DEMONSTRATION SOFTWARE FOR AN EXPERIMENTAL VIDEO WORKSTATION

by

WILBERT L. BLAKE, JR.

Submitted to the Department of
Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements
for the Degree of
Bachelor of Science in Electrical Engineering
at the
Massachusetts Institute of Technology
May 1986

Signature of Author ______________________
Department of Electrical Engineering and Computer Science
May 9, 1986

Certified by [illegible]
Glorianna Davenport
Thesis Supervisor

Accepted by [illegible]
David Alder
Chairman, Department Committee

TABLE OF CONTENTS

...........ACKNOWLEDGEMENTS....................................PAGE i

...........ABSTRACT...PAGE ii

PART I...GLOSSARY...PAGE 1

PART II...INTRODUCTION TO EVW................................PAGE 3

PART III...SYSTEM BACKGROUND................................PAGE 4

PART IV...SPECIFICATION OF SOFTWARE FUNCTION...............PAGE 5

PART V...CODE DESCRIPTION ....PAGE 6

PART VI...OBSERVATIONS: IMPROVEMENT, EXPANSION, UTILITY....PAGE 10

PART VII...ONE INTERFACE APPLICATION........................PAGE 14

...........APPENDIX I:..DIAGRAMS AND CODE LISTINGS.........PAGE 16

...........APPENDIX II:..CONTROL.C..........................PAGE 17

ACKNOWLEDGEMENT:

Several individuals contributed greatly to my programming and editing efforts, and I must mention them to partially express my appreciation. These individuals are: my thesis supervisor, Glorianna Davenport; EVW system hardware designer and NAB project supervisor Keishi Kandori; EVW software coordinator Russ Sasnett; EVW control software designer Reza Jalili; Unix hackers/consultants Charles Coleman and Cecil MacCannon. Glorianna and Keishi provided the opportunity to contribute to a very interesting project, and the experience has proved rewarding. Everyone mentioned patiently answered the most trivial questions about Unix and C, EVW, and video in general. Finally, I acknowledge Shelly Johnson, who prepared this document far better than I conceived it.

I dedicate this to my family, Ila, Eleece, and Paul.

Wil Blake
May 16, 1986

ABSTRACT:

M.I.T. Film/Video desired a computer controlled, Society of Motion Picture and Television Engineers (S.M.P.T.E.) standard compatible video editing and viewing environment for its movie laboratory. Such an environment would serve as a video workstation, providing means in the lab for computer driven manipulation of video information. Computers in the experimental workstation should readily connect to the computational resources in the M.I.T. Media Laboratory, as well as offer challenges to programmers in the M.I.T. community. An "open" system, then, supporting a variety of video and computational devices, would enable system expansion while best utilizing financial resources.

Various user interfaces to video information could receive attention in an experimental system. A hardware grant from the Asaca/Shibosoku Corporation promised an interesting user interface: 32 simultaneously displayed frames of video organized into four columns each holding eight frames; one to four columns gets input from one (of four total) video sources. This "multiview" mode contains a graphic audio signal display alongside the columns of frames. This multiview/audio graphic capability invites comparison to film image manipulation, a comparison that the experimental workstation will explore further.

Having outlined research goals and acquired initial hardware for an "experimental video workstation," focus became demonstration of its capabilities at the National Association of Broadcasters (N.A.B.) Convention in Dallas this past April. At the NAB convention exhibit, the experimental workstation would receive evaluation from the commercial broadcast community in the presence of the latest post production video systems.

GLOSSARY

Here are brief explanations for terms that occur frequently in this document.

Term	Meaning
NAB	National Association of Broadcasters, a major governing body whose members include both radio and television broadcast companies and broadcast equipment manufacturers. The annual NAB convention holds technical discussions concerning broadcast engineering issues in addition to a huge exhibit for manufacturers' latest hardware. An "Experimental Video Workstation" was part of the Asaca/Shibosoku Corporation exhibit at the 1986 NAB convention, held April 13-16 in Dallas.
CMX KEYBOARD	Video editing system configuration where edit commands originate from a computer keyboard that follow key assignments used by the CXM corporation in its computer editing systems. Control software for an "Experimental Video Workstation" emulates the CMX keyboard assignments of editing functions to keys.
SMPTE	Society of Motion Picture and Television Engineers. Organization which establishes technical standards for film and video motion pictures.
timecode	SMPTE standardized format for referencing videotape. Each frame on the tape holds a unique reference designation as hours: minutes: seconds: frame. For example, "00:00:00:01" names the first frame on a videotape. Timecode's adoption in videotape production because it

Term	Meaning
	grants highly accurate, repeatable access to individual frames regardless of the editing system components.
EDL	Edit Decision List. A table of timecode values representing transitions between video input sources. An edl serves as a "script" for a multisource editing session.

