction() to abort playback, and CheckProgress(), CheckAction(), or WaitAction() to determine when it's safe to reuse the structure.

If CAF_USEMATTE flag is true in the ClipAction structure, the video channel this function uses will change to the matte color specified in MatteY,MatteI,MatteQ fields when the clip finishes or is stopped.

INPUTS
clipaction - pointer to structure containing all information needed for playback, and to receive results when done.

NOTES
Only matte black is currently supported for CAF_USEMATTE

flyer.library/FlyerQuit **flyer.library/FlyerQuit**

NAME
FlyerQuit -- Stop Flyer execution, return to boot ROM

SYNOPSIS
error = FlyerQuit(board)
D0 D0

ULONG FlyerQuit(UBYTE);

INPUTS
board - specifies the Flyer board (0-3)

flyer.library/FlyerReadCalib **flyer.library/FlyerReadCalib**

NAME
FlyerReadCalib - Inspect the Flyer's calibration registers

SYNOPSIS
error = FlyerReadCalib(board,item,valueptr)
D0 D0 D1 A0

ULONG FlyerReadCalib(UBYTE,UWORD,WORD *);

FUNCTION
Reads the specified Flyer calibration register. See flyer.h for the "item" values (CALIB_xxxx). Value placed at pointer "valueptr".

INPUTS
board - specifies the Flyer board (0-3)
item - which register to change (see flyer.h)
valueptr - pointer to a UWORD to fill in with the value

SEE ALSO FlyerWriteCalib

flyer.library/FlyerReadLine **flyer.library/FlyerReadLine**

NAME
FlyerReadLine - Read a scan line from a field previously opened

SYNOPSIS
error = FlyerReadLine(action,buffer)
D0 A0 A1

ULONG FlyerReadLine(struct ClipAction *,UBYTE *);

FUNCTION
Decompresses next scan line from open field and transfers into caller's buffer (must be big enough to receive 752 bytes). NTSC line 21 is the first line read from the field, and 262 is the last. Any extra calls will fill the buffer with the fill color (usually black).

This function does software emulation of the Flyer's hardware which converts VTASC-compressed data into D2 data, including FIR filtering.

The fields which need setup prior to calling FlyerReadLine:
ca_FldHandle - Field handle from successful OpenReadField
ca_ReturnTime - RT_xxx value desired (not currently supported)

INPUTS
action - pointer to structure which contains the field handle to read from and the return time for this call.
buffer - buffer to receive composite scan line data

SEE ALSO CloseField, OpenReadField, SetFillColor, FlyerWriteLine

NAME
FlyerRecord - Record a video/audio clip

SYNOPSIS
error = FlyerRecord(clipaction)
D0 A0

ULONG FlyerRecord(struct ClipAction *);

FUNCTION
Records a video/audio clip as specified in the structure whose pointer is given. The definition of this structure is in "Flyer.h".

Except when ReturnTime = RTT_STOPPED, recording can be stopped at any time with the AbortAction() command (see below).

Do not call with both CAB_AUDIO1/2 and CAB_VIDEO flags clear, as this is nonsensical.

If the video channel or SCSI channel needed to accomplish this action are in use when this clip needs to begin, the PermissFlags indicate what actions the Flyer can take to free up the necessary resource(s). CAPB_STEALOURVIDEO allows the Flyer to stop a clip on the video channel specified for this new record. CAPB_KILLOTHERVIDEO allows the Flyer to stop clips on other video channels if needed to gain access to the SCSI drive for this new clip.

If the CAPB_ERRIFBUSY flag is set, this call will return with error FERR_CHANINUSE if the clip cannot be recorded without waiting for other resources. If this flag is not set, the Flyer will delay recording if needed while waiting for resources it needs.

When using a ReturnTime of RTT_STOPPED, you may modify/recycle the ClipAction structure once this call returns. For RTT_IMMED and RTT_STARTED, you must not modify the ClipAction structure until the clip stops or is aborted. Use AbortAction() to abort recording, and CheckProgress(), CheckAction(), or WaitAction() to determine when it's safe to reuse the structure.

INPUTS
clipaction - pointer to structure containing all information needed for recording, and to receive results when done.

flyer.library/FlyerRename **flyer.library/FlyerRename**

NAME
FlyerRename -- rename a file/dir on a Flyer drive

SYNOPSIS
error = FlyerRename(oldclip,newgrip,newname)
D0 A0 D0 A1

ULONG FlyerRename(struct ClipAction *,ULONG,char *);

INPUTS
oldclip - specifies the path/name of the file to rename
newgrip - base grip to which 'newname' is relative
newname - new path/name for file (relative to 'newgrip')

NAME
FlyerRenameDisk -- rename a Flyer drive volume

SYNOPSIS
```
error = FlyerRenameDisk(flyervolume,newname)
D0                      A0        A1

ULONG FlyerRenameDisk(struct FlyerVolume *,char *);
```

INPUTS
volume - pointer to structure which specifies drive
newname - pointer to new name string for volume

NAME
FlyerRunning -- test if Flyer firmware is downloaded and running

SYNOPSIS
```
error = FlyerRunning(board)
D0                   D0

ULONG FlyerRunning(UBYTE);
```

INPUTS
board - specifies the Flyer board (0-3)

NAME
FlyerSetBits -- set protect bits for dir/file

SYNOPSIS
```
error = FlyerSetBits(flyervolume,grip,bits)
D0                   A0        D0  D1

ULONG FlyerSetBits(struct FlyerVolume *,ULONG,ULONG);
```

INPUTS
volume - pointer to structure which specifies drive/path/name of file
grip - grip of file/dir
bits - new bits (32)

NAME
FlyerSetComment -- set comment for dir/file

SYNOPSIS
```
error = FlyerSetComment(flyervolume,grip,comment)
D0                      A0        D0  A1

ULONG FlyerSetComment(struct FlyerVolume *,ULONG,char *);
```

INPUTS
volume - pointer to structure which specifies drive/path/name of file
grip - grip of file/dir
comment - new comment string

flyer.library/FlyerSetDate **flyer.library/FlyerSetDate**

NAME
FlyerSetDate -- set date for file/dir

SYNOPSIS
error = FlyerSetDate(flyervolume,grip,days,minutes,ticks)
D0 A0 D0 D1 D2 D3

ULONG FlyerSetDate(struct FlyerVolume *,ULONG,ULONG,ULONG,ULONG);

INPUTS
volume - pointer to structure which specifies drive/path/name of file
grip - grip of file/dir
days - date: days
minutes - date: minutes
ticks - date: ticks

flyer.library/FlyerStripAudio **flyer.library/FlyerStripAudio**

NAME
FlyerStripAudio - Strip audio from a clip, make an audio-only clip

SYNOPSIS
error = FlyerStripAudio(srcvolume,destvolume)
D0 A0 A1

ULONG FlyerStripAudio(struct FlyerVolume *,struct FlyerVolume *);

FUNCTION
Creates a new clip containing only the audio from the source clip. Will fail if the source clip is not found or does not contain audio. Destination clip must not already exist on the destination volume, or an error will result. Both source and destination volumes must be attached to the same Flyer card.

INPUTS
srcvolume - pointer to structure which describes the source clip name and the volume on which it is found.
destvolume - pointer to structure which describes the destination clip name and the volume on which to place it.

flyer.library/FlyerTermination **flyer.library/FlyerTermination**

NAME
FlyerTermination - Set Flyer's video termination on/off

SYNOPSIS
error = FlyerTermination(board,flags)
D0 D0 D1

ULONG FlyerTermination(UBYTE,UBYTE);

FUNCTION
Specifies which of the Flyer's 4 video terminators to turn on.

INPUTS
board - specifies the Flyer board (0-3)
flags - One flag for each of 4 terminators (0=off, 1=on)
Bit 0 = Toaster Input 1 terminator
Bit 1 = Toaster Input 3 terminator
Bit 2 = Toaster Input 4 terminator
Bit 3 = Toaster Main terminator
Power-up default is Inputs 3 & 4 terminated, Main and Input 1 not terminated.

SEE ALSO FlyerInputSel, ToasterMux

NAME
FlyerWriteCalib - Manually set Flyer's calibration registers

SYNOPSIS
error = FlyerWriteCalib(board,item,value,saveflag)
D0 D0 D1 D2 D3

ULONG FlyerWriteCalib(UBYTE,UWORD,WORD,UBYTE);

FUNCTION
Sets the value of one of the calibration registers. See flyer.h for the "item" values (CALIB_xxxx). Starting with the Rev 4 board, these values are kept in non-volatile memory on-board the Flyer. To also save the specified value to memory, set the "saveflag" argument.

INPUTS
board - specifies the Flyer board (0-3)
item - which register to change (see flyer.h)
value - item-specific value
saveflag - 0=just use value, 1=also save to non-volatile memory

SEE ALSO FlyerReadCalib

NAME
FlyerWriteLine - Write a scan line to a field previously opened

SYNOPSIS
error = FlyerWriteLine(action,buffer)
D0 A0 A1

ULONG FlyerWriteLine(struct ClipAction *,UBYTE *);

FUNCTION
Transfers scan line data from caller's buffer, compresses it, and places in the open clip as specified in (Easy)OpenWriteField call. Expects 752 bytes from caller's buffer. NTSC line 21 is expected on the first call to this function, and 262 on the last. Any extra calls will be ignored.

This function does software emulation of the Flyer's hardware which converts D2 data into VTASC-compressed data, including FIR filtering.

The fields which need setup prior to calling FlyerWriteLine:
ca_FldHandle - Field handle from successful OpenWriteField or EasyOpenWriteField.
ca_ReturnTime - RT_xxx value desired (not currently supported)

Returns FERR_FULL if out of room in current field

INPUTS
action - pointer to structure which contains the field handle to write to and the return time for this call.
buffer - contains composite scan line data (will be modified)

NOTES
Currently modifies the data at the "buffer" pointer

SEE ALSO CloseField, EasyOpenWriteField, OpenWriteField, FlyerReadLine

NAME
GetClipInfo - get information about a specific clip

SYNOPSIS
```
error = GetClipInfo(volume,clipinfo)
D0                  A0    A1

ULONG GetClipInfo(struct FlyerVolume *,struct ClipInfo *);
```

FUNCTION
Fills in the provided ClipInfo structure with information about the specified clip. Of particuar interest are: ci_fields equals the number of fields the clip contains, ci_flags describes the type of data in the clip. See OIB_HASVIDEO and OIB_HASAUDIO in "Flyer.h". Data files (such as icons) will have neither flag set.

You MUST initialize ci_len to the value CI_sizeof before calling this function. This is to ensure that future Flyer software does not break old application software.

A non-zero return code indicates a failure (structure is not filled in). This call does not return until complete. The structure may be modified or reused in any way after it returns. This routine is useful for obtaining info about a clip which has no icon. Also retrieves any starting SMPTE time-code from the clip, which can be read using GetSMPTE.

INPUTS
volume - pointer to a FlyerVolume structure which describes the clip
clipinfo - pointer to structure to contain clip information

SEE ALSO GetSMPTE

NAME
GetFieldClock - Retrieve the Flyer's field counter

SYNOPSIS
```
error = GetFieldClock(clockptr)
D0                    A0

ULONG GetFieldClock(ULONG *);
```

FUNCTION
This returns the Flyer's internal field counter by plugging it into the provided pointer to a ULONG. If more than one Flyer exists, they are automatically sync'd together by the flyer.library. Therefore, no board number or volume name is required for this function.

INPUTS
clockptr - pointer to ULONG to receive the clock value

NAME
GetFrameHeader -- Read Frame Header structure from clip

SYNOPSIS
```
error = GetFrameHeader(action,buffer)
D0                     A0     A1

ULONG GetFrameHeader(struct ClipAction *,APTR);
```

FUNCTION
Retrieves a copy of a specific FrameHeader structure from an audio or video clip. FrameHeader chosen is the one that contains the field number specified in ca_VidStartField (even for audio clips). Places data at the structure pointed to by "buffer". If the return value is not FERR_OKAY, something went wrong (such as the clip was not found, or the requested field number is out of range).

Note: on success, clipaction->ca_StartBlk will contain the actual block number where the frame header is found, and clipaction->ca_Volume->fv_SCSIdrive will contain the actual drive number.

INPUTS
clipaction - specifies the volume/clip name and the desired field number
buffer - Pointer to caller's structure to fill in

SEE ALSO PutFrameHeader

flyer.library/GetSMPTE **flyer.library/GetSMPTE**

NAME
GetSMPTE - Return SMPTE time code information

SYNOPSIS
error = GetSMPTE(board,SMPTEinfo)
D0 D0 A0

ULONG GetSMPTE(UBYTE,struct SMPTEinfo *);

FUNCTION
This returns the last SMPTE time code information retrieved from a clip.

This is generally used after a "DoFindField" call to retrieve the SMPTE information related to that field, or after a GetClipInfo call to get the start SMPTE time for the clip.

INPUTS
board - specifies the Flyer board (0-3)
SMPTEinfo - pointer to SMPTEinfo structure to receive time code info

flyer.library/InitFlyers **flyer.library/InitFlyers**

NAME
InitFlyers - setup all attached Flyer cards

SYNOPSIS
error = InitFlyers(lock)
D0 D0

ULONG InitFlyers(BPTR);

FUNCTION
Perform setup on all Flyer boards present (programs all chips, places all video channels in play mode, playing black). Must be called from a process so that library can access DOS functions. Looks for all chip files needed in the directory which 'lock' is on.

INPUTS
lock - lock on directory in which to look for chip files

flyer.library/Inquiry **flyer.library/Inquiry**

NAME
Inquiry -- Do SCSI Inquiry command

SYNOPSIS
error = Inquiry(flyervolume,buffersize,buffer)
D0 A0 D0 A1

ULONG Inquiry(struct FlyerVolume *,UBYTE,APTR);

INPUTS
volume - pointer to structure which specifies drive
buffersize - size of buffer provided (in bytes)
buffer - pointer to buffer to receive Inquiry data

flyer.library/LocateField **flyer.library/LocateField**

NAME
LocateField - find a specific field in clip

SYNOPSIS
error = LocateField(clipaction)
D0 A0

ULONG LocateField(struct ClipAction *);

FUNCTION
Finds the color frame that contains the field number specified in ca_VidStartField. This differs from the Begin/Do/EndFindField calls in that this function just locates the field -- it does not attempt to play its video or audio. Also, the Begin/Do/End trio are designed for multiple calls on the same clip (such as for jog/shuttling), whereas this function is much simpler for just one lookup operation.

If the return value is not FERR_OKAY, something went wrong (such as the requested field number is out of range for the clip).

INPUTS
clipaction - structure that contains the following data
ca_Volume -- ptr to FlyerVolume structure that contains board, SCSI drive, and pathname for clip
ca_VidStartField -- field number to locate

RESULT
If ERR_OKAY returned, clipaction->ca_StartBlk will contain the block number of the frame header for the color frame which contains the requested field

flyer.library/LockFlyVolList **flyer.library/LockFlyVolList**

NAME
LockFlyVolList - obtain lock on internal Flyer volumes list

SYNOPSIS
ptr = LockFlyVolList()
D0

struct MinList *LockFlyVolList(void);

FUNCTION
Returns a pointer to a MinList containing the currently mounted Flyer volumes. Also locks this list so that you may safely inspect it. No modifications to the list are allowed. Be sure to release lock using UnLockFlyVolList.

A return value of 0 indicates a failure.

RESULT
ptr - pointer to a MinList of Flyer Volume Node structures (or 0 for failure)

SEE ALSO UnLockFlyVolList

flyer.library/MakeClipHead **flyer.library/MakeClipHead**

NAME
MakeClipHead - Define an A/B head for the specified clip

SYNOPSIS
error = MakeClipHead(clipaction)
D0 A0

ULONG MakeClipHead(struct ClipAction *);

FUNCTION
Define an A/B head for the specified clip. Use the ca_VidStartField, ca_AudStartField, ca_VidFieldCount, and ca_AudFieldCount entries to specify where the head should start and how long it should be.

This function can be used in two ways. If used by itself, the A/B head is made immediately. If used between StartHeadList and EndHeadList calls, the definition is just added to an internal list which will be created when EndHeadList is called with makeit=1. The second method can optimize your A/B heads and can take advantage of heads already in existence to shorten its work load. The immediate method of MakeClipHead cannot do any of these optimizations.

INPUTS
clipaction - specifies the clip and the in/out points

SEE ALSO VoidAllHeads, VoidCardHeads, VoidClipHead

flyer.library/MakeFlyerFile **flyer.library/MakeFlyerFile**

NAME
MakeFlyerFile - Create an empty file on a Flyer drive

SYNOPSIS
error = MakeFlyerFile(volume,blocks,startptr)
D0 A0 D0 A1

ULONG MakeFlyerFile(struct FlyerVolume *,ULONG,ULONG *);

FUNCTION
Creates a file of a specified size on a Flyer drive and adds it to the drive's filesystem. The start block for the file data area is returned to the caller, who may then fill the file with something useful.

Previous library versions would only create files of a size which was a multiple of 512. Starting rev 4.04, MakeFlyerFile can be used to create a file of any size, but for backward compatibility, here's how you must specify the size:

Write the byte size into the variable that 'startptr' points to Pass 'blocks' value of 0
Call MakeFlyerFile()

The value pointed to by 'startptr' has the same meaning on return as before.

INPUTS
volume - pointer to structure which describes a volume and name for new file
blocks - size of file in blocks (512 bytes each)
startptr - ptr to a ULONG to receive the start block reserved for file's data. This ULONG also contains the byte size of the file to create, providing that 'blocks' is 0.

NOTES
Errors will be returned if not enough contiguous space for the file, or if a file of the same name already exists on that drive/path.

SEE ALSO FlyerCopyClip, FlyerCopyClipNew

flyer.library/ModeSelect **flyer.library/ModeSelect**

NAME
ModeSelect -- Do SCSI ModeSelect command

SYNOPSIS
error = ModeSelect(flyervolume,buffersize,buffer,PFbyte)
D0 A0 D0 A1 D1

ULONG ModeSelect(struct FlyerVolume *,UBYTE,APTR,UBYTE);

INPUTS
volume - pointer to structure which specifies drive
buffersize - size of buffer provided (in bytes)
buffer - pointer to buffer which contains ModeSelect data
PFbyte - SCSI PageFormat byte

flyer.library/ModeSense **flyer.library/ModeSense**

NAME
ModeSense -- Do SCSI ModeSense command

SYNOPSIS
error = ModeSense(flyervolume,buffersize,page,buffer)
D0 A0 D0 D1 A1

ULONG ModeSense(struct FlyerVolume *,UBYTE,UBYTE,APTR);

INPUTS
volume - pointer to structure which specifies drive
buffersize - size of buffer provided (in bytes)
page - Mode page code to read
buffer - pointer to buffer to receive ModeSense data

flyer.library/NewSequence **flyer.library/NewSequence**

NAME
NewSequence - Prepare Flyer for a sequence download

SYNOPSIS
error = NewSequence(board)
D0 D0

ULONG NewSequence(UBYTE);

FUNCTION
Used to begin sending a sequence definition to the Flyer. Then, using other calls, each piece of the sequence is defined, the sequence is "closed", and then it may be played with one call. This is allows the Flyer to do much more complicated sequences successfully than by using FlyerPlay calls in a double-buffered fashion (which is now only supported in a limited way).

INPUTS
board - specifies the Flyer board (0-3)

SEE ALSO
AddSeqClip
EndSequence
EndSequenceNew
PlaySequence

flyer.library/OpenReadField **flyer.library/OpenReadField**

NAME
OpenReadField - Open a field from a clip for reading

SYNOPSIS
error = OpenReadField(action,field,modes)
D0 A0 D0 D1

ULONG OpenReadField(struct ClipAction *,ULONG,UBYTE);

FUNCTION
Locates specified field of named clip and prepares to decompress and transfer each scan line of the field using the FlyerReadLine call.

This function, if successful, places a valid ca_FldHandle in the ClipAction structure provided. This same structure must be used for any other calls relating to this open field, or you must manually copy the value in ca_FldHandle into the ClipAction structure you wish to use.

No compression information is required, as this information is embedded in the clips themselves.