EVW System Notation:

Shorthand	Component Name	Model
Video devices		
LDP1, LDP2	laserdisk players 1, 2	Sony LDP1000
MV	video multiviewer	Asaca AEV300
SW	video switcher	Asaca ASW300
RVTR	record vtr	Sony BVU820

Control Devices (Computers)

DECPRO	Digital Equipment "DECPRO 350" computer
BOBCAT	Hewlett-Packard HP98710 "BOBCAT" computer

INTRODUCTION TO EVW:

An "Experimental Video Workstation," EVW, is a project in the Film/Video section of the MIT Media Laboratory; Glorianna Davenport, Lecturer and Project Coordinator in the Film/Video section, coordinates the project ongoing effort. EVW is a computer-based post production system for editing and accessing video information. Its design criteria specified an open, modular system utilizing traditional post production hardware components, meeting broadcasts data standards, and allowing interconnection to MIT's Media Laboratory computer system. In addition, EVW's design goals include flexible hardware configurability, which can generate new approaches to video segment access and movie creation. Asaca/Shibosoku Corporation included EVW as part of its exhibit at the National Association of Broadcasters convention April 13-16th in Dallas, Texas. Keishi Kandori, engineer with the Ashai Broadcasting Company and research associate at MIT Film/Video, designed the EVW hardware configuration and directed the NAB exhibit demonstration efforts. Five other individuals contributed software to the NAB demonstration of EVW. Demonstration software includes: edit decision list (EDL) database formulation and manipulation, developed by Andria Wong, MIT '86; EDL edit execution, developed by John Barbour, MIT '86; human interface (soft key and icon driven menus) and video segment database software, developed by Russ Sasnett, MIT M.S.V.S. '86 and current Research Affiliate at MIT Film/Video. Software described in this thesis depicts the features provided in the multiview edit system function of EVW. Reza Jalili, MIT '89, wrote the hardware control routines, which emulate the CMX keyboard command function. Both EVW system control computers operate licensed UNIX environment, Venix II on the decpro and HP-UX on the bobcat, thereby facilitating the network connection to the MIT Media Laboratory computer system. All EVW software is in the "C" programming language because of its straightforward interaction with UNIX operating systems.

SYSTEM BACKGROUND:

Diagram 1 depicts the editing system control block diagram. Control routines for the video components reside on the decpro. Video devices are: Laserdisk players 1 and 2 (LDP1 and LDP2), a record videotape deck (rvtr), a multiviewer, and a video switcher. A dual screen monitor serves as output picture display. CMX keyboard commands either local or from the bobcat cause the decpro to send the corresponding instructions to the appropriate video device. Diagram 2 shows the EVW system video signal block diagram. LDP1 and LDP2 are the video input sources, rvtr is the final recorded edit destination. All three have inputs to both the switcher and the multiviewer. CMX keystrokes govern selection among the video sources (diagram 3). Switcher output is the final video signal for recording by the edit destination, rvtr. Asaca's ASW300 video switcher performs effect switching from one of input device to another. It performs cuts, dissolves, and a 64 different wipes between input sources. Dissolves (fades) have seven possible durations. Asaca's AEV300 multiviewer has a "multiview" mode that digitizes 32 video frames in real time and displays them in a four column by eight frames per column format (see diagram 4). Active multiviewer columns continually display the input video signal frame by frame, as frames scroll up or down the active columns according to the current direction (forward or reverse) of the video input source. Frozen columns display their last sample. Columns are individually activated or frozen, so that each column may represent an eight frame sequence from one of the four possible input sources; all or any combination of columns may show a sequence from one source (see diagram 5). After freezing a column, individual frames in the column become accessible for movement, insertion elsewhere, deletion or temporary removal. This capability for access to "frozen" frames makes possible an edit "preview" done in the multiview mode before actually performing the edit.

SPECIFICATION OF SOFTWARE FUNCTIONS:

System designer Keishi Kandori outlined the specifications for the multiview edit demonstration software as one component in the complete NAB demonstration of the EVW. A newly introduced computer, the Hewlett Packard 98710 "Bobcat", would activate the device control software via a serial link to the decpro. Two software routines, one running on the bobcat, one on the decpro, would combine to perform the demonstration. Software for the multiview edit demonstration would reside primarily on the bobcat, activating decpro based device control routines. A routine on the bobcat would transmit CMX format command sequences to the decpro over the serial link. Acting as a preliminary command interpreter, the decpro routine will pass the commands as arguments to the device control software, or perform special functions for initializing the system's video components.

CODE DESCRIPTIONS:

Demonstration of the multiviewer edit system proceeds with the execution of a CMX command character sequence. "Submain.c", the decpro's routine, intercepts command characters transmitted from the bobcat and then utilizes the appropriate device control routine for execution of the command. Calls to "submain.c" follow as: submain (filename) char * filename. Refer to listing 1 for the subsequent description of "submain.c". In the code listing, explanations follow declarations of key variables. All the video devices undergoe initialization with the call to "setup", a routine included among the control software done by Reza Jalili. After initialization, the argument filename passed to "submain.c" becomes the source file for CMX commands. For testing a particular command sequence, the argument filename accesses a local (to the decpro) file containing the sequence that will be tested. Otherwise, the argument filename is "/dev/coml", the 9600 baud serial link to the bobcat, making the link the command source. Having determined the input command source, "submain.c" gets characters one at a time from the source and stores them in the global array "cmdlist". "Getachar", another control routine, gets characters from "cmdlist", then the "switch" statement performs special commands where the case requires. Subsequently, the appropriate characters go to "control", which executes the device specific instructions. Here the "while" loop statement begins its next iteration, getting another character until either the command list exhausts itself or a "q" command character indicates the end of a command sequence.

Character reception done by the decpro routine "submain.c" requires a source of CMX command characters. "Submain.c" expects that its formal parameter serves as the command source. This parameter names a file, which can contains a list of commands for "submain.c" to execute sequentially. Because Unix/C handles input and output, i/o with files, the argument passed to submain can also specify an i/o port, such as the serial line "/dev/coml". Access to both local files and i/o ports as command sources provides flexibility in the formulation, manipulation, and recall of editing command sequences. An editor using EVW could perform an edit sequence manually, before committing the sequence to a local file, and finally moving the command sequence to a remote computer; "Submain.c" and Unix handle the interpretation of a command uniformly regardless of the nature of its source file.

Functionally, the demonstration software running on the bobcat should perform sequential CMX command character transmission to the decpro via the serial input/output link uniting both computers. "Demo.c", the bobcat's demonstration routine, requires an argument file that contains the sequence of CMX commands that will perform the demonstration. File manipulation ease in Unix and C lends itself to this approach. "Demo.c" can transmit a CMX command sequence from any recognizable demonstration file passed as an argument. Listing 3 shows an example demonstration file. "Demo.c" will use values from the demonstration file to initialize an array of structures, each structure representing information for transmission of one CMX command character. Every line in the demonstration file corresponds to the values for one structure, structures being named "events". Entries within a demonstration file line individually contain the values for individual members (elements) of an event structure. "Demo.c" transfers information from the demonstration file into its array of of event structures on a line by line basis. One line in the demonstrtion file has four entries, three of which receive assignment to an event structure. The first entry in the file acts solely as visual aid for making demonstration files. It is a number that identifies the line; line numbers need not have consecutive order, but should increase with progress through the file.

After the first entry, fields within a line of a demonstration file contain information used in CMX command transmission. An actual command character comprises the second entry in the line. "Demo.c" will assign that character to the "command" element of an event structure, and at some point transmit the character to the decpro. An optional character string makes up the third entry in demonstration file line; "demo.c" assigns the string to the "options" element of the current event structure. Some CMX commands require additional information for their execution. For example, cueing to a laserdisk frame requires specification of the frame. Necessary command information, such as a frame number, goes into the optional entry within a line. "Demo.c" transmits the optional information to the decpro after sending the CMX command character. A string of format "XX:YY" comprises the final entry in a line of a demonstration file. The format represents a time of day expressed in minutes and seconds, and can easily expand to hours. Assigned by "demo.c" to the "time" member of an event structure, this string indicates when to send the current command character and provides explicit timing for character transmission. That is, one version of "demo.c" compares the real time in minutes and seconds with the time member of the current event structure; at the indicated time, "demo.c" transmits the associated CMX character and then waits for the time to transmit the next command.