INPUTS
action - pointer to structure which describes a volume and the name of the clip to operate on.
field - field number of clip (starts at 0)
modes - various flags
FRF_HALFLINES - allows reading the half lines. Without this flag set, half lines are skipped

SEE ALSO CloseField, EasyOpenWriteField, OpenWriteField, FlyerReadLine

flyer.library/OpenWriteField **flyer.library/OpenWriteField**

NAME
OpenWriteField - Open a field from a clip for writing

SYNOPSIS
error = OpenWriteField(action,field,modes,compinfo)
D0 A0 D0 D1 A1

ULONG OpenWriteField(struct ClipAction *,ULONG,UBYTE,struct VidCompInfo *);

FUNCTION
Prepares to transfer and compress each scan line of a field using the FlyerWriteLine call. How the new data is integrated into the clip depends on the "modes" flags specified:

FWF_NEW (field = dont care)
Writes the first field of a new clip (deletes old if exists)

FWF_APPEND (field = dont care)
Appends another field onto the clip

FWF_REWRITE (field = n)
Overwrites an existing field in the clip (must be same size or smaller)

FWF_APPEND + FWF_REWRITE (field = dont care)
Replaces the last field. Used for retrying with different compression

FWF_APPEND + FWF_REWRITE + FWF_FRAME (field = n)
Rewrite multiple fields in the last color frame (each field sequentially). Field must be in the last color frame. Used for retrying entire color frame with different compression.

FWF_REWRITE + FWF_FRAME (field = n)
Rewrite multiple fields in a color frame (each field sequentially). Used for retrying entire color frame with different compression.

FWF_HALFLINES allows writing of half lines. Without this flag set, half lines are skipped and padded.

"compinfo" points to a structure containing information about how to compress the data. If this pointer is NULL, the Flyer will default to its best algorithm.

This function, if successful, places a valid ca_FldHandle in the ClipAction structure provided. This same structure must be used for any other calls relating to this open field, or you must manually copy the value in ca_FldHandle into the ClipAction structure you wish to use. This function may fail and return FERR_FULL if not enough contiguous storage exists at the end of the clip to handle appending a field.

Also, field writing may fail if the data produced is too large for the hardware to play. An FERR_FULL error from FlyerWriteLine indicates that the field needs to be compressed harder in order to fit. If this happens, the field should be closed and reopened using a different level of compression or algorithm. Also set the FWF_REWRITE mode flag to indicate to replace the previous data.

When replacing fields in the middle of a clip, the compressed data must be the same size or smaller, as no space insertion is currently supported. If an FERR_FULL occurs in this case, you must either retry with a tighter compression method or write the original field data back into the clip. Otherwise, this field will flash unpredictable data near the bottom when the clip is played back.

Always creates clips with integral color frames regardless of how many fields are written. If a clip is left with less than a full color frame at the end, the remaining fields in the color frame are temporariliy padded with NTSC black. These pad fields are automatically replaced when new fields are appended.

INPUTS
action - pointer to structure which describes a volume and the name of the clip to operate on.
field - field number of clip (starts at 0). Is a don't care with some open modes (see below)
modes - flags describing how to handle writing field
FWF_NEW - Erase existing clip (if any), start new clip
FWF_APPEND - Append field to clip
FWF_REWRITE - Re-write over field
FWF_FRAME - Re-write field (must redo all following fields in the same color frame)
compinfo - pointer to a VidCompInfo structure (or null for defaults)

NOTES
Replacing fields in the middle of clips not fully tested

SEE ALSO CloseField, EasyOpenWriteField, OpenReadField, FlyerWriteLine

flyer.library/PauseAction **flyer.library/PauseAction**

NAME
PauseAction - pause/resume a previously started action

SYNOPSIS
error = PauseAction(action,pauseflag)
D0 A0 D0

ULONG PauseAction(struct ClipAction *,UBYTE pauseflag);

FUNCTION
Pauses or resumes a Flyer action that has been previously started. Provide a pointer to the ClipAction structure used to start the action. No error occurs if action is already in the state specified (already paused, for example). AbortAction can be used to terminate a paused action. Does nothing if the action has already finished.

INPUTS
action - ptr to ClipAction structure used to start the action
pauseflag - 1 to pause, 0 to resume

NOTES
Currently works only with FlyerRecord actions

SEE ALSO AbortAction

flyer.library/PlayMode **flyer.library/PlayMode**

NAME
PlayMode - Ready Flyer for playback

SYNOPSIS
error = PlayMode(board)
D0 D0

ULONG PlayMode(UBYTE);

FUNCTION
Readies the Flyer for playback. This takes about 1/2 second. Return value indicates success (0) or the error code on failure

INPUTS
board - specifies the Flyer board (0-3)

NOTES
You must currently ensure that no playing or recording is occurring before calling this function

NAME
PlaySequence - Play the Flyer's internal sequence

SYNOPSIS
error = PlaySequence(board,basetime)
D0 D0 D1

ULONG PlaySequence(UBYTE, ULONG);

FUNCTION
Starts the sequence playing that was previously downloaded to the Flyer. "basetime" is the time (on the Flyer's clock) to begin. All components in the sequence are relative to this start time.

This call returns immediately so that takes, effects may be done synchronous with the sequence. No other interaction with the Flyer is required (or recommended) for the sequence to play, other than aborting the sequence early (with AbortSequence).

See NewSequence for more info on Flyer sequencing.

INPUTS
board - specifies the Flyer board (0-3)

SEE ALSO AddSeqClip, EndSequence, EndSequenceNew, NewSequence

flyer.library/PutFrameHeader **flyer.library/PutFrameHeader**

NAME
PutFrameHeader -- Write Frame Header structure back to clip

SYNOPSIS
error = PutFrameHeader(action,buffer)
D0 A0 A1

ULONG PutFrameHeader(struct ClipAction *,APTR);

FUNCTION
Replaces a specific FrameHeader structure in an audio or video clip with the data structure provided. FrameHeader chosen is the one that contains the field number specified in ca_VidStartField (even for audio clips).

If the return value is not FERR_OKAY, something went wrong (such as the clip was not found, or the requested field number is out of range).

CAUTION! This function is intended to be used to read/modify/write a Frame Header (using GetFrameHeader). It is dangerous if not stupid to hand-craft a header from scratch and plug it in. Doing so is very difficult and is bound to cause problems. Also, be very cautious when modifying data in this structure. The only thing that's safe/useful to modify is the SerData buffer and associated control values. All else is toxic, flammable, noxious, etc.

INPUTS
clipaction - specifies the volume/clip name and the desired field number
buffer - Pointer to caller's structure to fill in

SEE ALSO GetFrameHeader

flyer.library/Read10 **flyer.library/Read10**

NAME
Read10 -- Transfer data from SCSI drive to DMA memory

SYNOPSIS
error = Read10(action,blocks,lba,buffer)
D0 A0 D0 D1 D2

ULONG Read10(struct ClipAction *,WORD,ULONG,ULONG);

INPUTS
clipaction - specifies the volume and return method
blocks - blocks to transfer
lba - starting lba
buffer - DMA buffer start (block) to receive data

flyer.library/ReadSize **flyer.library/ReadSize**

NAME
ReadSize -- Read SCSI drive capacity

SYNOPSIS
error = ReadSize(flyervolume,countptr,lengthptr)
D0 A0 A1 A2

ULONG ReadSize(struct FlyerVolume *,ULONG *,ULONG *);

INPUTS
volume - pointer to structure which specifies drive
countptr - pointer to ULONG to receive drive size in blocks
lengthptr - pointer to ULONG to receive logical block size (bytes)

flyer.library/ReadTest **flyer.library/ReadTest**

NAME
ReadTest -- Do a read speed test on a Flyer SCSI drive

SYNOPSIS
error = ReadTest(flyervolume,blocks,repeat,lba,dblflag)
D0 A0 D0 D1 D2 D3

ULONG ReadTest(struct FlyerVolume *,ULONG,ULONG,ULONG,UBYTE);

INPUTS
volume - pointer to structure which specifies drive
blocks - size of each transfer (in blocks)
repeat - number of transfers to perform
lba - starting lba on drive
dblflag - 0=simple test, 1=double-buffered test

flyer.library/RecordMode **flyer.library/RecordMode**

NAME
RecordMode - Ready Flyer for recording

SYNOPSIS
error = RecordMode(board)
D0 D0

ULONG RecordMode(UBYTE);

FUNCTION
Readies the Flyer for recording. This takes about 1/2 second. Return value indicates success (0) or the error code on failure

INPUTS
board - specifies the Flyer board (0-3)

NOTES
You must currently ensure that no playing or recording is occurring before calling this function

flyer.library/ReqSense **flyer.library/ReqSense**

NAME
ReqSense -- Do SCSI RequestSense command

SYNOPSIS
error = ReqSense(flyervolume,buffersize,buffer)
D0 A0 D0 A1

ULONG ReqSense(struct FlyerVolume *,UBYTE,APTR);

INPUTS
volume - pointer to structure which specifies drive
buffersize - size of data buffer provided (in bytes)
buffer - pointer to buffer to receive Sense data

flyer.library/ResetFlyer **flyer.library/ResetFlyer**

NAME
ResetFlyer -- Reset Flyer to known state

SYNOPSIS
error = ResetFlyer(board,flags)
D0 D0 D1

ULONG ResetFlyer(UBYTE,ULONG);

INPUTS
board - specifies the Flyer board (0-3)
flags - misc flags (unused)

flyer.library/SCSIinit **flyer.library/SCSIinit**

NAME
SCSIinit -- Test and Initialize SCSI bus on Flyer

SYNOPSIS
error = SCSIinit(flyervolume)
D0 A0

ULONG SCSIinit(struct FlyerVolume *);

INPUTS
volume - pointer to structure which specifies bus to init

NOTES
Set v_SCSIdrive to the SCSI bus number x 8. Also set 'FVF_USENUMS' in v_Flags

flyer.library/SCSIreset **flyer.library/SCSIreset**

NAME
SCSIreset -- Hardware reset all SCSI busses on Flyer

SYNOPSIS
error = SCSIreset(board)
D0 D0

ULONG SCSIreset(UBYTE);

INPUTS
board - specifies the Flyer board (0-3)

flyer.library/SCSIseek **flyer.library/SCSIseek**

NAME
SCSIseek -- Do SCSI seek command

SYNOPSIS
error = SCSIseek(flyervolume,lba)
D0 A0 D0

ULONG SCSIseek(struct FlyerVolume *,ULONG);

INPUTS
volume - pointer to structure which specifies drive
lba - lba to which to seek

flyer.library/SetFillColor **flyer.library/SetFillColor**

NAME
SetFillColor - set fill color to use for blank video

SYNOPSIS
error = SetFillColor(action)
D0 A0

ULONG SetFillColor(struct ClipAction *);

FUNCTION
Sets the specified Matte color as the fill color to use for blank video, such as when skipping lines with SkipLines or closing the write before all scan lines are transferred.

This color remains valid for the context of this field only. Defaults to black when a new field is opened.

The fields which need setup prior to calling SetFillColor:
ca_FldHandle - Field handle returned from successful OpenReadField or (Easy)OpenWriteField call
ca_ReturnTime - RT_xxx value desired (not currently supported)
ca_MatteY - Luminance value
ca_MatteI - Signed I value
ca_MatteQ - Signed Q value

INPUTS
action - pointer to structure which contains:
The field handle with which to associate this fill color
The fill color (in YIQ color space)
The return time for this call

SEE ALSO CloseField, EasyOpenWriteField, OpenWriteField, FlyerWriteLine

NAME
SetFlooby - used to set various Flyer internal values

SYNOPSIS
error = SetFlooby(board,chan,item,value)
D0 D0 D1 D2 D3

ULONG SetFlooby(UBYTE,UBYTE,UBYTE,ULONG);

FUNCTION
Selectively changes Flyer internal parameters by specifying the parameter number and its new value.

INPUTS
board - specifies the Flyer board (0-3)
chan - video channel (0 or 1)
item - the parameter number to change
value - the value to assign to the parameter

NOTES
All parameters are currently private. Name derived from the term "FloobyDust"

NAME
SetFlyerTime - sets the Flyer's internal clock to a preset date/time

SYNOPSIS
error = SetFlyerTime(datestamp)
D0 A0

ULONG SetFlyerTime(struct DateStamp *);

FUNCTION
Sets the internal real-time clock of all attached Flyers to the date and time specified in the structure whose pointer is given. They maintain this date/time for the purpose of date-stamping files.

INPUTS
datestamp - pointer to an AmigaDOS DateStamp structure

NAME
SetSerDevice - Select type of device attached to a Flyer serial port

SYNOPSIS
error = SetSerDevice(board,port,type,device)
D0 D0 D1 D2 D3

ULONG SetSerDevice(UBYTE,UBYTE,UBYTE,UBYTE);

FUNCTION
Specifies the type and model of device which the user wishes to attach to one of the Flyer's two serial ports. These ports can be used for a variety of things such as SMPTE read, SMPTE write, MIDI, serial control, etc. The Flyer will take care of details such as the baud rate, format conversion, etc. for all devices defined in Flyer.h

INPUTS
board - specifies the Flyer board (0-3)
port - specifies the serial port (0 = A, 1 = B)
type - type of serial device (SERDEVTYPE_xxx in Flyer.h)
device - device model (SERDEV_xxx in Flyer.h)

NAME
SkipLines - Seek past scan lines in a field previously opened

SYNOPSIS
error = SkipLines(action,lines)
D0 A0 D0

ULONG SkipLines(struct ClipAction *,UWORD);

FUNCTION
Seeks past a number of scan lines in a field previously opened. If opened for reading, skips over unwanted scan lines. If opened for writing, fills skipped lines with fillcolor (usually black). Returns FERR_FULL if out of room in current field when writing.

The fields which need setup prior to calling SkipLines:
ca_FldHandle - Field handle returned from successful OpenReadField or (Easy)OpenWriteField call
ca_ReturnTime - RT_xxx value desired (not currently supported)

INPUTS
action - pointer to structure which contains the field handle to work with and the return time for this call.
lines - number of scan lines to skip

SEE ALSO CloseField, EasyOpenWriteField, OpenReadField, OpenWriteField, FlyerReadLine, SetFillColor, FlyerWriteLine

NAME
StartClipCutList - Prepares to perform clip cutting

SYNOPSIS
error = StartClipCutList(clip,flags)
D0 A0 D0

ULONG StartClipCutList(struct ClipAction *,UBYTE);

FUNCTION
Used to begin clip cutting and processing for the clip specified. After opening the list with this function, use AddClipCut to make each subclip definition, then close the list using EndClipCutList.

Two major types of processing can currently be accomplished using this mechanism: destructive and non-destructive. The destructive processing will make the listed sub-clips and delete any unused parts of the original, doing a regional de-frag operation so as to not fragment the drive. The non-destructive operation leaves the original intact and makes new sub-clips.

INPUTS
clip - specifies the master clip from which to make sub-clip(s)
flags - specifies the type of processing (see Flyer.h CCL_xxx flags)

NOTES
Currently only one ClipCut list may be open at a time.

SEE ALSO AddClipCut, EndClipCutList

NAME
StartHeadList - Prepares Flyer for list of A/B heads

SYNOPSIS
error = StartHeadList(board)
D0 D0

ULONG StartHeadList(UBYTE);

FUNCTION
Prepares specified Flyer to compose a list of A/B heads. Create a list like this when opening an existing project. This will be more efficient than just submitting head definitions one at a time, because it allows the Flyer to do some optimizations.

INPUTS
board - specifies the Flyer board (0-3)

flyer.library/StillMode **flyer.library/StillMode**

NAME
StillMode - Set video looping method for video channel

SYNOPSIS
error = StillMode(board,chan,mode)
D0 D0 D1 D2

ULONG StillMode(UBYTE,UBYTE,UBYTE);

FUNCTION
Used to specify the type of video looping to use on stilled video. The default at power-up is MODE_FRAME.

INPUTS
volume - pointer to structure which describes a volume (used to pick specific Flyer card).
chan - video channel (0 or 1)
mode - video looping mode:
MODE_FIELD - loops a single field of video
MODE_PAIR - loops an interlaced pair of video fields
MODE_FRAME - loops an entire color frame (default)

flyer.library/TBCcontrol **flyer.library/TBCcontrol**

NAME
TBCcontrol - Sense/control TBC functions

SYNOPSIS
error = TBCcontrol(board,TBCctrl,oper)
D0 D0 A0 D1

ULONG TBCcontrol(UBYTE,struct TBCctrl *,UBYTE);

FUNCTION
Provides access to the (optional) Flyer TBC module.

The "oper" flags describe which portions of the TBCctrl structure to apply. This allows somewhat simplified use of this command without always needing to set all values for each call, as well as the ability to check the TBC status flags without modifying anything.

To determine if the TBC module is present, use this function setting only the TBCOF_STATUS oper flag, then check the "status" flags returned for TBCSF_MODULE to indicate that one was detected.

INPUTS
board - specifies the Flyer board (0-3)
TBCctrl - pointer to TBCctrl structure
oper - various flags indicating what kind of operation(s) to perform:
TBCOF_STATUS -- get status flags
TBCOF_MODES -- set modes, flags, and muxes
TBCOF_ADJUST -- set adjustment values
Any combination of these operations can be specified.

NOTES
If no TBC module is present, an error will be reported if this command is used for anything except to get status

SEE ALSO FlyerInputSel, flyer.h

flyer.library/ToasterMux **flyer.library/ToasterMux**

NAME
ToasterMux - Set Flyer/Toaster multiplex switches

SYNOPSIS
error = ToasterMux(board,input3,input4,preview)
D0 D0 D1 D2 D3

ULONG ToasterMux(UBYTE,UBYTE,UBYTE,UBYTE);

FUNCTION
Controls how the Flyer interacts with the Toaster's inputs 3 and 4 and preview output.

INPUTS
board - specifies the Flyer board (0-3)
input3 - video source fed to switcher input 3
0 = Toaster input 3
1 = Flyer video output (channel 0)
input4 - video source fed to switcher input 4
0 = Toaster input 4
1 = Flyer video output (channel 1)
preview - video fed to preview output
0 = Toaster preview bus
1 = Flyer camcorder input

SEE ALSO FlyerInputSel, FlyerTermination

flyer.library/UnLockFlyVolList **flyer.library/UnLockFlyVolList**

NAME
UnLockFlyVolList - release lock on Flyer volumes list

SYNOPSIS
error = UnLockFlyVolList(list)
D0 A0

ULONG UnLockFlyVolList(struct MinList *);

FUNCTION
Releases the lock obtained using LockFlyVolList. Like most other library functions, a return value of FERR_OKAY indicates success.

INPUTS
list - pointer to list (previously obtained with LockFlyVolList)

SEE ALSO LockFlyVolList

flyer.library/VideoCompressModes **flyer.library/VideoCompressModes**

NAME
VideoCompressModes - set video compression modes and strategy

SYNOPSIS
error = VideoCompressModes(board,bestmode,worstmode,strategy)
D0 D0 D1 D2 D3

ULONG VideoCompressModes(UBYTE,UBYTE,UBYTE,UBYTE);

FUNCTION
Sets the range of video compression qualities that the Flyer may use when recording video. The default is all modes. But this may be pared down by narrowing this range or one specific mode may be forced.

Strategy picks the strategy the Flyer should use for auto-switching between modes. The only supported value currently is 0, which uses compressed data size to switch modes.

INPUTS
board - specifies the Flyer board (0-3)
bestmode - specifies the best video compression quality mode to use
worstmode - specifies the worst video compression quality mode to use
Currently supported modes, in order of decreasing video quality:
0 (D2) Best quality, worst compression
1 (D2)
2 (SN)
3 (SN)
4 (SN) Worst quality, best compression
strategy - always 0 for now (size based strategy)

flyer.library/VideoParams **flyer.library/VideoParams**

NAME
VideoParams - set video compression parameters

SYNOPSIS
err=VideoParams(board,vchan,mintol,maxtol,freq,vidlen,FIRset,special)
D0 D0 D1 D2 D3 D4 D5 D6 D7

ULONG VideoParams(UBYTE,UBYTE,UBYTE,UBYTE,UBYTE,UWORD,UBYTE,UBYTE);

FUNCTION
Sets the default video compression parameters for each video channel. For auto-adjusting compression modes, this is only used for the first field of video.

INPUTS
board - specifies the Flyer board (0-3)
vchan - video channel (0 or 1)
mintol - minimum tolerance mode (0 best, 6 worst)
maxtol - maximum tolerance mode
freq - random noise frequency
vidlen - desired length of compressed field data (in SCSI blocks)
FIRset - FIR presets to use
0 = custom
1 = 25%
2 = 33%
3 = 50%
4 = 100%
special - for testing only

NOTES
Use only tolerance modes 0, 1, 4, 5, and 6

****** This may change as we finalize how the user's controls the amount of compression ******

flyer.library/VoidAllHeads **flyer.library/VoidAllHeads**

NAME
VoidAllHeads - Remove all A/B heads from all Flyers

SYNOPSIS
error = VoidAllHeads()
D0

ULONG VoidAllHeads(void);

FUNCTION
Removes all A/B heads from all drives attached to all Flyers.