Refer to Listing 4a for the following description of "demo.c". The structure "event" represents run-time information for transmission of one CMX command. "Event-table", an array of event structures, represents run time information for transmission of a CMX command sequence. Execution begins with the call to "init.c", which loads the static demonstration file values into the event-table. "Init.c" reads the demonstration file line by line into a buffer, where "sscanf" formats the line entry values and assigns them into an event structure. Some diagnostic printouts monitor the event structure values, before the array offset, "table", and the current number of events, "total", receive increments. Once the demonstration file ends, execution returns to "demo.c". "Demo.c" then progresses through the array of event structures, sequentially transmitting the command element to the serial line "/dev/tty00". Input/Output in Unix and C accesses i/o ports as files, allowing the library formatted file print routine, "fprintf," to perform transmission of the CMX command characters over the serial line. A call to "ioctl" initialized "/dev/tty00", (the serial line) to the hardware specifications indicated in the structure "sgtty-fdbuff". For this EVW configuration, those specifications set the baud rate to 9600. After sending the CMX command, "demo.c" uses a "switch" statement to determine whether the command requires optional information, and if so, that optional string also goes to the serial line. In the version of Listing 5, "demo.c" pauses for a time between command transmissions. Listing 5's version of "demo.c" uses the system clock (and some format conversion with "sscanf") to transmit the command at the time expressed in the present event's "time" member.

RESULTS, OBSERVATIONS:

Multiview edit demonstration software performed consistently at the NAB exhibit, there meeting its most emphatic design requirement. One part of the EVW research project, the NAB demonstration software "simulated" possible editing application rather than actually meet application requirements. Observations in this section arrive from the projection of the simulated editing function to an operational editing application. Time constraints on the multiviewer edit software operation precluded substantial efforts for code improvement, so this section offers improvements focused on an editing application. Development of the complete NAB demonstration software package placed similar constraints on all software contributors; one such constraint discussed in this section may be avoided in future EVW project efforts. Finally, comments from NAB observers of EVW deserve attention because they suggest goals for the eventual implementation and use of multiview edit systems.

Both routines in the multiview editor demonstration software use datafiles containing a CMX command character sequence. By design, any Unix editor may create or modify these data files, but the files have to follow a rigid format in order for the correct execution of editing commands. Optimizing these file formats for portability and usage ease becomes the key to enhancing the multiview edit demonstration software. First, examine a decpro command file, accessed by "submain.c", shown in Listing 2. A decpro command file literally consists of a sequence of CMX command characters that "submain.c" will; 1.) store in a device control buffer and 2.) execute in order. Spaces, which correspond to the CMX "allstop" command, cannot separate the characters in a decpro command file without execution of "allstop". File readability dictates that spaces separate command characters in a sequence. A command file "interpretation" routine could allow intracharacter spacing without causing an "allstop" instruction. For this interpretation, "allstop" occurs only with a predetermined number of spaces, otherwise spaces just parse command characters. This interpretation routine, incorporated into "submain.c", would provide a better format for command character files on the decpro.

"Demo.c", the bobcat situated multiview edit demonstration routine, also requires a formatted file to supply CMX command characters that will travel to the decpro. This format appears in Listing 3, and it gives a reasonable indication of an ordered command sequence. Tabular listing provided in this format improves the readability of a CMX command sequence; extension of the same or a similar format to decpro command files promises standardization and portability for command sequence format. Unix' file manipulation and C's format conversion routines suggest adoption of standard representation for command character files. Then the (computer dependent) local CMX command file interpretation routines could convert the file information for either command execution (on the decpro) or command transmission (on the bobcat). In this method, the same editing command sequence can execute locally, or travel to a remote edit controller.

Multiview editor demonstration software and (all device accessing demonstration software) utilized the existing control software (written by Reza Jalili) in order to manipulate EVW's video components. Code revisions obviously took place during the harried softwared development stages; revisions in the control software generally required changes in the muliview edit demonstration software. Fox example, control software header file changes meant modifications and recompilation of "submain.c", the decpro demonstration routine. Occasionally, control software revisions became evident only after failure occurred in previously operational demonstration software. A "make" revision managing utility would have greatly alleviated confusion over necessary improvements to the control software.

At NAB, demonstrations viewers responded positively to the multiview editor. Most comments praised the multiview mode's resemblance to film, with each eight frame column analogous to an eight frame strip of film. Functionally, however, the film similarity has limitations concerning editing, yet it offers advantages useful for video information access. Unlike individual frame availability in film, the multiview mode cannot locate actual video input frames. Frames in the multiview display are not actual video input frames, but are processed rgb signal representations of input video frames. Multiview mode frame manipulations, such as scene insertions and deletions, affect the frozen display rather than the source video input signal. In fact, freezing a multiview mode column removes its display from the video input source. Editing operations affect the video input sources, best shown in the normal, full screen mode. Frozen frames in the multiview mode cannot receive normal size display, because normal mode display only shows the currently selected input source.