SEE ALSO MakeClipHead, VoidCardHeads, VoidClipHead

flyer.library/VoidCardHeads **flyer.library/VoidCardHeads**

NAME
VoidCardHeads - Remove all A/B heads for the Flyer card specified

SYNOPSIS
error = VoidCardHeads(board)
D0 D0

ULONG VoidCardHeads(UBYTE board);

FUNCTION
Removes all A/B heads from drives attached to specified Flyer card

INPUTS
board - specifies the Flyer board (0-3)

SEE ALSO MakeClipHead, VoidAllHeads, VoidClipHead

flyer.library/VoidClipHead **flyer.library/VoidClipHead**

NAME
VoidClipHead - Remove an A/B head for the specified clip

SYNOPSIS
error = VoidClipHead(clipaction)
D0 A0

ULONG VoidClipHead(struct ClipAction *);

FUNCTION
Removes an A/B head for the specified clip. Must exactly match the range of a previously defined head (with MakeClipHead) or this does nothing.

INPUTS
clipaction - specifies clip and in/out points of head to remove

SEE ALSO MakeClipHead, VoidAllHeads, VoidCardHeads

flyer.library/WaitAction **flyer.library/WaitAction**

NAME
WaitAction - Wait for a previously issued action to complete

SYNOPSIS
error = WaitAction(action)
D0 A0

ULONG WaitAction(struct ClipAction *);

FUNCTION
Does not return until the specified action is complete.

INPUTS
action - pointer to structure that was used to issue the original command

RESULT
error - return code (from command)

SEE ALSO CheckAction

NAME
Write10 -- Transfer data from DMA memory to SCSI drive

SYNOPSIS
error = Write10(action,blocks,buffer,lba)
D0 A0 D0 D1 D2

ULONG Write10(struct ClipAction *,WORD,ULONG,ULONG);

INPUTS
clipaction - specifies the volume and the return method
blocks - blocks to transfer
buffer - DMA buffer start (block) of data to write
lba - starting lba

NAME
WriteTest -- Do a write speed test on a Flyer SCSI drive

SYNOPSIS
error = WriteTest(flyervolume,blocks,repeat,lba,dblflag)
D0 A0 D0 D1 D2 D3

ULONG WriteTest(struct FlyerVolume *,ULONG,ULONG,ULONG,UBYTE);

INPUTS
volume - pointer to structure which specifies drive
blocks - size of each transfer (in blocks)
repeat - number of transfers to perform
lba - starting lba on drive
dblflag - 0=simple test
1=double-buffered test

```
*****************************************************************************
*
* Flyer.i - Flyer include file
*
* $Id: Flyer.i,v 1.4 1995/10/16 17:46:23 Flick Exp $
*
* Copyright (c) 1994 NewTek, Inc.
* Confidental and Proprietary. All rights reserved.
*
*    02/23/94        Marty           Created
*****************************************************************************

    IFND    INC_FLYER_I
INC_FLYER_I SET 1

**  Include file for use by Apps calling flyer.library
**  (c) Copyright 1994 NewTek, Inc.
**     All Rights Reserved
**  Marty Flickinger
**
    IFND    EXEC_NODES_I
    INCLUDE     "exec/nodes.i"
    ENDC
    IFND    DOS_DOS_I
    INCLUDE     "dos/dos.i"
    ENDC

FLYERLIBNAME:  MACRO
        DC.B    'flyer.library',0
        DS.W    0
        ENDM

*****************************************
*** Public Return Codes for Flyer Calls ***
*****************************************

*** General Flyer Errors ***

FERR_OKAY           EQU $00     ;All went well
FERR_CMDFAILED      EQU $01     ;Command failed for some reason
FERR_BUSY           EQU $02     ;Still in progress
FERR_ABORTED        EQU $03     ;User abort
FERR_BADPARAM       EQU $04     ;Bad command parameter
FERR_BADCOMMAND     EQU $05     ;Command not defined/supported
FERR_BADVIDHDR      EQU $06     ;Ran out of video - no header detected
FERR_WRONGMODE      EQU $07     ;Wrong play/rec mode for action
FERR_OLDDATA        EQU $08     ;Incompatible data
FERR_NOAUDIOCHAN    EQU $09     ;No free audio channel(s)
FERR_CHANINUSE      EQU $0A     ;Video/SCSI channel not available
FERR_BADFLDHAND     EQU $0B     ;Bad or missing field handle
FERR_CLIPLATE       EQU $0C     ;A/V clip started late
FERR_DROPPEDFLDS    EQU $0D     ;Dropped 1 or more fields


*** Flyer Internal Errors ***

FERR_NOTASKS        EQU $10     ;No SCSI tasks available for use
FERR_LISTCORRUPT    EQU $11     ;Internal list corrupt
FERR_NOTINRANGE     EQU $12     ;Internal list error
FERR_EEFAILURE      EQU $13     ;EEPROM failure
FERR_NOFINDERS      EQU $14     ;No FrameFinders available for use
FERR_BADMODULE      EQU $1F     ;Incompatible module provided
```

***** FileSystem Errors *****

```
FERR_OBJNOTFOUND    EQU $20  ;Could not find file/dir
FERR_FULL           EQU $21  ;Drive full
FERR_DIRFULL        EQU $22  ;Directory full
FERR_EXHAUSTED      EQU $23  ;Directory list exhausted
FERR_FSFAIL         EQU $24  ;FileSystem failure
FERR_WRONGTYPE      EQU $25  ;Wrong type of object
FERR_UNFORMATTED    EQU $26  ;Drive not high-level formatted
FERR_EXCLUDED       EQU $27  ;Exclusive lock prevented action
FERR_OUTOFRANGE     EQU $28  ;Seek beyond bounds
FERR_CANTEXTEND     EQU $29  ;End of file, and cannot extend file
FERR_PROTECTED      EQU $2A  ;Drive write-protected
FERR_DIFFERENT      EQU $2B  ;Grips are different objects
FERR_EXISTS         EQU $2C  ;File already exists
FERR_NOMEM          EQU $2D  ;Out of storage
FERR_DELPROT        EQU $2E  ;Delete-protected file
FERR_READPROT       EQU $2F  ;Read-protected file
FERR_WRITEPROT      EQU $30  ;Write-protected file
FERR_INUSE          EQU $31  ;Disk/object in use
FERR_DIRNOTEMPTY    EQU $32  ;Directory was not empty
```

***** SCSI Errors *****

```
FERR_SELTIMEOUT     EQU $40  ;SCSI Time-out -- no drive
FERR_BADSTATUS      EQU $41  ;Bad status after executing command
```

***** Sequencing Errors *****

```
FERR_WRONGDATATYPE  EQU $60  ;Asked for improper type of data from clip
FERR_DRIVEINCAPABLE EQU $61  ;Using video clip from a non-video drive
FERR_NO_BROLLDRIVE  EQU $62  ;No video B-roll drive found
FERR_HEADFAILED     EQU $63  ;A/B head missing/problem
```

***** Amiga Library Errors *****

```
FERR_NOCARD         EQU $70  ;Flyer card specified does not exist
FERR_LIBFAIL        EQU $71  ;Library failed to pass command to Flyer
FERR_ASYNCFAIL      EQU $72  ;An asynchronous command failed
FERR_VOLNOTFOUND    EQU $73  ;Volume name not found
FERR_NOFREECMD      EQU $74  ;Library<->Flyer RAM clogged
FERR_BADID          EQU $75  ;Illegal async ID

FERR_LIMIT          EQU $7F
```

****
***** Mode - for use with StillMode command *****
****

```
MODE_FIELD          EQU      1
MODE_PAIR           EQU      2
MODE_FRAME          EQU      4
```

***** Structure returned from FlyerDriveInfo call *****

```
STRUCTURE FlyerVolInfo,0
    UWORD   fvi_len                 ;Length of this structure
    ULONG   fvi_Ident               ;'ROOT' for good volumes
    UBYTE   fvi_Version             ;Version of FileSystem that wrote drive
    UBYTE   fvi_LTitle              ;Length for L-String
    STRUCT  fvi_Title,80            ;String (null-terminated)
    ULONG   fvi_Blocks              ;Total user blocks
    ULONG   fvi_BlksFree            ;User blocks free
    UBYTE   fvi_Flags               ;FVIF_xxx - see below
    UBYTE   fvi_DiskOkay            ;FileSys in good shape?
    UWORD   fvi_BlkSize             ;Block size
    STRUCT  fvi_DiskDate,ds_SIZEOF
    ULONG   fvi_FragBlks            ;Fragmented blocks
    ULONG   fvi_Largest             ;Largest free chunk (in blocks)
    ULONG   fvi_Optimized           ;Largest free chunk if optimized (in blocks)
    STRUCT  fvi_reserved,4*4
    LABEL   FVI_sizeof
```

***** FlyerVolInfo Flags *****

```
    BITDEF  FVI,VIDEOREADY,0        ;Drive can handle video
    BITDEF  FVI,AUDIOREADY,1        ;Drive can handle audio
    BITDEF  FVI,WRITEPROT,2         ;Drive is not writable
```

***** Flyer Volume node structure *****

```
STRUCTURE FlyVolNode,LN_SIZE
    STRUCT  fvn_Name,80             ;Name of volume
    UBYTE   fvn_Board               ;Flyer board number
    UBYTE   fvn_SCSIdrive           ;SCSI channel/unit
    UBYTE   fvn_Flags               ;From FlyerVolInfo (FVIF_xxx)
    UBYTE   fvn_pad
    LABEL   FVN_sizeof
```

***** Structure returned from GetClipInfo call *****

```
STRUCTURE ClipInfo,0
    UWORD   ci_len                  ;Length of this structure
    STRUCT  ci_Name,80
    STRUCT  ci_Comment,80
    UBYTE   ci_Flags                ;CIF_xxxx -- see below
    UBYTE   ci_Type                 ;Unused -- FLYER_TYPE_xxx
    STRUCT  ci_Date,ds_SIZEOF
    ULONG   ci_Bits                 ;User defined
    ULONG   ci_Fields               ;Length in fields
    ULONG   ci_Start                ;Start block on drive
    ULONG   ci_Length               ;Byte length
    ULONG   ci_IndexBlk             ;Location of clip's index
    UBYTE   ci_NumAudChans          ;Number of audio channels contained
    UBYTE   ci_VideoGrade           ;VG_xxx below
    ULONG   ci_EndBlk               ;Last blk used + 1
    ULONG   ci_LengthExt            ;Extended Byte length: high 32 bits
    STRUCT  ci_reserved,22
    LABEL   CI_sizeof
```

```
*** Flyer object types ***

FLYER_TYPE_FILE                 EQU     1
FLYER_TYPE_DIR                  EQU     2
FLYER_TYPE_ROOT                 EQU     3
```

***** ClipInfo Flags *****

```
    BITDEF  CI,HASVIDEO,0                   ;Clip contains video
    BITDEF  CI,HASAUDIO,1                   ;Clip contains audio
```

***** Video grade values *****

```
VG_STD                          EQU     0   ;Standard grade video
VG_HQ5                          EQU     1   ;HQ5 grade video
```

***** Structure returned from GetSMPTE call *****

```
  STRUCTURE SMPTEinfo,0
    UBYTE   si_SMPTEvalid                   ;SMPTE info valid?
    UBYTE   si_SMPTEhours                   ;Hours
    UBYTE   si_SMPTEminutes                 ;Minutes
    UBYTE   si_SMPTEseconds                 ;Seconds
    UBYTE   si_SMPTEframes                  ;Frames (1/30th)
    UBYTE   si_SMPTEuser1                   ;BW1 and 2
    UBYTE   si_SMPTEuser2                   ;BW3 and 4
    UBYTE   si_SMPTEuser3                   ;BW5 and 6
    UBYTE   si_SMPTEuser4                   ;BW7 and 8
    UBYTE   si_SMPTEflags                   ;See SIx_xxxx below
    LABEL   SI_sizeof
```

***** SMPTE Flags *****

```
    BITDEF  SI,DROPFRAME,0                  ;Position is in drop-frame format
    BITDEF  SI,COLORFRAME,1                 ;Color Frame identification applied
    BITDEF  SI,REVERSE,2                    ;Time code is reverse-direction
```

****
***** Defines for the SetSerDevice command *****
****

```
SERDEVTYPE_NONE                 equ     0
SERDEVTYPE_SMPTE                equ     1
SERDEVTYPE_CTRL                 equ     2
SERDEVTYPE_MIDI                 equ     3

SERDEV_GEN_9600                 equ     1
SERDEV_HORITA                   equ     10
SERDEV_TELCOM                   equ     11
```

***** Structure used with OpenWriteField call *****

```
  STRUCTURE VidCompInfo,0
     UBYTE   vci_Algo                    ;Algorithm type
     UBYTE   vci_Tolerance               ;Error tolerance
     UBYTE   vci_FIRcomp                 ;FIR compensation level
     UBYTE   vci_RndFreq                 ;Noise frequency
     UBYTE   vci_RndSeed                 ;Noise seed
     UBYTE   vci_Flags                   ;VCIx_xxx Flags
     UWORD   vci_DataSize                ;Max field size in blocks
     ULONG   vci_Private1                ;Internal use only
     ULONG   vci_Private2                ;Internal use only
     STRUCT  vci_reserved,6
     LABEL   VCI_sizeof

ALGO_D2                          EQU  1     ;D2
ALGO_SN                          EQU  2     ;sub-nyquist
```

***** VidCompInfo Flags *****

```
     BITDEF   VCI,AUTOMODE,0  ;Optimize compression on the fly

SAMPLESPERLINE                   EQU  752   ;Samples per active video line
TOASTERLINES                     EQU  242   ;Lines per active video field (Toaster)
LINESPERFIELD                    EQU  243   ;Lines per active video field (Full NTSC)
```

***** Field Modes - for use with OpenWriteField command *****

```
     BITDEF   FW,NEW,0                   ;Create new clip
     BITDEF   FW,APPEND,1                ;Append to clip
     BITDEF   FW,REWRITE,2               ;Rewrite field
     BITDEF   FW,FRAME,3                 ;Redo entire frame
     BITDEF   FW,HALFLINES,4             ;Support full NTSC fields (+half lines)
     BITDEF   FW,NOFIR,5                 ;Omit FIR filtering
```

***** Field Modes - for use with OpenReadField command *****

```
     BITDEF   FR,HALFLINES,4             ;Support full NTSC fields (+half lines)


  STRUCTURE FlyerVolume,0
     APTR     fv_Path                    ;Pointer to Volume:clipname string
     UBYTE    fv_Board                   ;Board number (0...n)
     UBYTE    fv_SCSIdrive               ;SCSI unit on channel
     UBYTE    fv_Flags                   ;See below
     UBYTE    fv_pad
     STRUCT   fv_reserved,4*4
     LABEL    FV_sizeof


     BITDEF   FV,USENUMS,0               ;Ignore volume in string, use numbers
```

```
STRUCTURE ClipAction,0
  APTR    ca_Volume                ;Ptr to FlyerVolume structure
  ULONG   ca_ID                    ;For asynchronous operation
  UBYTE   ca_ReturnTime            ;When this call should return
  UBYTE   ca_Channel               ;Video channel to use: 0, 1, or FF for choice
  UBYTE   ca_Flags                 ;CAF_xxxxx -- see below
  UBYTE   ca_PermissFlags          ;CAPF_xxxxx -- see below
  ULONG   ca_VidStartField
  ULONG   ca_AudStartField
  ULONG   ca_VidFieldCount
  ULONG   ca_AudFieldCount
  ULONG   ca_GoClock
  UWORD   ca_MatteY                ;For use with CAF_USEMATTE
  BYTE    ca_MatteI
  BYTE    ca_MatteQ
  UWORD   ca_VolAttack             ;Attack time/ramp time
  UWORD   ca_VolSust1              ;Channel 1 sustain volume
  UWORD   ca_VolSust2              ;Channel 2 sustain volume
  UWORD   ca_VolDecay              ;Decay time
  WORD    ca_AudioPan1             ;Channel 1 pan (-left, 0 ctr, +right)
  WORD    ca_AudioPan2             ;Channel 2 pan (-left, 0 ctr, +right)
  ULONG   ca_TotalAudStart         ;Combined audio start field
  ULONG   ca_TotalAudLength        ;Combined audio field count
  STRUCT  ca_reserved0,4*4
  ;These block values provide a "raw" access to clip points
  ULONG   ca_StartBlk
  ULONG   ca_EndBlk
  ULONG   ca_UserID                ;Caller-private ID used for sequencing errors
  STRUCT  ca_reserved1,3*4
  ;These things used by the FileSystem
  ULONG   ca_Grip
  ULONG   ca_FileID
  UBYTE   ca_Access
  UBYTE   ca_pad2
  ULONG   ca_FldHandle             ;Handle to open field for direct R/W
  STRUCT  ca_reserved2,3*4
  ;Status communication data
  ULONG   ca_Status
  ;Special return values -- Valid only when command complete
  ULONG   ca_LastFieldDone
  LABEL   CA_sizeof
```

***** ReturnTime values *****

```
RT_FREE      equ  0   ;Return immediately, never follow-up
RT_IMMED     equ  1   ;Return immediately
RT_STARTED   equ  2   ;Return when actually started
RT_STOPPED   equ  3   ;Return when action stops
```

***** ClipAction flags *****

```
  BITDEF  CA,VIDEO,0          ;Include clip video
  BITDEF  CA,AUDIOL,1         ;Include left audio channel
  BITDEF  CA,AUDIOR,2         ;Include right audio channel
  BITDEF  CA,USEMATTE,3       ;Display matte color when clip done
  BITDEF  CA,NOPREROLL,4      ;Skip pre-roll
  BITDEF  CA,APPEND,5         ;For single-frame appending
                                -- NOT SUPPORTED HERE
  BITDEF  CA,REPROCESS,6      ;Force clip re-processing
```

***** ClipAction permission flags *****

```
BITDEF   CAP,STEALOURVIDEO,0      ;Can steal requested video channel
BITDEF   CAP,KILLOTHERVIDEO,1     ;Can kill other video channel(s)
BITDEF   CAP,ERRIFBUSY,2          ;Return error rather than wait for it
BITDEF   CAP,AUTOMUTE,3           ;Automatically mute audio when looping
BITDEF   CAP,USEHEADS,4           ;Use A/V heads when present and needed
```

***** FlyerInputSel values (video source) *****

```
FI_Camcorder        equ   0   ;Needs TBC
FI_SVHS             equ   1   ;Needs TBC
FI_Toaster1         equ   2
FI_Toaster2         equ   3
FI_ToasterMain      equ   4
FI_ToasterPV        equ   5
```