Rather than a device for edit execution, the multiview editor presents itself as a visual tool for recalling desired video information from multiple sources. Single frames from the multiview mode may act as symbols for the video segments from which the frames originated. Up to 24 frozen (of 32 total multiview frames) can represent one segment from any of the input sources. One of the four columns remains active for displaying the current input segment. For example, "edlview.c", demonstrates this capability. Running on the decpro, this routine uses edl file information to cue video sources (ldp1, ldp2, and rvtr) to both edl line inpoints, with the subsequent eight frames shown in active display column 4 (see diagram 6). Then shots from both inpoints travel to a storage "bin" in frozen display column 1. Subsequent "bin" frames will scroll previous frames into frozen columns 1 and 2, until all 16 bin locations contain edl inpoint frames. Applications like "edlview.c" uses the multiview editor as an interface to video information because visible multiview images convey more about a video segment than an edl entry or a timecode value. Of course, the multiview frames reduced sized limits the detailed information gathered from the image.

Presence of the audio graphic alongside the multiview "strips" (columns) constitutes another similarity to film, where the audio resides next to the picture image on the filmstrip. Sound edits can benefit from the audio graphic. Changes in the graphic display can mark inpoints and outpoints for edit decision lists or actual edits. Time constraints prohibited any exploration of audio edits assisted by the multiview editor, but it remains a goal for future EVW projects. Another drawback for audio edits concerned the laserdisk players, which mute their audio output at low playback speed, thereby removing the audio graphic from the multiview display.

Liabilites in the multiview editor center on inaccurate multiview frame representation at low laserdisk player and record vtr playback speeds. At these "jogging" speeds, the multiviewer's display update functions do not accurately regenerate display of its input video signal. The multiviewer updates its display at a fixed rate for all input device playback speeds; lower playback speeds cause the multiviewer to update the same input frame so often that it "sees" the single frame as if it were a sequence of identical frames (comprising a still shot). As a result, the active multiview columns display falsely identical frames even though the input video signal contained slowly advancing individual frames. Multiviewer weakness at low input playback speeds will allow correction, because the multiviewer instruction set includes adjustment of its output display refresh rate. EVW device control software will eventually include the MV refresh rate adjustment to correct erroneous multiview display at low playback speeds.

PART VII: ONE INTERFACE APPLICATION:

One application developed on the decpro demonstrates the multiview editor user interface. "Edlview.c" accepts an edl file as its argument and, using information from one edl line, proceeds to: 1.) cue one source to the source inpoint timecode location, inserting the first frame into a frozen "bin" display column, and 2) cue the record source to the record input location, inserting its first frame into the bin. Frame insertion happens such that previously inserted display frames scroll from column 1 to column 2 during subsequent insert operations. This allows for up to 16 inpoints/frames appearing in the storage bin at once. Consult Listing 6 for the preceeding description of "edlview.c". After successfully opening the argument edl file, "setup" initialized all the video devices. "Submain.c", previously described in part 5, executes the "freeze 2" file of CMX command characters, freezing the two leftmost multiview mode display columns for use as the shot storage bin. A "for" loop limits the total display loop to eight iterations, correspondingly limiting the edl file length to eight lines or events. Inside the "for" loop, a "while" loop repeats execution until the edl file either ends or provides an unrecognizeable line. "Sscanf" formats the current edl file line, assigning values into the device and inpoint variables while suppressing unused edl line information. Named for the associated edl line "device" field, the device variable tells which input device acts as the current source. Inpoint variables, source and record, contain timecode character strings from the edl line "source inpoint" and "record inpoint" fields. Another "sscanf" call converts the time codes into seven digit integers which "convert" translates into frame numbers suitable for laserdisk player search (cue) instructions. "Control ("s")" or "control ("d")" selects the specified device, ldp1 or ldp2, for reception of the CMX cue commands. "Control ("n")" and "control ("n")" cue the selected laser disk player and record vtr to the source inpoint frame and record input frame locations;

"submain ("insert")" inserts the first frame from the inpoint segments into the frame storage bin, scrolling the previously stored bin frames. The edl line input/ variable assignment, inpoint cueing, and shot storage execution loop continues until the edl line length maximum occurs, or until the edl file exhausts itself. Once the execution loop terminates, "control ("Esc-q")" reinvokes multiview mode for display of the storage bin.