***** FlyerInputSel values (sync source) *****

```
FS_ToasterMain      equ   0
FS_Toaster1         equ   1
```

*** Flyer Calibration values ***

```
CALIB_DACA_PHASE_EDGE       equ 0
CALIB_DACA_PHASE_COURSE     equ 1
CALIB_DACA_PHASE_FINE       equ 2
CALIB_DACB_PHASE_EDGE       equ 3
CALIB_DACB_PHASE_COURSE     equ 4
CALIB_DACB_PHASE_FINE       equ 5
CALIB_ADC_PHASE_EDGE        equ 6
CALIB_ADC_PHASE_COURSE      equ 7
CALIB_ADC_PHASE_FINE        equ 8
CALIB_HPLAYOFFSETA          equ 9
CALIB_HPLAYOFFSETB          equ 10
CALIB_HRECOFFSETA           equ 11
CALIB_HRECOFFSETB           equ 12
CALIB_PEDESTALA             equ 13
CALIB_PEDESTALB             equ 14
```

```
*****************************************
*** TBC control structure and defines ***
*****************************************

  STRUCTURE TBCctrl,0
     UBYTE   tbc_Status          ;Status flags
     UBYTE   tbc_Flags           ;General flags
     UBYTE   tbc_DecFlags        ;Decoder flags
     UBYTE   tbc_EncFlags        ;Encoder flags
     UBYTE   tbc_InputSel        ;Input mux control
     UBYTE   tbc_Term            ;Termination control
     BYTE    tbc_Bright          ;Brightness value (-64 to 63)
     UBYTE   tbc_Contrast        ;Contrast value (0 to 127)
     UBYTE   tbc_Sat             ;Saturation value (0 to 127)
     BYTE    tbc_Hue             ;Hue value (-64 to 63)
     UWORD   tbc_Phase           ;Phase adjust ($000 to $7FF)
     UWORD   tbc_HorAdj          ;Horizontal adjust ($000 to $FFF)
     UBYTE   tbc_Fader           ;Fader value (0 to 255)
     UBYTE   tbc_KeyerFlags      ;Keyer flags
     STRUCT  tbc_reserved,38
     LABEL   TBC_sizeof


*** TBC Operations ***

     BITDEF  TBCO,STATUS,0       ;Fill in status field
     BITDEF  TBCO,MODES,1        ;Set modes/flags/muxes
     BITDEF  TBCO,ADJUST,2       ;Set adjustment values


*** TBC Status flags ***

     BITDEF  TBCS,MODULE,0       ;TBC module present
     BITDEF  TBCS,VIDEO,1        ;Video present at input
     BITDEF  TBCS,STABLE,2       ;Video input stable


*** TBC General flags ***

     BITDEF  TBCG,BYPASS,0       ;Bypass TBC
     BITDEF  TBCG,FREEZE,1       ;Freeze video


*** TBC Decoder flags ***

     BITDEF  TBCD,AGC,0          ;Enable AGC
     BITDEF  TBCD,CHROMAAGC,1    ;Enable chroma AGC
     BITDEF  TBCD,MONOCHROME,2   ;Monochrome input


*** TBC Encoder flags ***

     BITDEF  TBCE,BARS,0         ;Output 100% sat, 75% ampl bars
     BITDEF  TBCE,KILLCOLOR,1    ;Disable color on output


*** TBC InputSel values ***

TBCIN_YC                         equ0;Flyer SVHS input
TBCIN_COMP                       equ1;Flyer Composite input
TBCIN_TMAIN                      equ2;Toaster Main output
TBCIN_FADER                      equ3;TBC Fader output
```

```
*** TBC Termination ***

      BITDEF   TBCT,FADERA,2          ;Flyer A channel input
      BITDEF   TBCT,FADERB,3          ;Flyer B channel input
      BITDEF   TBCT,OUT,4             ;Video output
      BITDEF   TBCT,GENIN,5           ;Genlock input
      BITDEF   TBCT,COMPIN,7          ;Composite video input


*** TBC Keyer flags ***

      BITDEF   TBCK,KEYONB,0          ;Src (A/B)
      BITDEF   TBCK,MODE0,1           ;Keyer mode 0 bit
      BITDEF   TBCK,MODE1,2           ;Keyer mode 1 bit
      BITDEF   TBCK,FADEROUT,3        ;TBC/Fader mux (1 = Fader)


*** StartClipCutList flags ***

      BITDEF   CCL,DESTRUCTIVE,0      ;Delete unused portions of original clip


**********************************************
*** Structure used with FlyerAudioCtrl call ***
**********************************************

   STRUCTURE FlyAudCtrl,0
      UBYTE    fac_Flags             ;General flags
      UBYTE    fac_pad
      UBYTE    fac_LeftSense         ;Left channel overrange detector (0 - 3)
      UBYTE    fac_RightSense        ;Right channel overrange detector (0 - 3)
      UBYTE    fac_LeftSrc           ;Left input source mux (see below)
      UBYTE    fac_RightSrc          ;Right input source mux (see below)
      UBYTE    fac_LeftGain          ;Left channel gain (0 or 1 to 16)
      UBYTE    fac_RightGain         ;Right channel gain (0 or 1 to 16)
      UBYTE    fac_Aux1Mix           ;Aux1 mix-in amount (-16 to 15, -128 to mute)
      UBYTE    fac_reserved1
      UBYTE    fac_Aux2Mix           ;Aux2 mix-in amount (-16 to 15, -128 to mute)
      UBYTE    fac_reserved2
      STRUCT   fac_reserved,8
      LABEL    FAC_sizeof

*** Audio Ctrl Operations ***

      BITDEF   FACO,SENSE,0          ;Report overrange information
      BITDEF   FACO,SETGAIN,1        ;Set input gain as specified
      BITDEF   FACO,SETSRC,2         ;Set input sources as specified
      BITDEF   FACO,SETMIX,3         ;Set aux mixing as specified
      BITDEF   FACO,SENSE8,4         ;Return 8 bit L/R readings in record mode

*** Audio Ctrl Input Sources ***

FACS_LINE1          equ   0   ;RCA line-in jacks on rear of card
FACS_AUX1           equ   1   ;Aux1 connector (JP19)
FACS_LINE2          equ   2   ;Line2 connector (JP1)

*** FlyerOptions Non-Volatile Flags ***

      BITDEF   FLYOPT,DropFramDet,0  ;Stop recording on dropped frame
      BITDEF   FLYOPT,NOT_HQ5,1      ;=0 to enable HQ5

      ENDC     ; INC_FLYER_I
```

```
/*****************************************************************************\
*
* Flyer.h - Flyer include file
*
* $Id: Flyer.h,v 1.4 1995/10/16 17:28:47 Flick Exp $
*
* $Log: Flyer.h,v $
*Revision 1.4  1995/10/16  17:28:47  Flick
* Copyright (c) 1994 NewTek, Inc.
* Confidental and Proprietary. All rights reserved.
*
*     02/23/94          Marty             Created
\*****************************************************************************/

#ifndef    INC_FLYER_H
#define    INC_FLYER_H

/*
**    Include file for use by Apps calling flyer.library
**
**    (c) Copyright 1994 NewTek, Inc.
**       All Rights Reserved
*/

#ifndef EXEC_NODES_H
#include "exec/nodes.h"
#endif
#ifndef DOS_DOS_H
#include "dos/dos.h"
#endif

#define    FLYERLIBNAME          "flyer.library"

/*******************************************\
**** Public Return Codes for Flyer Calls ****
\*******************************************/

/*** General Flyer Errors ***/

#define    FERR_OKAY             0x00    /* All went well */
#define    FERR_CMDFAILED        0x01    /* Command failed for some reason */
#define    FERR_BUSY             0x02    /* Still in progress */
#define    FERR_ABORTED          0x03    /* User abort */
#define    FERR_BADPARAM         0x04    /* Bad command parameter */
#define    FERR_BADCOMMAND       0x05    /* Command not defined/supported */
#define    FERR_BADVIDHDR        0x06    /* Ran out of video - no header detected */
#define    FERR_WRONGMODE        0x07    /* Wrong play/rec mode for action */
#define    FERR_OLDDATA          0x08    /* Incompatible data */
#define    FERR_NOAUDIOCHAN      0x09    /* No free audio channel(s) */
#define    FERR_CHANINUSE        0x0A    /* Video/SCSI channel not available */
#define    FERR_BADFLDHAND       0x0B    /* Bad field handle */
#define    FERR_CLIPLATE         0x0C    /* A/V clip started late */
#define    FERR_DROPPEDFLDS      0x0D    /* Dropped 1 or more fields */

/*** Flyer Internal Errors ***/

#define    FERR_NOTASKS          0x10    /* No SCSI tasks available for use */
#define    FERR_LISTCORRUPT      0x11    /* Internal list corrupt */
#define    FERR_NOTINRANGE       0x12    /* Internal list error */
#define    FERR_EEFAILURE        0x13    /* EEPROM failure */
#define    FERR_NOFINDERS        0x14    /* No FrameFinders available for use */
#define    FERR_BADMODULE        0x1F    /* Incompatible module provided */
```

/* FileSystem Errors ***/**

```
#define       FIRSTFSERR          0x20
#define   FERR_OBJNOTFOUND        0x20   /* Could not find file/dir */
#define   FERR_FULL               0x21   /* Drive full */
#define   FERR_DIRFULL            0x22   /* Directory full */
#define   FERR_EXHAUSTED          0x23   /* Directory list exhausted */
#define   FERR_FSFAIL             0x24   /* FileSystem failure */
#define   FERR_WRONGTYPE          0x25   /* Wrong type of object */
#define   FERR_UNFORMATTED        0x26   /* Drive not high-level formatted */
#define   FERR_EXCLUDED           0x27   /* Exclusive lock prevented action */
#define   FERR_OUTOFRANGE         0x28   /* Seek beyond bounds */
#define   FERR_CANTEXTEND         0x29   /* End of file, and cannot extend file */
#define   FERR_PROTECTED          0x2A   /* Drive write-protected */
#define   FERR_DIFFERENT          0x2B   /* Grips are different objects */
#define   FERR_EXISTS             0x2C   /* File already exists */
#define   FERR_NOMEM              0x2D   /* Out of storage */
#define   FERR_DELPROT            0x2E   /* Delete-protected file */
#define   FERR_READPROT           0x2F   /* Read-protected file */
#define   FERR_WRITEPROT          0x30   /* Write-protected file */
#define   FERR_INUSE              0x31   /* Disk/object in use */
#define   FERR_DIRNOTEMPTY        0x32   /* Directory was not empty */
#define       LASTFSERR           0x32
```

/* SCSI Errors ***/**

```
#define   FERR_SELTIMEOUT         0x40   /* SCSI Time-out -- no drive */
#define   FERR_BADSTATUS          0x41   /* Bad status after executing command */
```

/* Sequencing Errors ***/**

```
#define   FERR_WRONGDATATYPE      0x60   /* Asked for wrong type of data from clip */
#define   FERR_DRIVEINCAPABLE     0x61   /* Using video clip from a non-video drive */
#define   FERR_NO_BROLLDRIVE      0x62   /* No video B-roll drive found */
#define   FERR_HEADFAILED         0x63   /* A/B head missing/problem */
```

/* Amiga Library Errors ***/**

```
#define   FERR_NOCARD             0x70   /* Flyer card specified does not exist */
#define   FERR_LIBFAIL            0x71   /* Library failed to pass command to Flyer */
#define   FERR_ASYNCFAIL          0x72   /* An asynchronous command failed */
#define   FERR_VOLNOTFOUND        0x73   /* Volume name not found */
#define   FERR_NOFREECMD          0x74   /* Library<->Flyer RAM clogged */
#define   FERR_BADID              0x75   /* Illegal async ID */

#define   FERR_LIMIT              0x7F
```

/* Mode - for use with StillMode command ***/**

```
#define   MODE_FIELD              1
#define   MODE_PAIR               2
#define   MODE_FRAME              4
```

/* Structure returned from FlyerDriveInfo call ***/**

```
struct FlyerVolInfo {
    UWORD       len;                    /* Length of this structure */
    ULONG       Ident;                  /* 'ROOT' for good volumes */
    UBYTE       Version;                /* Version of FileSystem that wrote drive */
    UBYTE       LTitle;                 /* Length for L-String */
    char        Title[80];              /* String (null-terminated) */
    ULONG       Blocks;                 /* Total user blocks */
    ULONG       BlksFree;               /* User blocks free */
    UBYTE       Flags;                  /* FVIF_xxx - see below */
    UBYTE       DiskOkay;               /* FileSys in good shape? */
    UWORD       BlkSize;                /* Block size */
    struct      DateStamp   DiskDate;
    ULONG       FragBlks;               /* Fragmented blocks */
    ULONG       Largest;                /* Largest free chunk (in blocks) */
    ULONG       Optimized;              /* Largest free chunk if optimized (in blocks) */
    ULONG       reserved[4];
};
```

/* FlyerVolInfo Flags ***/**

```
#define   FVIF_VIDEOREADY       1       /* Drive can handle video */
#define   FVIF_AUDIOREADY       2       /* Drive can handle audio */
#define   FVIF_WRITEPROT        4       /* Drive is not writable */
```

/* Volume node structure ***/**

```
struct FlyerVolNode {
    struct      Node   Node;
    char        Name[80];               /* Name of volume */
    UBYTE       Board;                  /* Flyer board number */
    UBYTE       SCSIdrive;              /* SCSI channel/unit */
    UBYTE       Flags;                  /* From FlyerVolInfo (FVIF_xxx) */
    UBYTE       pad;
};
```

/* Structure returned from GetClipInfo call ***/**

```
struct ClipInfo {
    UWORD       len;                    /* Length of this structure */
    char        Name[80];
    char        Comment[80];
    UBYTE       Flags;                  /* OIF_xxxx -- see below */
    UBYTE       Type;                   /* FLYER_TYPE_xxx */
    struct      DateStamp     Date;
    ULONG       Bits;                   /* User defined */
    ULONG       Fields;                 /* Length in fields */
    ULONG       Start;                  /* Start block on drive */
    ULONG       Length;                 /* Byte length */
    ULONG       IndexBlk;               /* Location of clip's index */
    UBYTE       NumAudChans;            /* Number of audio channels contained */
    UBYTE       VideoGrade;             /* VG_xxx below */
    ULONG       EndBlk;                 /* Last blk used +1 */
    ULONG       LengthExt;              /* Extended Byte length: high 32 bits */
    UBYTE       reserved[22];
};
```

/* Flyer object types ***/**

```
#define    FLYER_TYPE_FILE      1
#define    FLYER_TYPE_DIR       2
#define    FLYER_TYPE_ROOT      3
```

/* ClipInfo Flags ***/**

```
#define    CIF_HASVIDEO         1    /* Clip contains video */
#define    CIF_HASAUDIO         2    /* Clip contains audio */
```

/* Video Grade values ***/**

```
#define    VG_STD               0    /* Standard grade video */
#define    VG_HQ5               1    /* HQ5 grade video */
```

/* Structure returned from GetSMPTE call ***/**

```
struct SMPTEinfo {
      UBYTE    SMPTEvalid;          /* SMPTE info valid? */
      UBYTE    SMPTEhours;          /* Hours */
      UBYTE    SMPTEminutes;        /* Minutes */
      UBYTE    SMPTEseconds;        /* Seconds */
      UBYTE    SMPTEframes;         /* Frames (1/30th) */
      UBYTE    SMPTEuser1;          /* BW1 and 2 */
      UBYTE    SMPTEuser2;          /* BW3 and 4 */
      UBYTE    SMPTEuser3;          /* BW5 and 6 */
      UBYTE    SMPTEuser4;          /* BW7 and 8 */
      UBYTE    SMPTEflags;          /* See SIF_xxxx below */
};
```

/* SMPTE flags ***/**

```
#define    SIF_DropFrame        1    /* Position is in drop-frame format */
#define    SIF_ColorFrame       2    /* Color Frame identification applied */
#define    SIF_Reverse          4    /* Time code is reverse-direction */
```

/* Defines for the SetSerDevice command ***/**

```
#define    SERDEVTYPE_NONE      0
#define    SERDEVTYPE_SMPTE     1
#define    SERDEVTYPE_CTRL      2
#define    SERDEVTYPE_MIDI      3

#define    SERDEV_GEN_9600      1
#define    SERDEV_HORITA        10
#define    SERDEV_TELCOM        11
```

/* Structure used with OpenWriteField call ***/**

```
struct VidCompInfo {
    UBYTE       Algo;           /* Algorithm type */
    UBYTE       Tolerance;      /* Error tolerance */
    UBYTE       FIRcomp;        /* FIR compensation level */
    UBYTE       RndFreq;        /* Noise frequency */
    UBYTE       RndSeed;        /* Noise seed */
    UBYTE       Flags;          /* VCIF_xxx Flags */
    UWORD       DataSize;       /* Max field size in blocks */
    ULONG       Private1;
    ULONG       Private2;
    UBYTE       reserved[6];
};

#define   ALGO_D2               1       /* D2 */
#define   ALGO_SN               2       /* Sub-nyquist */

#define   VCIF_AUTOMODE         1       /* Optimize compression on the fly */

#define   SAMPLESPERLINE        752     /* Samples per active video line */
#define   TOASTERLINES          242     /* Lines per active video field (Toaster) */
#define   LINESPERFIELD         243     /* Lines per active video field (Full NTSC) */
```

/* FieldModes - for use with OpenWriteField command ***/**

```
#define   FWF_NEW               1       /* Create new clip */
#define   FWF_APPEND            2       /* Append field to clip */
#define   FWF_REWRITE           4       /* Rewrite field */
#define   FWF_FRAME             8       /* Redo entire frame */
#define   FWF_HALFLINES         16      /* Support full NTSC fields (+half lines) */
#define   FWF_NOFIR             32      /* Omit FIR filtering */
```

/* FieldModes - for use with OpenReadField command ***/**

```
#define   FRF_HALFLINES         16      /* Support full NTSC fields (+half lines) */

struct FlyerVolume {
    char        *Path;          /* Pointer to volume:clipname string */
    UBYTE       Board;          /* Board number (0...n) */
    UBYTE       SCSIdrive;      /* SCSI channel/unit */
    UBYTE       Flags;          /* See below */
    UBYTE       pad;
    ULONG       reserved[4];
};

#define   FVF_USENUMS           0x01    /* Ignore volume in string, use numbers */
```

```
struct ClipAction {
    struct FlyerVolume *Volume;          /* Volume and clip name info */
    ULONG  ID;                           /* For asynchronous operation */
    UBYTE  ReturnTime;                   /* See RT_xxx below */
    UBYTE  Channel;                      /* Video channel to use: 0, 1, or FF for choice */
    UBYTE  Flags;                        /* CAF_xxxxx -- see below */
    UBYTE  PermissFlags;                 /* CAPF_xxxxx -- see below */
    ULONG  VidStartField;
    ULONG  AudStartField;
    ULONG  VidFieldCount;
    ULONG  AudFieldCount;
    ULONG  GoClock;
    UWORD  MatteY;                       /* For use with CAF_USEMATTE */
    BYTE   MatteI;
    BYTE   MatteQ;
    UWORD  VolAttack;                    /* Attack time/ramp time */
    UWORD  VolSust1;                     /* Channel 1 sustain volume */
    UWORD  VolSust2;                     /* Channel 2 sustain volume */
    UWORD  VolDecay;                     /* Decay time */
    WORD   AudioPan1;                    /* Channel 1 pan (-left, 0 ctr, +right) */
    WORD   AudioPan2;                    /* Channel 2 pan (-left, 0 ctr, +right) */
    ULONG  TotalAudStart;                /* Combined audio start field */
    ULONG  TotalAudLength;               /* Combined audio field count */
    ULONG  reserved0[4];
    /* These block values provide a "raw" access to clip points */
    ULONG  StartBlk;
    ULONG  EndBlk;
    ULONG  UserID;                       /* Caller-private ID used for sequencing errors */
    ULONG  reserved1[3];
    /* These things used by the FileSystem */
    ULONG  Grip;
    ULONG  FileID;
    UBYTE  Access;
    UBYTE  pad2;
    ULONG  FldHandle;                    /* Handle to open field for direct R/W */
    ULONG  reserved2[3];
    /* Status communication data */
    ULONG  Status;
    /* Special return values -- Valid only when command complete */
    ULONG  LastFieldDone;
};
```

/* ReturnTime values ***/**

```
#define RT_FREE          0      /* Return immediately, never follow-up */
#define RT_IMMED         1      /* Return immediately */
#define RT_STARTED       2      /* Return when actually started */
#define RT_STOPPED       3      /* Return when action stops */
```

/* ClipAction flags ***/**

```
#define CAF_VIDEO        0x01   /* Include clip video */
#define CAF_AUDIOL       0x02   /* Include left audio channel */
#define CAF_AUDIOR       0x04   /* Include right audio channel */
#define CAF_USEMATTE     0x08   /* Display matte color when clip done */
#define CAF_NOPREROLL    0x10   /* Skip pre-roll */
#define CAF_APPEND       0x20   /* For single-frame appending --
                                   NOT SUPPORTED HERE */
#define CAF_REPROCESS    0x40   /* Force clip re-processing */
```

/* ClipAction permission flags ***/**

```
#define   CAPF_STEALOURVIDEO       0x01  /* Can steal requested video channel */
#define   CAPF_KILLOTHERVIDEO      0x02  /* Can kill other video channel(s) */
#define   CAPF_ERRIFBUSY           0x04  /* Return error rather than wait for it */
#define   CAPF_AUTOMUTE            0x08  /* Automatically mute audio when looping */
#define   CAPF_USEHEADS            0x10  /* Use A/V heads when present and needed */
```

/* FlyerInputSel values (video source) ***/**

```
#define   FI_Camcorder             0     /* Needs TBC */
#define   FI_SVHS                  1     /* Needs TBC */
#define   FI_Toaster1              2     /* Toaster input 1 */
#define   FI_Toaster2              3     /* Toaster input 2 */
#define   FI_ToasterMain           4     /* Toaster Main bus output */
#define   FI_ToasterPV             5     /* Toaster Preview bus output */
```

/* FlyerInputSel values (sync source) ***/**

```
#define   FS_ToasterMain           0     /* Toaster Main bus output */
#define   FS_Toaster1              1     /* Toaster input 1 */
```