APPENDIX I: DIAGRAMS AND CODE LISTINGS

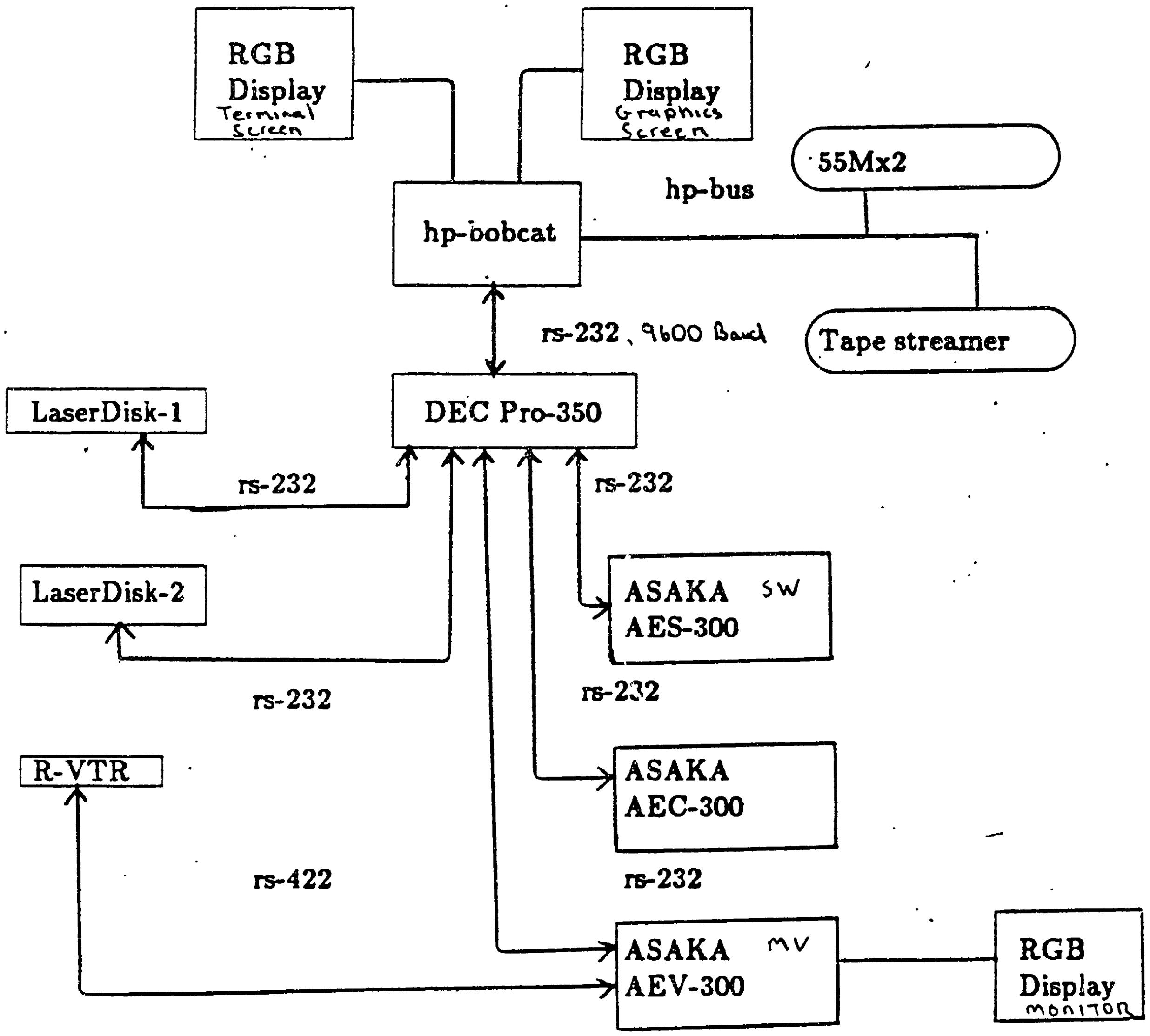
DIAGRAMS

1. EVW SYSTEM CONTROL
2. EVW VIDEO SIGNALS
3. VIDEO INPUT SELECTION
4. MULTIVIEW MODE
5. "FROZEN" vs. "ACTIVE" MULTIVIEW MODE COLUMNS
6. EDLVIEW.C' s MULTIVIEW FRAME MOVEMENTS