/* Flyer Calibration values ***/**

```
#define   CALIB_DACA_PHASE_EDGE        0
#define   CALIB_DACA_PHASE_COURSE      1
#define   CALIB_DACA_PHASE_FINE        2
#define   CALIB_DACB_PHASE_EDGE        3
#define   CALIB_DACB_PHASE_COURSE      4
#define   CALIB_DACB_PHASE_FINE        5
#define   CALIB_ADC_PHASE_EDGE         6
#define   CALIB_ADC_PHASE_COURSE       7
#define   CALIB_ADC_PHASE_FINE         8
#define   CALIB_HPLAYOFFSETA           9
#define   CALIB_HPLAYOFFSETB           10
#define   CALIB_HRECOFFSETA            11
#define   CALIB_HRECOFFSETB            12
#define   CALIB_PEDESTALA              13
#define   CALIB_PEDESTALB              14
```

```
/*** TBC control structure and defines ***/

struct TBCctrl {
    UBYTE   Status;                 /* Status flags */
    UBYTE   Flags;                  /* General flags */
    UBYTE   DecFlags;               /* Decoder flags */
    UBYTE   EncFlags;               /* Encoder flags */
    UBYTE   InputSel;               /* Input mux control */
    UBYTE   Term;                   /* Termination control */
    BYTE    Bright;                 /* Brightness value (-64 to 63) */
    UBYTE   Contrast;               /* Contrast value (0 to 127) */
    UBYTE   Sat;                    /* Saturation value (0 to 127) */
    BYTE    Hue;                    /* Hue value (-64 to 63) */
    UWORD   Phase;                  /* Phase adjust ($000 to $7FF) */
    UWORD   HorAdj;                 /* Horizontal adjust ($000 to $FFF) */
    BYTE    Fader;                  /* Fader value (0 to 255) */
    UBYTE   KeyerFlags;             /* Keyer flags */
    UBYTE   reserved[38];
};


/*** TBC Operations ***/

#define   TBCOF_STATUS      0x01    /* Fill in status field */
#define   TBCOF_MODES       0x02    /* Set modes/flags/muxes */
#define   TBCOF_ADJUST      0x04    /* Set adjustment values */


/*** TBC Status flags ***/

#define   TBCSF_MODULE      0x01    /* TBC module present */
#define   TBCSF_VIDEO       0x02    /* Video present at input */
#define   TBCSF_STABLE      0x04    /* Video input stable */


/*** TBC General flags ***/

#define   TBCGF_BYPASS      0x01    /* Bypass TBC */
#define   TBCGF_FREEZE      0x02    /* Freeze video */


/*** TBC Decoder flags ***/

#define   TBCDF_AGC         0x01    /* Enable AGC */
#define   TBCDF_CHROMAAGC   0x02    /* Enable chroma AGC */
#define   TBCDF_MONOCHROME  0x04    /* Monochrome input */


/*** TBC Encoder flags ***/

#define   TBCEF_BARS        0x01    /* Output 100% sat, 75% ampl bars */
#define   TBCEF_KILLCOLOR   0x02    /* Disable color on output */


/*** TBC InputSel values ***/

#define   TBCIN_YC          0       /* Flyer SVHS input */
#define   TBCIN_COMP        1       /* Flyer Composite input */
#define   TBCIN_TMAIN       2       /* Toaster Main output */
#define   TBCIN_FADER       3       /* TBC Fader output */
```

```
/*** TBC Termination ***/

#define   TBCTF_FADERA      0x04  /* Flyer A channel input */
#define   TBCTF_FADERB      0x08  /* Flyer B channel input */
#define   TBCTF_OUT         0x10  /* Video output */
#define   TBCTF_GENIN       0x20  /* Genlock input */
#define   TBCTF_COMPIN      0x80  /* Composite video input */

/*** TBC Keyer flags ***/

#define   TBCKF_KEYONB      0x01  /* Src (A or B) */
#define   TBCKF_MODE0       0x02  /* Keyer mode 0 */
#define   TBCKF_MODE1       0x04  /* Keyer mode 1 */
#define   TBCKF_FADEROUT    0x08  /* TBC/Fader mux (1 = Fader) */


/*** StartClipCutList flags ***/

#define   CCLF_DESTRUCTIVE  0x01  /* Delete unused portions of original clip */


/*** Structure used with FlyerAudioCtrl call ***/

struct FlyAudCtrl {
      UBYTE   Flags;              /* General flags */
      UBYTE   pad;
      UBYTE   LeftSense;          /* Left channel overrange detector (0 - 3) */
      UBYTE   RightSense;         /* Right channel overrange detector (0 - 3) */
      UBYTE   LeftSrc;            /* Left input source mux (see below) */
      UBYTE   RightSrc;           /* Right input source mux (see below) */
      UBYTE   LeftGain;           /* Left channel gain (0 or 1 to 16) */
      UBYTE   RightGain;          /* Right channel gain (0 or 1 to 16) */
      UBYTE   Aux1Mix;            /* Aux1 mix-in amount (-16 to 15, -128 to mute) */
      UBYTE   reserved1;
      UBYTE   Aux2Mix;            /* Aux2 mix-in amount (-16 to 15, -128 to mute) */
      UBYTE   reserved2;
      UBYTE   reserved[8];
};

/*** Audio Ctrl Operations ***/

#define   FACOF_SENSE       0x01  /* Report overrange information */
#define   FACOF_SETGAIN     0x02  /* Set input gain as specified */
#define   FACOF_SETSRC      0x04  /* Set input sources as specified */
#define   FACOF_SETMIX      0x08  /* Set aux mixing as specified */
#define   FACOF_SENSE8      0x10  /* Return 8 bit L/R readings in record mode */

/*** Audio Ctrl Input Sources ***/

#define   FACS_LINE1        0     /* RCA line-in jacks on rear of card */
#define   FACS_AUX1         1     /* Aux1 connector (JP19) */
#define   FACS_LINE2        2     /* Line2 connector (JP1) */

/*** FlyerOptions Non-Volatile Flags ***/

#define   FLYOPTF_DropFramDet   0x01  /* Stop recording on dropped frame */
#define   FLYOPTF_NOT_HQ5       0x02  /* =0 to enable HQ5 */

#endif    /* INC_FLYER_H */
```

***** FlyerLib.i 10/16/95 by Marty Flickinger**

```
**********************************************************
**          /inc/flyerlib.fd
**********************************************************
    IFND    _INC_FLYERLIB_I
_INC_FLYERLIB_I SET   1

_LVOAbortCmd            EQU     -30
_LVOCheckCmd            EQU     -36
_LVOWaitAction          EQU     -42
_LVOCheckAction         EQU     -48
_LVOAbortAction         EQU     -54
_LVOError2String        EQU     -60
_LVOInitFlyers          EQU     -66
_LVOFirmware            EQU     -72
_LVORunModule           EQU     -78
_LVOPgmFPGA             EQU     -84
_LVOSBusWrite           EQU     -90
_LVOSBusRead            EQU     -96
_LVOFIRinit             EQU     -102
_LVOFIRcustom           EQU     -108
_LVOFIRmapRAM           EQU     -114
_LVODSPboot             EQU     -120
_LVOGetFieldClock       EQU     -126
_LVOFlyerQuit           EQU     -132
_LVOPlayMode            EQU     -138
_LVORecordMode          EQU     -144
_LVOFlyerPlay           EQU     -150
_LVOFlyerRecord         EQU     -156
_LVOChangeAudio         EQU     -162
_LVOStartHeadList       EQU     -168
_LVOEndHeadList         EQU     -174
_LVOMakeClipHead        EQU     -180
_LVOVoidClipHead        EQU     -186
_LVOVoidCardHeads       EQU     -192
_LVOVoidAllHeads        EQU     -198
_LVOAudioParams         EQU     -204
_LVOBeginFindField      EQU     -210
_LVODoFindField         EQU     -216
_LVOEndFindField        EQU     -222
_LVOFindFieldAudio      EQU     -228
_LVOGetSMPTE            EQU     -234
_LVOVideoParams         EQU     -240
_LVOStillMode           EQU     -246
_LVOSetPlayMode         EQU     -252
_LVOSetRecMode          EQU     -258
_LVOSetNoMode           EQU     -264
_LVOToasterMux          EQU     -270
_LVOFlyerInputSel       EQU     -276
_LVOFlyerTermination    EQU     -282
_LVOSetFlooby           EQU     -288
_LVODefaults            EQU     -294
_LVOOpenReadField       EQU     -300
_LVOOpenWriteField      EQU     -306
_LVOCloseField          EQU     -312
_LVOFlyerReadLine       EQU     -318
_LVOFlyerWriteLine      EQU     -324
_LVOSetFillColor        EQU     -330
_LVOSkipLines           EQU     -336
_LVOSCSIreset           EQU     -342
_LVOSCSIinit            EQU     -348
```

```
_LVOFindDrives          EQU   -354
_LVOCopyData            EQU   -360
_LVOReqSense            EQU   -366
_LVOInquiry             EQU   -372
_LVOModeSelect          EQU   -378
_LVOModeSense           EQU   -384
_LVOReadSize            EQU   -390
_LVORead10              EQU   -396
_LVOWrite10             EQU   -402
_LVOSCSIseek            EQU   -408
_LVOFlyerSCSIdirect     EQU   -414
_LVOFlyerDriveCheck     EQU   -420
_LVOFlyerDriveInfo      EQU   -426
_LVOFlyerLocate         EQU   -432
_LVOFlyerFileInfo       EQU   -438
_LVOFlyerFreeGrip       EQU   -444
_LVOFlyerCopyGrip       EQU   -450
_LVOFlyerCmpGrips       EQU   -456
_LVOFlyerParent         EQU   -462
_LVOFlyerExamine        EQU   -468
_LVOFlyerDirList        EQU   -474
_LVOFlyerFileOpen       EQU   -480
_LVOFlyerFileClose      EQU   -486
_LVOFlyerFileSeek       EQU   -492
_LVOFlyerFileRead       EQU   -498
_LVOFlyerFileWrite      EQU   -504
_LVOFlyerCreateDir      EQU   -510
_LVOFlyerDelete         EQU   -516
_LVOFlyerRename         EQU   -522
_LVOFlyerRenameDisk     EQU   -528
_LVOFlyerFormat         EQU   -534
_LVOFlyerDeFrag         EQU   -540
_LVOFlyerSetBits        EQU   -546
_LVOFlyerSetDate        EQU   -552
_LVOFlyerSetComment     EQU   -558
_LVOFlyerWriteProt      EQU   -564
_LVOFlyerChangeMode     EQU   -570
_LVOMakeFlyerFile       EQU   -576
_LVOGetClipInfo         EQU   -582
_LVOFlyerCopyClip       EQU   -588
_LVOCPUwrite            EQU   -594
_LVOCPUread             EQU   -600
_LVOCPUDMA              EQU   -606
_LVODebugMode           EQU   -612
_LVOReadTest            EQU   -618
_LVOWriteTest           EQU   -624
_LVOSetFlyerTime        EQU   -630
_LVOFlyerStripAudio     EQU   -636
_LVOFlyerWriteCalib     EQU   -642
_LVOFlyerReadCalib      EQU   -648
_LVOWriteEEreg          EQU   -654
_LVOReadEEreg           EQU   -660
_LVOResetFlyer          EQU   -666
_LVOSetClockGen         EQU   -672
_LVOTeachFPGA           EQU   -678
_LVOFlyerRunning        EQU   -684
_LVOFlyerLoadVideo      EQU   -690
_LVOSetSerDevice        EQU   -696
_LVOFlyerSelfTest       EQU   -702
_LVOVideoCompressModes  EQU   -708
_LVOFIRquery            EQU   -714
_LVOGetClrSeqError      EQU   -720
```

```
_LVOLockFlyVolList      EQU   -726
_LVOUnLockFlyVolList    EQU   -732
_LVOTBCcontrol          EQU   -738
_LVOPauseAction         EQU   -744
_LVOStartClipCutList    EQU   -750
_LVOAddClipCut          EQU   -756
_LVOEndClipCutList      EQU   -762
_LVOEasyOpenWriteField  EQU   -768
_LVOFlyerAudioCtrl      EQU   -774
_LVOAppendFields        EQU   -780
_LVONewSequence         EQU   -786
_LVOAddSeqClip          EQU   -792
_LVOEndSequence         EQU   -798
_LVOPlaySequence        EQU   -804
_LVOFlyerOptions        EQU   -810
_LVOLocateField         EQU   -816
_LVOCacheTest           EQU   -822
_LVOFlyerCopyClipNew    EQU   -828
_LVOEndSequenceNew      EQU   -834
_LVOFlyerDeFragNew      EQU   -840
_LVOGetFrameHeader      EQU   -846
_LVOPutFrameHeader      EQU   -852

    ENDC    ; _INC_FLYERLIB_I
```

```
/* FlyerLib.h 10/16/95 by Marty Flickinger */

#include "dos/dos.h"

/****** Library Operations ******/
ULONG    AbortCmd(ULONG id);
ULONG    CheckCmd(ULONG id);
ULONG    WaitAction(struct ClipAction *action);
ULONG    CheckAction(struct ClipAction *action);
ULONG    AbortAction(struct ClipAction *action);
char  *Error2String(UBYTE error);

/****** Setup ******/
ULONG    InitFlyers(BPTR lock);
ULONG    Firmware(UBYTE board,ULONG length,APTR data,ULONG offset);
ULONG    RunModule(UBYTE board,ULONG length,APTR data,ULONG *ID,UWORD argc,ULONG *argv);
ULONG    PgmFPGA(UBYTE board,UBYTE chipnum,ULONG length,APTR data,UBYTE revision);
ULONG    SBusWrite(UBYTE board,UBYTE addr,UBYTE data);
ULONG    SBusRead(UBYTE board,UBYTE addr,UBYTE *data);
ULONG    FIRinit(UBYTE board,UWORD reg0,UWORD reg1);
ULONG    FIRcustom(UBYTE board,UBYTE prepost,UWORD scale,UWORD *coefdata);
ULONG    FIRmapRAM(UBYTE board,UBYTE bank,UBYTE scale,UBYTE shape);
ULONG    DSPboot(UBYTE board,ULONG length,APTR dataptr);
ULONG    GetFieldClock(ULONG *clockptr);
ULONG    FlyerQuit(int unit);
ULONG    PlayMode(UBYTE board);
ULONG    RecordMode(UBYTE board);

/****** Video Operations ******/
ULONG    FlyerPlay(struct ClipAction *clip);
ULONG    FlyerRecord(struct ClipAction *clip);
ULONG    ChangeAudio(struct ClipAction *clip);
ULONG    StartHeadList(UBYTE board);
ULONG    EndHeadList(UBYTE board,UBYTE flag);
ULONG    MakeClipHead(struct ClipAction *clip);
ULONG    VoidClipHead(struct ClipAction *clip);
ULONG    VoidCardHeads(UBYTE board);
ULONG    VoidAllHeads(void);
ULONG    AudioParams(void);
ULONG    BeginFindField(struct ClipAction *clip);
ULONG    DoFindField(struct ClipAction *clip);
ULONG    EndFindField(struct ClipAction *clip);
ULONG    FindFieldAudio(struct ClipAction *clip);
ULONG    GetSMPTE(UBYTE board,struct SMPTEinfo *info);

/****** Mode and Misc Operations ******/
ULONG    VideoParams(UBYTE board,UBYTE vidchan,UBYTE mintol,UBYTE maxtol,
      UBYTE rndfreq,UWORD vidlen,UBYTE FIRset,UBYTE special);
ULONG    StillMode(UBYTE board,UBYTE vidchan,UBYTE mode);
ULONG    SetPlayMode(UBYTE board);
ULONG    SetRecMode(UBYTE board);
ULONG    SetNoMode(UBYTE board);
ULONG    ToasterMux(UBYTE board,UBYTE input3,UBYTE input4,UBYTE preview);
ULONG    FlyerInputSel(UBYTE board,UBYTE video,UBYTE sync);
ULONG    FlyerTermination(UBYTE board,UBYTE flags);
ULONG    SetFlooby(UBYTE board,UBYTE chan,UBYTE item,ULONG value);
void  Defaults(struct ClipAction *clip);

/****** Direct Field Access ******/
ULONG    OpenReadField(struct ClipAction *action,ULONG field,UBYTE mode);
ULONG    OpenWriteField(struct ClipAction *action,ULONG field,UBYTE mode,struct VidCompInfo *);
ULONG    CloseField(struct ClipAction *action);
ULONG    FlyerReadLine(struct ClipAction *action,UBYTE *buffer);
ULONG    FlyerWriteLine(struct ClipAction *action,UBYTE *buffer);
ULONG    SetFillColor(struct ClipAction *action);
ULONG    SkipLines(struct ClipAction *action,UWORD lines);
```

```
/****** SCSI Operations ******/
ULONG   SCSIreset(UBYTE board);
ULONG   SCSIinit(struct FlyerVolume *volume);
ULONG   FindDrives(struct FlyerVolume *volume,APTR buffer);
ULONG   CopyData(struct FlyerVolume *src,struct FlyerVolume *dest,
        ULONG srcaddr,ULONG blocks,ULONG destaddr);
ULONG   ReqSense(struct FlyerVolume *volume,UBYTE bufsize,APTR buffer);
ULONG   Inquiry(struct FlyerVolume *volume,UBYTE bufsize,APTR buffer);
ULONG   ModeSelect(struct FlyerVolume *volume,UBYTE bufsize,APTR buffer,UBYTE PFbyte);
ULONG   ModeSense(struct FlyerVolume *volume,UBYTE bufsize,UBYTE page,APTR buffer);
ULONG   ReadSize(struct FlyerVolume *volume,ULONG *countptr,ULONG *lenptr);
ULONG   Read10(struct ClipAction *action,WORD blocks,ULONG lba,ULONG buffer);
ULONG   Write10(struct ClipAction *action,WORD blocks,ULONG buffer,ULONG lba);
ULONG   SCSIseek(struct FlyerVolume *volume,ULONG lba);
ULONG   FlyerSCSIdirect(UBYTE board,UBYTE unit,struct SCSICmd *scsiinfo,UBYTE structlen);

/****** FileSystem Interface ******/
ULONG   FlyerDriveCheck(struct FlyerVolume *vol);
ULONG   FlyerDriveInfo(struct FlyerVolume *vol,struct FlyerVolInfo *volume);
ULONG   FlyerLocate(struct ClipAction *clip);
ULONG   FlyerFileInfo(struct FlyerVolume *volume,struct ClipInfo *clipinfo);
ULONG   FlyerFreeGrip(struct FlyerVolume *volume,ULONG grip);
ULONG   FlyerCopyGrip(struct FlyerVolume *volume,ULONG grip,ULONG *newgrip);
ULONG   FlyerCmpGrips(struct FlyerVolume *volume,ULONG grip1,ULONG grip2);
ULONG   FlyerParent(struct FlyerVolume *volume,ULONG grip,ULONG *newgrip,ULONG *blockptr);
ULONG   FlyerExamine(struct FlyerVolume *volume,ULONG grip,struct ClipInfo *clipinfo);
ULONG   FlyerDirList(struct FlyerVolume *volume,ULONG grip,struct ClipInfo *clipinfo,
        UBYTE firstflag,UBYTE fsonlyflag);
ULONG   FlyerFileOpen(struct ClipAction *clip);
ULONG   FlyerFileClose(struct FlyerVolume *volume,ULONG fileid);
ULONG   FlyerFileSeek(struct FlyerVolume *volume,ULONG fileid,LONG pos,UBYTE mode,
        LONG    *newpos,LONG *oldpos);
ULONG   FlyerFileRead(struct FlyerVolume *volume,ULONG fileid,ULONG size,UBYTE *buffer,
        ULONG *actual);
ULONG   FlyerFileWrite(struct FlyerVolume *volume,ULONG fileid,ULONG size,UBYTE *buffer,
        ULONG *actual);
ULONG   FlyerCreateDir(struct ClipAction *clip);
ULONG   FlyerDelete(struct ClipAction *clip);
ULONG   FlyerRename(struct ClipAction *old,ULONG newgrip,char *newname);
ULONG   FlyerRenameDisk(struct FlyerVolume *volume,char *newname);
ULONG   FlyerFormat(struct FlyerVolume *volume,char *name,struct DateStamp *date,
        ULONG blocks,UBYTE flags);
ULONG   FlyerDeFrag(struct FlyerVolume *volume);
ULONG   FlyerSetBits(struct FlyerVolume *volume,ULONG grip,ULONG bits);
ULONG   FlyerSetDate(struct FlyerVolume *volume,ULONG grip,ULONG days,ULONG mins,ULONG ticks);
ULONG   FlyerSetComment(struct FlyerVolume *volume,ULONG grip,char *comment);
ULONG   FlyerWriteProt(struct FlyerVolume *volume,UBYTE value,UBYTE setflag,UBYTE *checkval);
ULONG   FlyerChangeMode(struct FlyerVolume *volume,ULONG grip,UBYTE access);
ULONG   MakeFlyerFile(struct FlyerVolume *volume,ULONG size,ULONG *startptr);
ULONG   GetClipInfo(struct FlyerVolume *volume,struct ClipInfo *clipinfo);
ULONG   FlyerCopyClip(struct FlyerVolume *srcvolume,struct FlyerVolume *destvolume);

/****** Flyer Testing ******/
ULONG   CPUwrite(UBYTE board,ULONG addr,UWORD data);
ULONG   CPUread(UBYTE board,ULONG addr,UWORD *data);
ULONG   CPUDMA(UBYTE board,ULONG cpuptr,ULONG dmaptr,UWORD length,UBYTE readflag);
ULONG   DebugMode(int unit,ULONG flags);
ULONG   ReadTest(struct FlyerVolume *volume,ULONG blocks,ULONG repeat,ULONG lba,UBYTE dblflag);
ULONG   WriteTest(struct FlyerVolume *volume,ULONG blocks,ULONG repeat,ULONG lba,UBYTE dblflag);

/****** Misc stuff ******/
ULONG   SetFlyerTime(struct DateStamp *datestamp);
ULONG   FlyerStripAudio(struct FlyerVolume *srcvolume,struct FlyerVolume *destvolume);
ULONG   FlyerWriteCalib(UBYTE board,UWORD item,WORD value,UBYTE saveflag);
ULONG   FlyerReadCalib(UBYTE board,UWORD item,WORD *valueptr);
ULONG   WriteEEreg(UBYTE board,UBYTE addr,UWORD data);
ULONG   ReadEEreg(UBYTE board,UBYTE addr,UWORD *dataptr);
ULONG   ResetFlyer(UBYTE board,ULONG flags);
```

```
ULONG    SetClockGen(UBYTE board,UBYTE clock,ULONG speed);
ULONG    TeachFPGA(UBYTE chipnum,ULONG length,APTR data);
ULONG    FlyerRunning(UBYTE board);
ULONG    FlyerLoadVideo(UBYTE board,APTR data,ULONG size);
ULONG    SetSerDevice(UBYTE board,UBYTE port,UBYTE type,UBYTE device);
ULONG    FlyerSelfTest(UBYTE board,UBYTE test,ULONG arg1,ULONG arg2,ULONG *result);
ULONG    VideoCompressModes(UBYTE board,UBYTE bestmode,UBYTE worstmode,UBYTE strategy);
ULONG    FIRquery(UBYTE board,UBYTE coefset,UBYTE prepost,UWORD *scale,UWORD *coefdata);
ULONG    GetClrSeqError(UBYTE board,UBYTE flag,UBYTE *doneptr,ULONG *userIDptr,
      ULONG *moreinfoptr);
APTR     LockFlyVolList(void);
ULONG    UnLockFlyVolList(APTR list);

/****** New for 4.0 ******/
ULONG    TBCcontrol(UBYTE board,struct TBCctrl *ptr,UBYTE oper);
ULONG    PauseAction(struct ClipAction *action,UBYTE pauseflag);
ULONG    StartClipCutList(struct ClipAction *clip,UBYTE flags);
ULONG    AddClipCut(struct ClipAction *subclip);
ULONG    EndClipCutList(UBYTE doit);
ULONG    EasyOpenWriteField(struct ClipAction *action,ULONG field,UBYTE mode,UBYTE quality);
ULONG    FlyerAudioCtrl(UBYTE board,struct FlyAudCtrl *ptr,UBYTE oper);
ULONG    AppendFields(struct ClipAction *action);

/****** New for 4.05 ******/
ULONG    NewSequence(UBYTE board);
ULONG    AddSeqClip(struct ClipAction *clip);
ULONG    EndSequence(UBYTE board,UBYTE doit);
ULONG    PlaySequence(UBYTE board,ULONG basetime);
ULONG    FlyerOptions(UBYTE board,UBYTE setflag,ULONG *options);
ULONG    LocateField(struct ClipAction *clip);
ULONG    CacheTest(UBYTE board);

/****** New for 4.1 ******/
ULONG    FlyerCopyClipNew(struct ClipAction *srcaction,struct FlyerVolume *destvolume);
ULONG    EndSequenceNew(struct ClipAction *action,UBYTE doit);
ULONG    FlyerDeFragNew(struct ClipAction *action);
ULONG    GetFrameHeader(struct ClipAction *action,APTR buffer);
ULONG    PutFrameHeader(struct ClipAction *action,APTR buffer);


/*PRAGMAS*/

/*------ Library Operations ----------------------------------------*/
#pragma libcall FlyerBase AbortCmd 1e 001
#pragma libcall FlyerBase CheckCmd 24 001
#pragma libcall FlyerBase WaitAction 2a 801
#pragma libcall FlyerBase CheckAction 30 801
#pragma libcall FlyerBase AbortAction 36 801
#pragma libcall FlyerBase Error2String 3c 001

/*------ Setup ----------------------------------------------------*/
#pragma libcall FlyerBase InitFlyers 42 001
#pragma libcall FlyerBase Firmware 48 281004
#pragma libcall FlyerBase RunModule 4e A2981006
#pragma libcall FlyerBase PgmFPGA 54 3821005
#pragma libcall FlyerBase SBusWrite 5a 21003
#pragma libcall FlyerBase SBusRead 60 81003
#pragma libcall FlyerBase FIRinit 66 21003
#pragma libcall FlyerBase FIRcustom 6c 821004
#pragma libcall FlyerBase FIRmapRAM 72 321004
#pragma libcall FlyerBase DSPboot 78 81003
#pragma libcall FlyerBase GetFieldClock 7e 801
#pragma libcall FlyerBase FlyerQuit 84 001
#pragma libcall FlyerBase PlayMode 8a 001
#pragma libcall FlyerBase RecordMode 90 001
```

```
/*------ Video/Audio Operations ----------------------------------------*/
#pragma libcall FlyerBase FlyerPlay 96 801
#pragma libcall FlyerBase FlyerRecord 9c 801
#pragma libcall FlyerBase ChangeAudio a2 801
#pragma libcall FlyerBase StartHeadList a8 001
#pragma libcall FlyerBase EndHeadList ae 1002
#pragma libcall FlyerBase MakeClipHead b4 801
#pragma libcall FlyerBase VoidClipHead ba 801
#pragma libcall FlyerBase VoidCardHeads c0 001
#pragma libcall FlyerBase VoidAllHeads c6 0
#pragma libcall FlyerBase AudioParams cc 0
#pragma libcall FlyerBase BeginFindField d2 801
#pragma libcall FlyerBase DoFindField d8 801
#pragma libcall FlyerBase EndFindField de 801
#pragma libcall FlyerBase FindFieldAudio e4 801
#pragma libcall FlyerBase GetSMPTE ea 8002

/*------ Mode and Misc Operations --------------------------------------*/
#pragma libcall FlyerBase VideoParams f0 7654321008
#pragma libcall FlyerBase StillMode f6 21003
#pragma libcall FlyerBase SetPlayMode fc 001
#pragma libcall FlyerBase SetRecMode 102 001
#pragma libcall FlyerBase SetNoMode 108 001
#pragma libcall FlyerBase ToasterMux 10e 321004
#pragma libcall FlyerBase FlyerInputSel 114 21003
#pragma libcall FlyerBase FlyerTermination 11a 1002
#pragma libcall FlyerBase SetFlooby 120 321004
#pragma libcall FlyerBase Defaults 126 801

/*------------ Direct Field Access -----------**/
#pragma libcall FlyerBase OpenReadField 12c 10803
#pragma libcall FlyerBase OpenWriteField 132 910804
#pragma libcall FlyerBase CloseField 138 801
#pragma libcall FlyerBase FlyerReadLine 13e 9802
#pragma libcall FlyerBase FlyerWriteLine 144 9802
#pragma libcall FlyerBase SetFillColor 14a 801
#pragma libcall FlyerBase SkipLines 150 0802

/*------ SCSI Operations -----------------------------------------------*/
#pragma libcall FlyerBase SCSIreset 156 001
#pragma libcall FlyerBase SCSIinit 15c 801
#pragma libcall FlyerBase FindDrives 162 9802
#pragma libcall FlyerBase CopyData 168 2109805
#pragma libcall FlyerBase ReqSense 16e 90803
#pragma libcall FlyerBase Inquiry 174 90803
#pragma libcall FlyerBase ModeSelect 17a 190804
#pragma libcall FlyerBase ModeSense 180 910804
#pragma libcall FlyerBase ReadSize 186 A9803
#pragma libcall FlyerBase Read10 18c 210804
#pragma libcall FlyerBase Write10 192 210804
#pragma libcall FlyerBase SCSIseek 198 0802
#pragma libcall FlyerBase FlyerSCSIdirect 19e 281004

/*------ FileSystem Interface ------------------------------------------*/
#pragma libcall FlyerBase FlyerDriveCheck 1a4 801
#pragma libcall FlyerBase FlyerDriveInfo 1aa 9802
#pragma libcall FlyerBase FlyerLocate 1b0 801
#pragma libcall FlyerBase FlyerFileInfo 1b6 9802
#pragma libcall FlyerBase FlyerFreeGrip 1bc 0802
#pragma libcall FlyerBase FlyerCopyGrip 1c2 90803
#pragma libcall FlyerBase FlyerCmpGrips 1c8 10803
#pragma libcall FlyerBase FlyerParent 1ce A90804
#pragma libcall FlyerBase FlyerExamine 1d4 90803
#pragma libcall FlyerBase FlyerDirList 1da 2190805
#pragma libcall FlyerBase FlyerFileOpen 1e0 801
#pragma libcall FlyerBase FlyerFileClose 1e6 0802
#pragma libcall FlyerBase FlyerFileSeek 1ec A9210806
#pragma libcall FlyerBase FlyerFileRead 1f2 A910805
```

```
#pragma libcall FlyerBase FlyerFileWrite 1f8 A910805
#pragma libcall FlyerBase FlyerCreateDir 1fe 801
#pragma libcall FlyerBase FlyerDelete 204 801
#pragma libcall FlyerBase FlyerRename 20a 90803
#pragma libcall FlyerBase FlyerRenameDisk 210 9802
#pragma libcall FlyerBase FlyerFormat 216 10A9805
#pragma libcall FlyerBase FlyerDeFrag 21c 801
#pragma libcall FlyerBase FlyerSetBits 222 10803
#pragma libcall FlyerBase FlyerSetDate 228 3210805
#pragma libcall FlyerBase FlyerSetComment 22e 90803
#pragma libcall FlyerBase FlyerWriteProt 234 910804
#pragma libcall FlyerBase FlyerChangeMode 23a 10803
#pragma libcall FlyerBase MakeFlyerFile 240 90803
#pragma libcall FlyerBase GetClipInfo 246 9802
#pragma libcall FlyerBase FlyerCopyClip 24c 9802

/*------ Flyer Testing ------------------------------------------------*/
#pragma libcall FlyerBase CPUwrite 252 18003
#pragma libcall FlyerBase CPUread 258 98003
#pragma libcall FlyerBase CPUDMA 25e 2198005
#pragma libcall FlyerBase DebugMode 264 1002
#pragma libcall FlyerBase ReadTest 26a 3210805
#pragma libcall FlyerBase WriteTest 270 3210805

/*------------ Misc stuff ----------------*/
#pragma libcall FlyerBase SetFlyerTime 276 801
#pragma libcall FlyerBase FlyerStripAudio 27c 9802
#pragma libcall FlyerBase FlyerWriteCalib 282 321004
#pragma libcall FlyerBase FlyerReadCalib 288 81003
#pragma libcall FlyerBase WriteEEreg 28e 21003
#pragma libcall FlyerBase ReadEEreg 294 81003
#pragma libcall FlyerBase ResetFlyer 29a 1002
#pragma libcall FlyerBase SetClockGen 2a0 21003
#pragma libcall FlyerBase TeachFPGA 2a6 81003
#pragma libcall FlyerBase FlyerRunning 2ac 001
#pragma libcall FlyerBase FlyerLoadVideo 2b2 18003
#pragma libcall FlyerBase SetSerDevice 2b8 321004
#pragma libcall FlyerBase FlyerSelfTest 2be 8321005
#pragma libcall FlyerBase VideoCompressModes 2c4 321004
#pragma libcall FlyerBase FIRquery 2ca 9821005
#pragma libcall FlyerBase GetClrSeqError 2d0 A981005
#pragma libcall FlyerBase LockFlyVolList 2d6 0
#pragma libcall FlyerBase UnLockFlyVolList 2dc 801

/*------------ New for 4.0 ----------------*/
#pragma libcall FlyerBase TBCcontrol 2e2 18003
#pragma libcall FlyerBase PauseAction 2e8 0802
#pragma libcall FlyerBase StartClipCutList 2ee 0802
#pragma libcall FlyerBase AddClipCut 2f4 801
#pragma libcall FlyerBase EndClipCutList 2fa 001
#pragma libcall FlyerBase EasyOpenWriteField 300 210804
#pragma libcall FlyerBase FlyerAudioCtrl 306 18003
#pragma libcall FlyerBase AppendFields 30c 801

/*------------ New for 4.05 ----------------*/
#pragma libcall FlyerBase NewSequence 312 001
#pragma libcall FlyerBase AddSeqClip 318 801
#pragma libcall FlyerBase EndSequence 31e 1002
#pragma libcall FlyerBase PlaySequence 324 1002
#pragma libcall FlyerBase FlyerOptions 32a 81003
#pragma libcall FlyerBase LocateField 330 801
#pragma libcall FlyerBase CacheTest 336 001
/*------------ New for 4.1 ----------------*/
#pragma libcall FlyerBase FlyerCopyClipNew 33c 9802
#pragma libcall FlyerBase EndSequenceNew 342 0802
#pragma libcall FlyerBase FlyerDeFragNew 348 801
#pragma libcall FlyerBase GetFrameHeader 34e 9802
#pragma libcall FlyerBase PutFrameHeader 354 9802
```

****
*** Extra Flyer Calling Docs 23 Aug 1994 ***
****

**** Description of the ClipAction structure ****

```
STRUCTURE ClipAction,0
    APTR    ca_Volume
    APTR    ca_ID
    UBYTE   ca_ReturnTime
    UBYTE   ca_Channel
    UBYTE   ca_Flags
    UBYTE   ca_PermissFlags
    ULONG   ca_VidStartField
    ULONG   ca_AudStartField
    ULONG   ca_VidFieldCount
    ULONG   ca_AudFieldCount
    ULONG   ca_GoClock
```

ca_Volume — Pointer to a FlyerVolume structure (see below)

ca_ID — References Flyer operations going on asynchronously. Can be pulled out and stored, then re-inserted into a ClipAction structure which is then passed to WaitAction, AbortAction, or CheckAction. When the operation is complete, some results can be found in the ClipAction structure as well.

ca_ReturnTime — Specifies when call should return. See RT_xxx values. Note that some calls cannot return until complete (these will ignore this field). Default value is RTT_STOPPED.

ca_Channel — Which hardware video channel to use.

Channel #	Hardware
0	Flyer output A (Toaster input 3)
1	Flyer output B (Toaster input 4)

Default value is channel 0.

ca_Flags — Various control flags:

CAF_VIDEO	Include video in action
CAF_AUDIOL	Include left audio in action
CAF_AUDIOR	Include right audio in action
CAF_USEMATTE	After play complete, display matte color given in MatteY,I,Q fields
CAF_NOPREROLL	OBSOLETE
CAF_APPEND	OBSOLETE -- use AppendFields() function
CAF_REPROCESS	Skip optimizations for CutList operations

Default value is CAF_VIDEO!CAF_AUDIOL!CAF_AUDIOR

ca_PermissFlags — Various permission flags:

CAPF_STEALOURVIDEO
Can steal requested video channel in order to successfully perform new action specified.

CAPF_KILLOTHERVIDEO
Can kill video on other channel(s) if needed to successfully perform new action specified.

CAPF_ERRIFBUSY
If set, will return an error if the Flyer cannot perform the specified action immediately. If not set, will wait for needed resources/channels to free up and then will begin action.

CAPF_AUTOMUTE
With DoFindField, mute audio after played once

CAPF_USEHEADS OBSOLETE

ca_VidStartField
ca_AudStartField
When playing a clip, these values specify the start field number at which to begin (for video and audio). Think of this as an offset field number "into the clip". The first field number of a recorded clip is always 0. Default = 0.

ca_VidFieldCount
ca_AudFieldCount
These specify limits on the number of fields to play or record, for both audio and video. A value of 0 means "unlimited". Default = 0.

ca_GoClock
Specifies the field clock value (the Flyer's) at which to begin playing or recording. A value of 0 means ASAP. Default value is 0.

UWORD ca_MatteY
BYTE ca_MatteI
BYTE ca_MatteQ

Define the matte color Flyer will put up when a clip has finished playing (if CAF_USEMATTE set) - Only BLACK yet! Also used with SetFillColor() - all values supported.

UWORD ca_VolAttack

Attack time of the volume envelope in fields. Default value is 0.

UWORD ca_VolSust1
UWORD ca_VolSust2

These specify the "plateau" level of the volume envelope (for each audio channel). $0000 is softest, $FFFF is loudest. Default value is $FFFF.

UWORD ca_VolDecay

Decay time of the volume envelope in fields. Default value is 0.

WORD ca_AudioPan1
WORD ca_AudioPan2

Pan position for audio
-$8000 is full left
$0000 is center
$7FFF is full right
All intermediate values supported too. Default is 0.

ULONG ca_TotalAudStart OBSOLETE
ULONG ca_TotalAudLength OBSOLETE

ULONG ca_StartBlk

If no name is specified in the "fv_Path" string, this value specifies a "raw" block number at which to start a clip playing or recording. This field is also updated with the actual block number found after a call that locates a specific field. Default value is 0.

ULONG ca_EndBlk

If no name is specified in the "fv_Path" string, this value specifies a "raw" block number at which to stop playing or recording. Default value is 0.

ULONG ca_Grip \
ULONG ca_FileID These are used privately by the FlyerFileSystem
UBYTE ca_Access /

ULONG ca_FldHandle

Reference number for an open field for R/W operations. Set for you by OpenReadField/OpenWriteField calls and used by FlyerWriteLine/FlyerReadLine/CloseField calls.

ULONG ca_LastFieldDone

Indicates the last field number played or recorded. Valid only when the action is complete (stopped or was aborted). For recording, this value = number of fields recorded - 1. For playing, this value will depend on the starting field number and how many fields were done.

****** Description of the FlyerVolume structure ******

STRUCTURE FlyerVolume,0

APTR fv_Path

Pointer to a null-terminated string which contains a volume name, clip name, or both. If ptr is NULL, or if no volume is specified, then next three values must be set properly to specify the Flyer card to talk to and the exact SCSI channel & drive numbers. Default value is NULL.

UBYTE fv_Board

Flyer board number (0...n). Not used if volume is specified in fv_Path string. Default = 0.

UBYTE fv_SCSIdrive

SCSI drive number. This is a combination of the channel number and unit number, and ranges from 0 to 23. Usually filled in for caller if volume is specified in fv_Path string. Default = 0.

UBYTE fv_Flags

FVF_USENUMS - Specifies that fv_SCSIdrive number shall override any volume found in the fv_Path string.

```
/*****************************************************************************\
*
* FlyerClipFmt.h - Flyer clip format
*
* $Id: FlyerClipFmt.h,v 1.0 1995/07/11 11:03:25 Flick Exp $
*
* $Log: FlyerClipFmt.h,v $
*Revision 1.0  1995/07/11  11:03:25  Flick
*FirstCheckIn
*
*
* Copyright (c) 1995 NewTek, Inc.
* Confidental and Proprietary. All rights reserved.
*
*    07/11/95        Marty                Created
\*****************************************************************************/


struct ClipHeader {
      ULONG  ID;                  // "CLIP" Identifies this block as the Clip header
      ULONG  Length;              // Length of clip in logical blocks (all components)
      UBYTE  Version;             // Flyer version that created file (4)
      UBYTE  VidCmpVer;           // Version of Video Compression chips used (2)
      UBYTE  AudVer;              // Audio data format/version (2)
      UBYTE  VidGrade;            // Video grade (std/HQ5)
      ULONG  Fields;              // Number of fields contained in clip (1/60 sec each)
      ULONG  DataStart;           // Block offset: clip hdr to first frame header
      ULONG  DataEnd;             // Block offset: clip hdr to last data blk of last frm+1
      ULONG  Tail;                // Block offset: clip hdr to Tail Header
      UBYTE  reserved2;
      UBYTE  AudioLength;         // No. logical blocks used per mono channel per field (3)
      UBYTE  VidFlag;             // Flag: includes video data (0 = no, 1 = yes)
      UBYTE  AudioChans;          // Number of mono audio channels included (0 or more)
      UBYTE  Pedestal;            // Digital pedestal value for playback (60)
      UBYTE  reserved3;
      WORD   CustCoefs[9];        // Custom FIR coefficients/scale used
      WORD   CustInvCoefs[9];     // Custom FIR coefficients/scale to use for playback
      UWORD LogBlkSize;           // Logical block size (512)
      UBYTE  FIRROM;              // FIR ROM lookup table used
      UBYTE  FIRRAM;              // FIR RAM lookup table used
      UBYTE  reserved4[438];
 };
```

```
struct Field {
    UWORD VidStart;          // Block offset: frm hdr to start of field's video data
    UWORD VidLength;         // Logical block length of this field's video data
    UBYTE  Tolerance;        // Compression error tolerance
    UBYTE  RndSize;          // Amplitude of noise generation
    UBYTE  RndFreq;          // Frequency of noise generation
    UBYTE  RndSeed;
    UWORD reserved1;
    UWORD reserved2;
    UBYTE  Flags;            // Private flags
    UBYTE  reserved3;
    UBYTE  reserved4[18];
    UBYTE  FIRpreset;        // FIR filter preset for recorded video data this field
    UBYTE  SerType;          // Type of serial device for recorded serial data
    UBYTE  SerMake;          // Make of serial device for recorded serial data
    UBYTE  SerialDataLen;    // Byte length of serial data captured for this field
    UBYTE  SerData[36];      // Serial data buffer (data recorded this field)
};


struct FrameHeader {
    ULONG  ID;               // "CFRM" Identifies this block as a frame header
    ULONG  Length;           // Length of this frame (including header)
    UBYTE  Version;          // Flyer version that created file (4)
    UBYTE  VidCmpVer;        // Version of Video Compression chips used (2)
    UBYTE  AudVer;           // Audio data format/version (2)
    UBYTE  reserved1;
    LONG   PrevFrame;        // Block offset from this to previous frame header
    LONG   NextFrame;        // Block offset from this to next frame header
    ULONG  FrameNumber;      // Ordinal color frame number (0 is first)
    UBYTE  reserved2;
    UBYTE  AudioLength;      // No. logical blocks used per mono channel per field (3)
    UBYTE  VidFlag;          // Flag: includes video data (0 = no, 1 = yes)
    UBYTE  AudioChans;       // Number of mono audio channels included (0 or more)
    UBYTE  reserved3[196];
    struct Field    fields[4];   // Info for each field in this frame
};

#define  GRADE_STD   0
#define  GRADE_HQ5   1

#define  ID_CLIP         0x434C4950
#define  ID_CFRM         0x4346524D
```

Format of Audio/Video data

Data between frame headers appears in the following format

If AudioChans >= 1...

Channel 1 Audio data for field 1
Channel 1 Audio data for field 2
Channel 1 Audio data for field 3
Channel 1 Audio data for field 4

If AudioChans = 2...

Channel 2 Audio data for field 1
Channel 2 Audio data for field 2
Channel 2 Audio data for field 3
Channel 2 Audio data for field 4

If VidFlag is TRUE...

Video data for field 1
Video data for field 2
Video data for field 3
Video data for field 4

Each chunk of data is always written in an integral number of logical blocks, which means that currently data is padded to the nearest whole SCSI sector (512 bytes).

---------- jmf ---------- 10 Feb 95 ----------

```
;Flyer PlayClip Programming Example
;by Daniel Wolf 7/3/95
;(c) Copyright 1995 by NewTek, Inc.
;use the Macro68 Assembler

 EXEOBJ
 RELAX
 MC68000
 NOCOMMENTMARKERS

 JMP _START                              ;jump past includes, etc.

;**************** Preliminaries

 INCLUDE "EXEC.I"                        ;handles the following structure
 INCLUDE "FLYER.I"                       ;and also necessary for flyer.i

SOFFSET SET 0                            ;structure for local variables
 LONG _STACK
 LONG _DOSBASE
 LONG _FLYBASE
 LONG COMMAND
 LONG STDOUT
 LONG ClipSize
SOFFSET SET 0

;*** Flyer Lib Offsets Used Here

_LVOPlayMode            EQU    -138
_LVOFlyerPlay           EQU    -150
_LVOGetClipInfo         EQU    -582
_LVOToasterMux          EQU    -270

;*** Amiga Exec and DOS Lib Offsets

LVO.CLOSELIBRARY        EQU $FFFFFE62
LVO.OPENLIBRARY         EQU $FFFFFDD8

LVO.OUTPUT              EQU $FFFFFFC4
LVO.WRITE               EQU $FFFFFFD0

;*** Some Useful Macros

FLYLIB MACRO
 move.l _FLYBASE(a5),a6
 jsr _LVO\1(a6)
 ENDM

DOSLIB MACRO
 move.l _DOSBASE(a5),a6
 jsr LVO.\1(a6)
 ENDM

JUST MACRO
 jsr LVO.\1(a6)
 ENDM
```

```
DOSPRINT MACRO                          ;*filehandle (d1),*buff (d2), [len (d3)]
 movem.l d0-d3/a0-a6,-(SP)
 move.l \1,d1
 move.l \2,d2
 move.l d2,a0                           ;put length of null-terminated
CALC\@                                  ;string into d3 for DOS WRITE command
 tst.b (a0)+
 bne.s CALC\@
 move.l a0,d3
 sub.l d2,d3
 subq.l #1,d3
 DOSLIB WRITE
 movem.l (SP)+,d0-d3/a0-a6
 ENDM

;**************** Program Code

_START                                  ;Amiga Startup Stuff
 lea VARIABLES,a5
 move.l SP,_STACK(a5)                   ;save stack pointer
 move.l $4,a6
 move.l a0,COMMAND(a5)                  ;save ptr to the clip name!
 cmpi.l #4,d0                           ;command at least 4 characters long!
 bmi _STARTERROR
 clr.b -1(a0,d0.W)                      ;null-terminate the command line
 lea _DOSNAME,a1
 moveq #34,d0
 JUST OPENLIBRARY
 move.l d0,_DOSBASE(a5)                 ;check for successful open
 beq _STARTERROR
 lea _FLYNAME,a1
 moveq #0,d0
 JUST OPENLIBRARY
 move.l d0,_FLYBASE(a5)
 beq _STARTERROR
 DOSLIB OUTPUT
 move.l d0,STDOUT(a5)                   ;set CLI window as OUTPUT file handle

NOWDOMAIN
 DOSPRINT STDOUT(a5),#TITLE             ;print out program title
 DOSPRINT STDOUT(a5),COMMAND(a5)        ;print out clip name from command line
 DOSPRINT STDOUT(a5),#TWOLINE           ;do 2 linefeeds and carriage returns

 jsr PLAYIT                             ;do the Flyer routines

_ERROR
 move.l d0,-(SP)                        ;clean up and exit to AmigaDos
 move.l $4,a6
 move.l _DOSBASE(a5),d0
 beq.s 5$
 move.l d0,a1
 JUST CLOSELIBRARY
5$
 move.l _FLYBASE(a5),d0
 beq.s 9$
 move.l d0,a1
 JUST CLOSELIBRARY
9$
 move.l (SP)+,d0
 move.l _STACK(a5),SP                   ;restore stack pointer
 rts                                    ;exit back to where we came from!

_STARTERROR
 moveq #$ffffffff,d0
 bra _ERROR
```

```
;**************** Flyer Code

PLAYIT
 move.w #FVI_sizeof,FVI                    ;tell FlyerVolumeInfo how long it is!

 moveq #0,d0                               ;Flyer board #0
 FLYLIB PlayMode                           ;reset Flyer to Play mode
 tst.l d0
 bne DONE                                  ;bummer - it wont play at all

 moveq.l #0,d0
 moveq.l #1,d1
 moveq.l #1,d2
 moveq.l #0,d3
 FLYLIB ToasterMux                         ;set to play on Toaster Main 3

 bsr GetClipSize
 tst.l d0
 bne DONE

 lea CA,a0                                 ;fill the ClipAction Structure!
 move.l #FV,(a0)                           ;tell ClipAction its FlyerVolume
 move.b #$1F,ca_Flags(a0)                  ;play everything
 move.b #$3,ca_PermissFlags(a0)            ;generous permission level
 moveq #0,d0
 move.b d0,ca_Channel(a0)
 move.w d0,ca_MatteY(a0)                   ;set black as Matte YIQ values
 move.b d0,ca_MatteI(a0)
 move.b d0,ca_MatteQ(a0)
 move.l d0,ca_VidStartField(a0)            ;starting video field = 0
 move.l d0,ca_AudStartField(a0)            ;starting audio field = 0
 move.l ClipSize(a5),d0
 move.l d0,ca_VidFieldCount(a0)            ;play all playable fields
 move.l d0,ca_AudFieldCount(a0)
 move.w #32767,ca_VolSust1(a0)             ;audio volume at half maximum
 move.w #32767,ca_VolSust2(a0)
 move.b #RT_IMMED,ca_ReturnTime(a0)        ;command returns without waiting!

 FLYLIB FlyerPlay

 tst.l d0
 bne PlayError
DONE
 rts

GetClipSize
 lea FV,a0
 move.l COMMAND(a5),fv_Path(a0)            ;clip full volume path and filename
 move.b #0,fv_Board(a0)                    ;board 0
 move.b #0,fv_SCSIdrive(a0)                ;scsi drive #0 on this channel
 move.b #0,fv_Flags(a0)                    ;no special flags
 lea CI,a1
 move.w #CI_sizeof,ci_len(a1)              ;tell ClipInfo how big itself is so
 FLYLIB GetClipInfo                        ;GetClipInfo knows how much to fill
 tst.l d0
 bne FlyError
 lea CI,a1
 move.l ci_Fields(a1),ClipSize(a5)
 subq.l #1,ClipSize(a5)                    ;playable fields = ClipSize - 1
 rts

FlyError
 moveq #1,d0                               ;flyer library returned error
 rts

PlayError
 moveq #2,d0                               ;cant do a FlyerPlay properly
 rts
```

```
;**************** Data Section

DATA

CNOP 0,4                                ;long word align just for good measure and on general principles

_FLYNAME    DC.B 'flyer.library',0
EVEN
_DOSNAME    DC.B 'dos.library',0
EVEN

TITLE       DC.B 13,10,' FlyerPlay by D. Wolf    1995 by NewTek'
TWOLINE     DC.B 10,13,10,13,0,0,0,0

CNOP 0,4

VARIABLES
    dx.l 10                             ;private (STACK, COMMAND, CMDLEN, etc.)
CNOP 0,4

FVI                                     ;Flyer VolumeInfo Structure
    dx.b FVI_sizeof
CNOP 0,4

CI                                      ;Flyer ClipInfo Structure
    dx.b CI_sizeof
CNOP 0,4

CA                                      ;Flyer ClipAction Structure
    dx.b CA_sizeof
CNOP 0,4

FV                                      ;Flyer Volume Structure
    dx.b FV_sizeof

END
```

Editor ARexx Documentation Oct 16 1995 Editor version 4.1

The Video Toaster's Editor interface is controlled through an entirely separate program, Edit. It has its own ARexx port, which is configured as a function host. The port name is PROJECT_REXX_PORT, and the functions it responds to are documented below. Although arexx programs can be executed (asynchronously) from within a sequence, the editor itself is preoccupied with its sequencing duties, and will not respond to any rexx messages sent to it until after the sequence is done playing. The upshot of this is that you must put your rexx croutons at the end of a sequence if you want them to talk to the editor. Have fun.

"CROUTONNAME"
Returns crouton name
ARGS: [Row,Column]

"CROUTONSPOT"
Return position in grid for current crouton
ARGS: NONE

"CROUTONTYPE"
Return Crouton Type
ARGS: [coords]

"CROUTONSINPROJECT"
Return total number of croutons in the current project.

"CROUTONPICK"
Pick Crouton -- Selects Crouton
ARGS: 0 args ==> deselect allRow,Column or no args to
ARGS: 1 args ==> Number OR "FIRST" or "LAST"
ARGS: 2 args ==> Row,Column

"CROUTONLOAD"
Load Crouton -- Adds named crouton to end of project
ARGS: CroutonName

"CROUTONDELETE"
Deletes currently selected crouton
ARGS: NONE

"CROUTONTIME"
Returns the start time for the selected crouton, measured from the start of the project. Value is in fields. This value is useful for locking a crouton (using the DELAY tag).
ARGS: NONE

"CROUTONRUN"
Run Crouton without adding to project
ARGS: CroutonName

"CROUTONSELECTED"
Return 1 if crouton is selected.
ARGS: [cords]

"CROUTONSTOP"
Aborts currently playing crouton (clip)
ARGS: none

"CROUTONGETTAG"
Return Tag value, 0 if tag doesn't exist OR tag has value of 0
ARGS: Tag name

"CROUTONSETTAG"
Set crouton tag value
ARGS: Tag name, Value

"CROUTONCHECKTAG"
Check existance of tag in crouton, return size
ARGS: Tag name

NOTE: It appears that this function, now debugged, always succeeds - so please check tag validity with CROUTONTYPE, etc. before applying some irrelevant or illegal tag. For example, use CROUTONTYPE to identify that a crouton is a Framestore so you don't foolishly apply a AUDIOVOLUME1 to it!

"PROJECTLOAD"
Load Project -- Load crouton into project and place at the end. If INSERT keyword is included, will instead insert after the currently selected crouton (if none is selected, will be placed at the beginning).
ARGS: Project name (full path) [INSERT]

"PROJECTSAVE"
Save Project
ARGS: Project name (full path)

"PROJECTPLAY"
Play Project
ARGS: none

"PROJECTUPDATE"
Update Project after modifying croutons or tags (updates total running time display)
ARGS: none

"FLYERDRIVE"
Return flyer drive name based on index in internal list, or ""
ARGS: Drive number, starting at 0

"FLYERSTATUS"
Return flyer recording status/error code, useful to determine if Flyer has failed/stopped recording, and if so why. The codes are listed below (FERR_xxx).
ARGS: NONE

"FLYEROUT"
Adjust Toaster settings for inputs 3 and 4
ARGS:[value] (if value is omitted, current setting is returned)
Setting is bit mask for inputs 3(bit 0) and 4(bit 1). Bit set means Flyer has that input, 0 means it is live toaster input.

"RECORDADD"
Record fields into a flyer clip, will append fields into an existing clip, or create a new one. The clip is always closed and usable after this command. Although the flyer can only record clips in whole (4-field) NTSC color-frames, this command will copy the correct fields into the clip, temporarily padding the clip on the end to make a complete color-frame if necessary.
ARGS: ClipName, # of Fields [, Source, Compression Mode]

Note: omitting source or mode will use last selected mode, note that all clips must have full flyer VOLUME name included.

"RECORDCLIP"
Record a named flyer clip, won't overwrite existing clip w/same name. This places Flyer into record mode. When finished recording, you MUST place it back into play mode. To do this, use RECORDSTOP (even if you detect with FLYERSTATUS that recording has stopped on its own).
ARGS: ClipName [Name, # of Fields, Source, Compression Mode]
if fields=0, recording continues 'til drive is full omitting source or mode will use last selected mode, note that all clips must have full flyer VOLUME name included. This command will probably not be able to record clips with an uneven number of color-frames.

"RECORDPAUSE"
Pause Recording
ARGS: Pause=1 for Pause, 0 for resume

"RECORDSTOP"
Stop Recording and put Flyer back into play mode. You MUST do this even if the Flyer stops recording on its own, or the Editor will NOT be able to play clips or do sequencing!
ARGS: none

"MAKEICON"
Create an icon for a flyer clip, optionally make 'Flyer Still' icon for 4=field clips by adding 'STILL' keyword.
ARGS: ClipName [, Field, ['STILL']]

"REQ_BUTTONS"
Put up requester with title and up to 4 labelled buttons. "Defaults" is a string of 0/1's specifying initial button states. Returns string of 0/1's for each final button state or "CANCEL" if user cancels requester. Strings are only as long as the number of buttons being used; first digit is for the first button.
ARGS: Title, Defaults, label1,[label2,[label3,[label4]]]

"REQ_ERROR"
Display error message at top of screen
ARGS: String

"REQ_FLYERJOG"
Put up a Flyer clip jog/shuttle requester
ARGS: Title, Pathname, [points]

Put up requester with a slider to allow user to jog/shuttle thru a Flyer clip to pick a specific time. This works by just specifying a pathname (volume:name) for the clip. It does not need to be in the current sequence. You MUST include the volume name, and it must NOT be the device name. In otherwords "FlyerA0:abc" is proper, but "FA0:abc" is not.

"Points" can be 1 or 2. Default if omitted is 1. This selects whether a 1-knob or 2-knob slider will be presented. This function returns "CANCEL" if user presses Cancel button, otherwise it returns the time value as a string. For 2-knob version, it returns the two time strings separated by a space.

This function will not affect the tags of any croutons in the project, so if the intent is to do this, you must use the output of this function and CROUTONSETTAG to actually change a crouton's tags.

Note that the time values used in the editor are usually displayed in drop-frame format. The time values returned by this function are non-drop format -- this means the values returned are actual numbers of frames. The 2nd point, however, is EXCLUSIVE. This means that the 2nd time returned is not exactly the time shown in the 2nd time box, but is adjusted so it includes the user's desired last frame. This also makes the TAG_DURATION easy to compute: 2nd time - 1st time.

"REQ_NUMBER"
Put up number requester with optional min,max limits Returns "CANCEL" if user presses Cancel button.
ARGS: Title, [num,[min,[max]]]

"REQ_STRING"
String requester. Returns "CANCEL" if user presses Cancel button.
ARGS: Title, [String]

"REQ_TELL"
Put up requester with title and up to 3 lines, return 1 or 0
ARGS: Title, [line1,[line2,[line3]]]

"REQ_TIME"
Time code string requester -- alters/returns time string like 'HH:MM:SS:FF. Returns "CANCEL" if user presses Cancel button.
ARGS: Title, [time]

"REQ_OPEN"
Open an asynchronous requester with title and up to 3 lines, return nothing
ARGS: Title, [line1,[line2,[line3]]]

"REQ_CLOSE"
Close asynchronous opened with REQ_OPEN
ARGS: NONE

"STARTFILEREQ"
Open Grazer as file requester, use QUERYFILEREQ to query result
ARGS: Title, Initial path, Initial file

"QUERYFILEREQ"
Return result of Grazer as file requester: "" if requester is up, 0 if canceled, or name
ARGS: NONE

"CURRENTPATH"
Return current path in bottom grazer window, if any
ARGS: NONE

"GETSCREEN"
Return editor/switcher screen address
ARGS: NONE

"SET_VIEW"
Change views between project/files, project/Switcher, etc. Returns the code for the previous mode
ARGS: View# 0-5 though 3 is not supported, 5 is no top window, ARexx-only mode

View	Code
Files/Files	0
Project	1
Project/Files	2
Project/Project	3
Project/Switcher	4

"ADDPROGRAM"
Add a Name and command to 'Programs' popup
ARGS: Program Name(28 chars max), command string(127 chars max), flags
where:
Name appears in popup, runs command as either rexx or dos script depending on flags bit 0 (i.e. 1 for ARexx, 0 for dos). Returns number in popup sequence.
N.B. The editor waits for Dos commands to return, thus an app. that will not finish immediately should be preceeded by the 'run' command.
Note: DOS execution may be broken???

"REMPROGRAM"
Remove user-program from 'Programs' popup
ARGS: program index (position in popup) as returned by Addprogram

"PROGRAMCMD"
Return user-program command string
ARGS: program index (position in popup) as returned by Addprogram

"PROGRAMNAME"
Return user-program name as it appears in Programs popup
ARGS: program index (position in popup) as returned by Addprogram

"PROGRAMNUM"
Return program index (position in popup)
ARGS: User-program command string

"ADDTOOL"
Add a Name and command to 'Tools' popup
ARGS: Program Name(28 chars max), command string(127 chars max), flags
where:
Name appears in popup, runs command as either rexx or dos script depending on flags bit 0 (i.e. 1 for ARexx, 0 for dos). Returns number in popup sequence.
N.B. The editor waits for Dos commands to return, thus an app. that will not finish immediately should be preceeded by the 'run' command.
Note: DOS execution may be broken???

"REMTOOL"
Remove user-program from 'Tools' popup
ARGS: program index (position in popup) as returned by Addtool

"TOOLCMD"
Return user-program command string
ARGS: program index (position in popup) as returned by Addtool

"TOOLNAME"
Return user-program name as it appears in Tool popup
ARGS: program index (position in popup) as returned by Addprogram

"TOOLNUM"
Return Tool index (position in popup)
ARGS: User-program command string

"TBC"
Adjust Flyer TBC settings
ARGS: Setting,[value] (if value is omitted, current setting is returned
Setting may be 1 of: (cap.s are min. abbrev.)
Bright(-64 - 63), Contrast(0-127), Saturation(0-127),
HUe(-64-63), Fader(0-255), Phase(0-2047),
HorizAdj(0-909), Key(0-3), Mode(0-2)(for keyer),
Encoder(0-15), Decoder(0-8),
Termination(0-31), Input(0-3), Out(0,1)

Here are the tag names you can use with the crouton tag commands. Internally, they correspond to ordered numbers which may have the first bit set to indicate whether the values are LONGs or variable length tables. The meaning of the tags should be evident from their names, and if they're not, that is a good indication that you shouldn't mess with them, and they probably won't do anything if you do.

Version
Revision
AAeffect
NonAAeffect
KillInterfaceNonAA
KillInterfaceAA
ButtonELHlogic
CustomButtonELHlogic
NumberOfAnims
RequestFileName
AnimFiles
Frames
FieldSync
VariableSpeeds
ForcePlayForward
ForcePlayReverse
LoopAnims
AnimControl
AudioFastSamples
AudioMediumSamples
AudioSlowSamples
AudioFiles
AudioControl
PauseTimes
LatchColors
TransparentColors
PaletteColors
Equations
Encoder
VerticalScroll
ReverseTime
ReverseButtLog
ReverseCustomButtLog
KeyMode
MatteColor
CustomMatteColor
BorderColor
CustomBorderColor
LineNumbers
LineNumberPlane
ForceFreeze4

ForceFreeze8
ForceLive
ForceLumKeyOn
ForceLumKeyOff
ForceLumKeyOnBlack
ForceLumKeyOnWhite
BadDefaultFX
LoadPictures
FadeInDuration
FadeOutDuration
DigitalFX
TimeMode
LUT
DigitalPairs
LatchAM
LatchBM
LatchIS
HonorPreviewOverLay
ForceDefaultMatte
TurnAudioFilterOff
AudioStartField
NumAudioFields
ISandClipPause
Interlaced
FirstFieldNTSCII
BounceILBM
LatchRanges
LatchList
TransparentRanges
TransparentList
Color0Transparent
AbortIfSlow
NumSkipFieldsAtEnd
TBarDoesAuto
DoNotStompSprite
AbortLoopAtEnd
NonAAremap
100PercentWhiteMatte
UseEffectColor
LoopCount
CroutonType
TimeLine
IndexID
FCountMode
VariableFCount
SlowFCount
MedFCount
FastFCount
VariableFCount68000
SlowFCount68000
MedFCount68000
FastFCount68000
NumFields
StartTime
DescriptorList
AboutList
CommentList
AlgoFXtype
AlgoFXborder
NumFramesSlow
NumFramesMedium
NumFramesFast
NumFramesVariable
Page
Speed
Delay
Duration
AudioAttack

AudioDecay
RecFields
AudioOn
AudioStart
AudioDuration
ClipStartField
FadeInVideo
MaxDuration
VideoSource
LoadedSlices
OriginalLocation
AudioVolume1
AudioVolume2
AudioPan1
AudioPan2
PanelMode
ColorMode
CycleMode
DataMode
AdjustedVideoStart
AdjustedVideoDuration
Asynchronous
CommandLine
SMPTEtime
TBarPosition
HoldFields
TakeOffset
ASourceLen
BSourceLen
AudioFadeFlags

Here are the Error codes returned by the FLYERSTATUS command.

```
/*** General Flyer Errors ***/
#define FERR_OKAY           0x00  /* All went well */
#define FERR_CMDFAILED      0x01  /* Command failed for some reason */
#define FERR_BUSY           0x02  /* Still in progress */
#define FERR_ABORTED        0x03  /* User abort */
#define FERR_BADPARAM       0x04  /* Bad command parameter */
#define FERR_BADCOMMAND     0x05  /* Command not defined/supported */
#define FERR_BADVIDHDR      0x06  /* Ran out of video - no header detected */
#define FERR_WRONGMODE      0x07  /* Wrong play/rec mode for action */
#define FERR_OLDDATA        0x08  /* Incompatible data */
#define FERR_NOAUDIOCHAN    0x09  /* No free audio channel(s) */
#define FERR_CHANINUSE      0x0A  /* Video/SCSI channel not available */
#define FERR_BADFLDHAND     0x0B  /* Bad field handle */
#define FERR_CLIPLATE       0x0C  /* A/V clip started late */

/*** Flyer Internal Errors ***/
#define FERR_NOTASKS        0x10  /* No SCSI tasks available for use */
#define FERR_LISTCORRUPT    0x11  /* Internal list corrupt */
#define FERR_NOTINRANGE     0x12  /* Internal list error */
#define FERR_EEFAILURE      0x13  /* EEPROM failure */
#define FERR_NOFINDERS      0x14  /* No FrameFinders available for use */
#define FERR_BADMODULE      0x1F  /* Incompatible module provided */

/*** FileSystem Errors ***/
#define     FIRSTFSERR      0x20
#define FERR_OBJNOTFOUND    0x20  /* Could not find file/dir */
#define FERR_FULL           0x21  /* Drive full */
#define FERR_DIRFULL        0x22  /* Directory full */
#define FERR_EXHAUSTED      0x23  /* Directory list exhausted */
#define FERR_FSFAIL         0x24  /* FileSystem failure */
#define FERR_WRONGTYPE      0x25  /* Wrong type of object */
#define FERR_UNFORMATTED    0x26  /* Drive not high-level formatted */
```

```
#define FERR_EXCLUDED           0x27  /* Exclusive lock prevented action */
#define FERR_OUTOFRANGE         0x28  /* Seek beyond bounds */
#define FERR_CANTEXTEND         0x29  /* End of file, and cannot extend file */
#define FERR_PROTECTED          0x2A  /* Drive write-protected */
#define FERR_DIFFERENT          0x2B  /* Grips are different objects */
#define FERR_EXISTS             0x2C  /* File already exists */
#define FERR_NOMEM              0x2D  /* Out of storage */
#define FERR_DELPROT            0x2E  /* Delete-protected file */
#define FERR_READPROT           0x2F  /* Read-protected file */
#define FERR_WRITEPROT          0x30  /* Write-protected file */
#define FERR_INUSE              0x31  /* Disk/object in use */
#define FERR_DIRNOTEMPTY        0x32  /* Directory was not empty */
#define     LASTFSERR           0x32

/*** SCSI Errors ***/
#define FERR_SELTIMEOUT         0x40  /* SCSI Time-out -- no drive */
#define FERR_BADSTATUS          0x41  /* Bad status after executing command */

/*** Sequencing Errors ***/
#define FERR_WRONGDATATYPE      0x60  /* Asked for improper type of data from clip */
#define FERR_DRIVEINCAPABLE     0x61  /* Using video clip from a non-video drive */
#define FERR_NO_BROLLDRIVE      0x62  /* No video B-roll drive found */
#define FERR_HEADFAILED         0x63  /* A/B head missing/problem */

/*** Amiga Library Errors ***/
#define FERR_NOCARD             0x70  /* Flyer card specified does not exist */
#define FERR_LIBFAIL            0x71  /* Library failed to pass command to Flyer */
#define FERR_ASYNCFAIL          0x72  /* An asynchronous command failed */
#define FERR_VOLNOTFOUND        0x73  /* Volume name not found */
#define FERR_NOFREECMD          0x74  /* Library<->Flyer RAM clogged */
#define FERR_BADID              0x75  /* Illegal async ID */

#define FERR_LIMIT          0x7F
```

Selecting and Configuring SCSI-2 Hard Drives for Flyer Systems

by Karl Schmidt

You need to examine many aspects of the drive. One of the most obvious traits is the drive's speed, but you must also look at subtler factors that affect overall performance. These factors are related to the drive's caches and how it does thermal calibration.

Requirement Summary

	Standard Mode	High Quality 5 (HQ5) Mode
Minimum transfer speed	3.7 Megabytes Per Second	4.8 Megabytes Per Second
Maximum idle time	??? Milliseconds	200 Milliseconds

All table items should be measured with NewTek's DRIVESPEED utility.

Minimum Data Transfer Speed

The minimum speed of the drive is the first thing to check. Manufacturers' specifications can be confusing as they may be stated as "transfer rate," which refers to the speed of the SCSI interface, not the speed that data is available during Flyer playback and record. Only the Newtek program DRIVESPEED (available free from your dealer) can properly measure speeds of potential Flyer video drives.

Keep in mind that the drive must be able to stay at this rate (,see table above) over its entire surface. Most drives cannot transfer data as fast to and from the inner tracks as they do the outer tracks. If the speed drops below 3.7 or 4.8 MB/sec at any point on the drive, it will not work properly with the Flyer. Testing a drive's speed only on the outer tracks, where it is usually fastest, may falsely lead you to believe it will work with the Flyer. "Paced" testing, where the drive's buffers are kept full, does not test for the worst case. NewTek's DRIVESPEED program measures the drive's performance while buffers empty. A drive should be able to sustain, over the entire surface, read and write speeds of no less than 3.7 MB/S (megabytes per second) for standard mode and 4.8 MB/S for HQ5 mode as measured with NewTek's DRIVESPEED utility.

While 3.7 MB/S an 4.8 MB/S are good target values, a faster drive will have certain advantages. If a drive can only read data as fast as the Flyer is asking for it, then the drive has no chance to fill its and the Flyer's buffers. If a momentary gap in the data stream arises, the buffer memory is drawn upon to make up for the shortage. After the gap has been filled by the buffer, a fast drive will refill the buffers faster than one that "just makes the grade."

Setting Drive Cache Parameters

Since drive speed is so important, the first thing to do with a new drive is to try to squeeze all of the speed out of it that you can by setting the drive's cache parameters. Some drives have their cache parameters set properly right out of the box, but most will need to be tweaked slightly to get the best performance with the Flyer. There are a few cache parameters that should be set on most drives. A drive must (if possible) have both Write Cache and Read Cache enabled. The Read and Write Retention Priorities should both be set to 1. This tells the drive that it should not keep old data in its cache, but replace it with new data as it becomes available. This parameter is related to others that affect the pre-fetch of a drive; how much data is read ahead and placed into the cache before it is actually requested.

There are eight cache parameters of interest supported by various drives:

Write Cache enable	should be set to 1, if supported.
Read Cache disable	should be set to 0, if supported.
Disable Pre-Fetch Transfer Length	should be set to 0xFFFF or as high as possible.
Minimum Pre-Fetch	should be set as small as possible, usually 0x0000.
Maximum Pre-Fetch	should be set as high as it will go, hopefully 0xFFFF
The Maximum Pre-Fetch Ceiling	should also be set as high as it will go and is usually the same value as the Maximum Pre-Fetch.
Number of Cache Segments	The Number of Cache Segments is not so straightforward. You will usually want to set it to 2,3, or 4. The best value for this parameter varies from drive to drive, so the best way to determine which value to use is through trial and error. Test the drive's speed with each setting and find the optimum setting for your drive.
Read Demand Retention Priority	should be set to 1, if supported.
Write Demand Retention Priority	should be set to 1, if supported.

Now that you know what cache parameters to set, you need to know how to set them. All drive cache parameters are selected through SCSI-2 Mode Sense/Mode Select Pages. In particular, the drive's cache settings are held in Page 0x08. A drive's cache settings can be modified with any software that can read and set values in these Pages. Most programs that change these values will also tell you which you may set. Drives vary in which parameters are supported and which can be changed. If you set a certain parameter and the drive does not seem to remember it, then there are a few steps that you can take to find out why. The first step is to see if that parameter is changeable on your particular drive. Mode page 0x48 contains a table of all of the values which are changeable. Most mode page software will allow you to view this page. If the value is listed as changeable, but does not seem to allow you to change it, you may need to set a different parameter first. You will need to find out if there is such a 'priority' or 'hierarchy' of parameters from the manufacturer of the drive. This information is usually contained in a technical document published by the manufacturer. Carefully study the documentation that came with your Mode Page software and/or from your drive's manufacturer.

Note that some drives have fixed values for some or all of these parameters and there may be other parameters that a given drive supports which would help its performance with the Flyer.

For further information refer to the manufacturer's documentation or ANSI standard X3T9.2/86-109 (the definition of SCSI-II).

Most drive manufacturers have mode page software available on their BBS. Here is a short list of some of the disk drive manufacturer's BBS phone numbers:

Manufacturer (state)	**BBS phone #**	**modem speed**
Conner International (CA)	408-456-4415	14,400
IBM (NC)	507 286 5314	14,400
Iomega (UT)	801-392-9819	14,400
Maxtor/Miniscribe (CO)	303-678-2020	9,600
Micropolis Corp (CA)	818-709-3310	2,400
Quantum (CA)	408-894-3214	2,400
Seagate (CA)	408-438-8771	28,800

Also you may consider CoComps SCSI tool set (800 658 5981) or PTI's SCSI Service Tool or SCSI tool box (800 829 7274). This software runs about $300 to $1,000.

Maximum Idle Times - Thermal Calibration

Gaps in deliverable data are caused by seeks, data errors, and thermal calibration. On a properly working drive, the only gap or idle time that is of concern to Flyer user is the T-CAL (Thermal CALibration). While you may have heard of T-CALs, and know that they cause problems when you are using the drive for video, do you know what they are and why they happen?

As a drive changes temperature, its data heads can become improperly positioned in relation to the data cylinders. Thermal calibration keeps the drive heads properly aligned with the data cylinders. During a T-CAL, a drive positions a data head over a special calibration cylinder and offsets the servo head until the data head is precisely centered on the calibration cylinder. This offset is then used for all seeks until the next T-CAL. The problem with this process is that while performing the T-CAL no data goes to the Flyer and the drive and Flyer are draining their buffers. If the T-CAL lasts long enough, the Flyer will run out of data and a stutter occurs as the flyer displays the last 4 fields of data again and again until new data is available.

Determining how long of a T-CAL is acceptable is difficult. Whether or not a T-CAL will cause the Flyer to stutter depends on the state of the Flyer's buffer at the time of the T-CAL. That in turn depends on all of the factors previously discussed. You can be sure that the longer the T-CAL lasts, the more likely that your video will stutter.

T-CAL is a function of the drive over which the user has little or no control, so a drive must be tested with NewTek's DRIVESPEED utility to see if its T-CAL behavior is acceptable for use with the Flyer.

Embedded servo drives do not require T-CAL interruptions at all! These drives work very well with the Flyer because their idle times are usually about 100 ms. They perform T-CALs without repositioning drive heads (interrupting the flow of data). One last comment on testing drives: some drive models show a surprising individual variation! This can be due to surface defects and gain errors. Gain errors are defects that occur after the drive is low level formatted and are "mapped out" on the fly. Sometimes a low level format can help an individual drive.

	blocks	size	feilds	time	megs/sec
Standard mode		5034484224	92168	1536.133333	3.277374506
		32967680	596	9.933333333	3.31889396
	64493	33020416	608	10.13333333	3.258593684
	34000	17408000	324	5.4	3.223703704
	263470	134896640	2428	40.46666667	3.333524876
	45644	23369728	392	6.533333333	3.576999184
	204665	104788480	1900	31.66666667	3.309109895
	421581	215849472	3796	63.26666667	3.411740864
	58929	30171648	568	9.466666667	3.187145915
	200898	102859776	1784	29.73333333	3.459409507
	6855	3509760	60	1	3.50976
				average	3.351477827
extended black	7733094	3959344128	253492	4224.866667	0.937152445
Extended	7733087	3959340544	113052	1884.2	2.101337726
	74353	38068736	1376	22.93333333	1.659973953
	102670	52567040	1832	30.53333333	1.721627948
	103348	52914176	1812	30.2	1.752125033
	133957	68585984	2288	38.13333333	1.798583497
	164147	84043264	2820	47	1.788154553
				average	1.803633785
HQ5	3451502	1767169024	30564	509.4	3.469118618
	134612	68921344	1056	17.6	3.915985455
	119476	61171712	836	13.93333333	4.390314258
	55866	28603392	420	7	4.086198857
	68196	34916352	608	10.13333333	3.445692632
	63301	32410112	964	16.06666667	2.017226888
				average	3.554089451
thoretical file sizes					
	extended	standard	HiQ5		
	>.750MB/s				
		3.75	4.79	Max	
		3.44	4.48	upper	
				deadband	
		2.75	3.58	Lower	

Using the Horita TRG-50 with the Flyer

By
Jeff Scheetz & LLoyd Slapar
NewTek Tek Support (LastUpdated 10/11/95)

Expectations

1.- The New Tek Video Flyer in revision 4.07 (and several before) does support the use of Drop Frame Time Code.
2.-The TRG-50 Supports LTC or Longitudinal Time Code. LTC is usually read from a third audio track on Beta VTRs.
3.- If your specific application demands use of VITC (Vertical Interval Time Code) you will to add a Horita VTL-50 to your system. The VTL-50 reads the VITC and puts out an LTC signal that the TRG-50 can understand.

Additional Required Hardware

- Null modem adapter from Radio Shack. Female DB9 to Male DB9 (Cat. No. 26-264A)
- 75 ohm BNC terminator also available from Radio Shack.

Connections

To record LTC to the Flyer:
1. Connect the *TC OUT* (or *AUDIO 3* out) of your VTR to the *TC IN* on the TRG-50.
2. Connect *REFERENCE VIDEO OUT* (or Sync out) to the *VIDEO IN* of the TRG-50.
3. Connect the *VIDEO OUT* of your VTR into the *INPUT 1* of the Flyer.
4. Place a 75 ohm terminator on the TRG-50's *VIDEO OUT* to terminate the reference signal.
5. Connect the TRG-50's *COMM* cable from the TRG-50 to the *NULL MODEM ADAPTOR*.
6. Connect the *NULL MODEM ADAPTOR* to the Flyer's *SERIAL A* cable, and double check the other end of the *SERIAL A* cable to be certain that it's connected to *SCSI A* on the Flyer.
7. Be sure the TGR-50 is receiving power.
8. The TRG-50 TC out can be left empty.

TRG-50 Settings

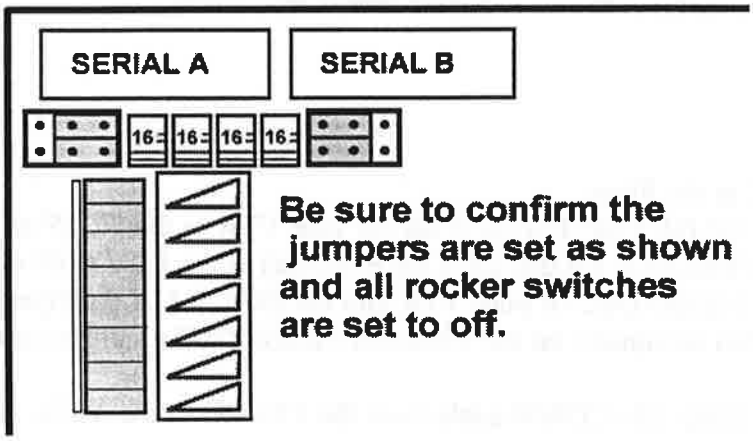
- Data should be set to TC.
- Mode should be set to RDR.
- V-SIZE and V-POS refer to window placement and may be ignored.

Flyer Operation with TRG-50

Same procedure as for normal Flyer operation.

FYI

- The Flyer will not display the Time Code while recording.
- Once in the MAKE CLIPS window, the default IN and OUT points should reflect the Time Code numbers.
- The Flyer will not playback time code or output an Edit Decision List. EDL output may be handled by out third party developers in the future.
- Be sure to set the Flyer's serial jumpers as shown in the diagram below, and that all rocker switches are set to the OFF position.



Null Modem Notes

For those who wish to make a null modem, the pin connections are as follows.

Female DB9	1	2	3	4	5	6	7	8	9	
Male DB9	4	3	2	6 and 1	1	5	4	8	7	Not Connected

Additional Information

Call NewTek Tek Support. (913) 228-8282