LISTINGS

1. SUBMAIN.C and UTIL.C
2. A DECPRO COMMAND FILE
3. A BOBCAT COMMAND FILE
4. DEMO.C
5. DEMO.C , time comparison version
6. EDLVIEW.C

control


RGB Display
Terminal Screen
RGB Display
Graphics Screen
55Mx2
hp-bus
hp-bobcat
rs-232, 9600 Baud
Tape streamer
LaserDisk-1
DEC Pro-350
rs-232
rs-232
ASAKA SW
AES-300
LaserDisk-2
rs-232
rs-232
R-VTR
ASAKA
AEC-300
rs-422
rs-232
ASAKA MV
AEV-300
RGB Display
monitor

Video,Audio,&Sync

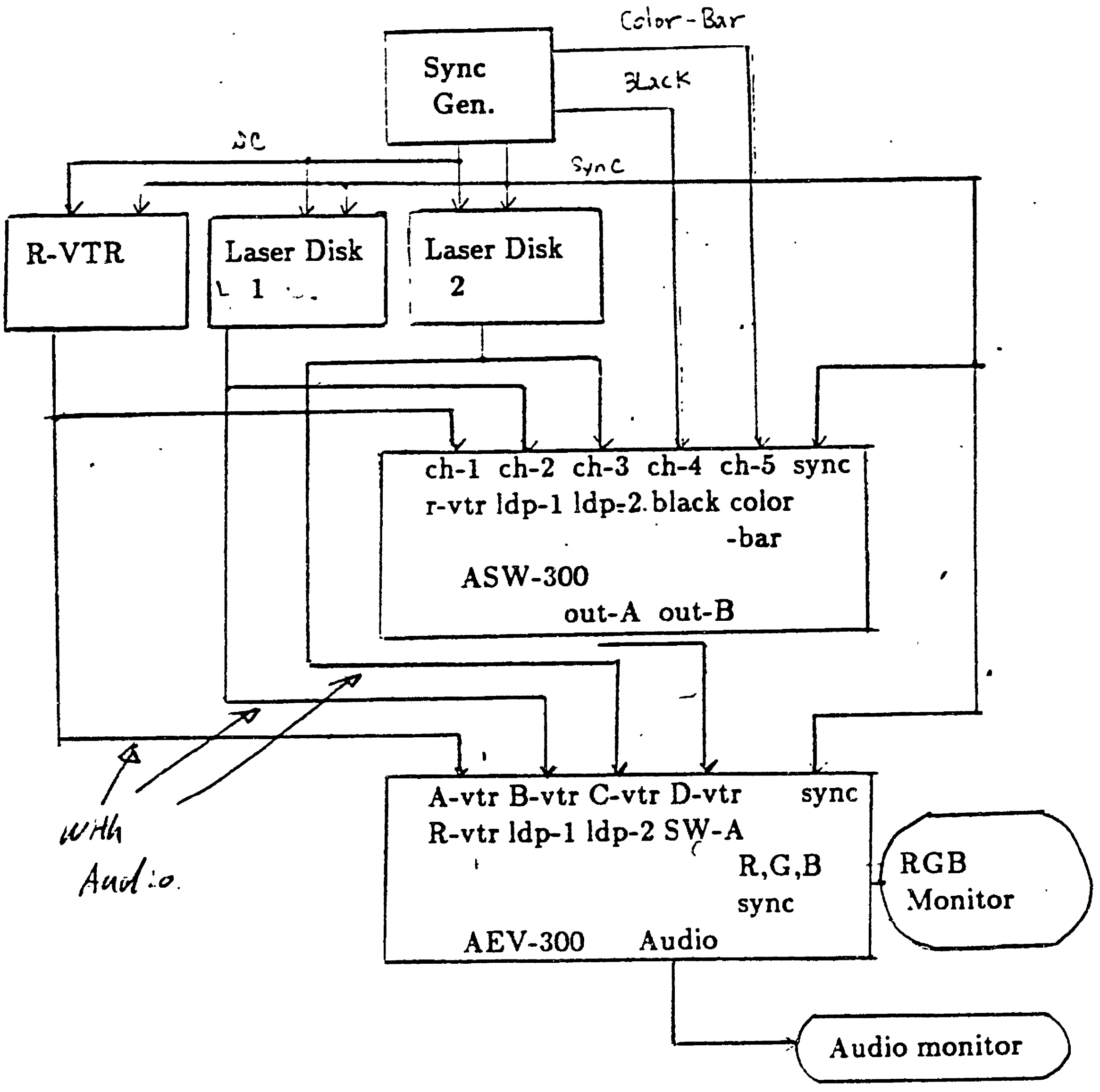


Diagram 3: Video Input Source Connections

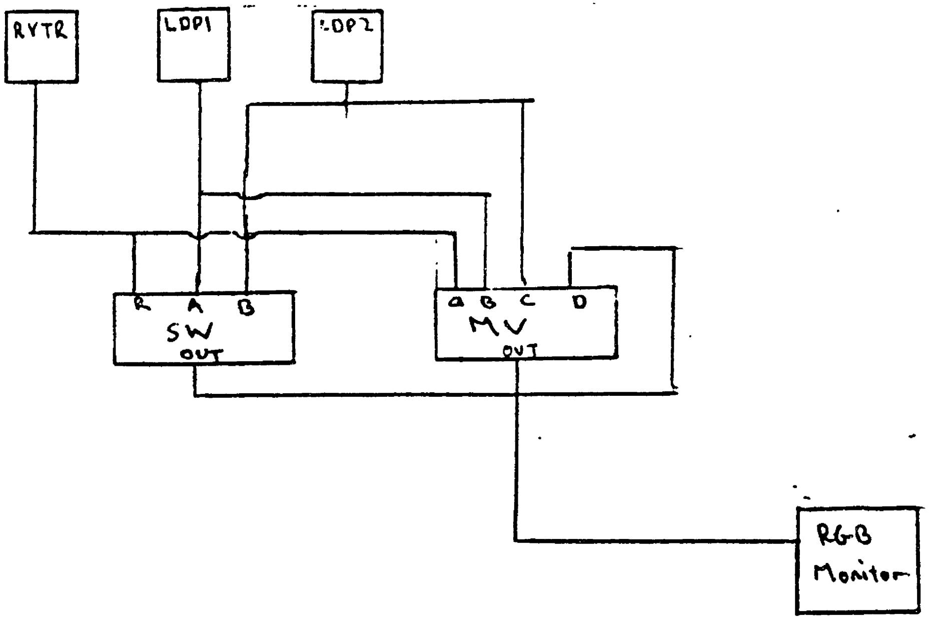
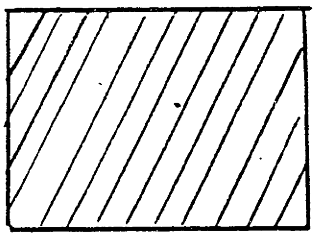
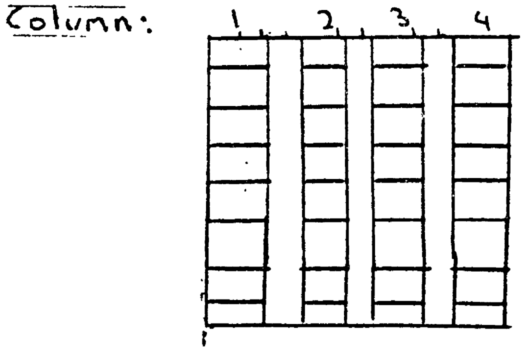


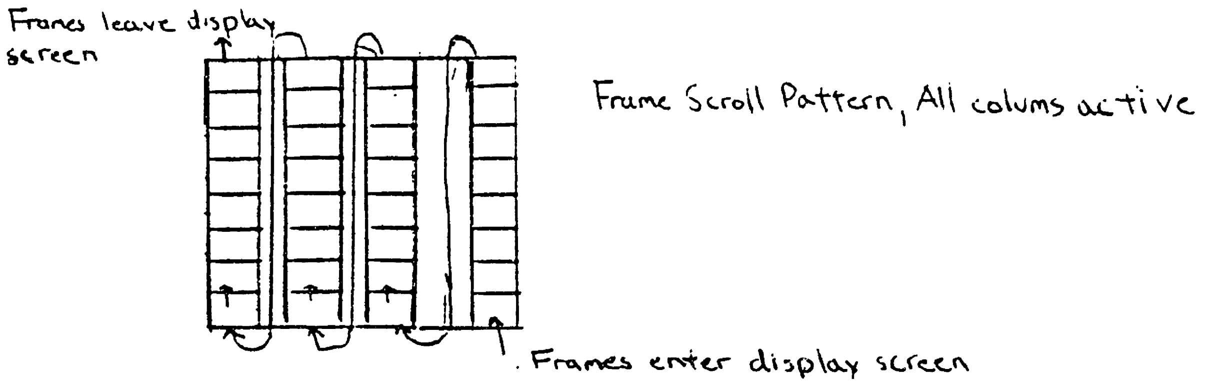
Diagram 4: Multiviewer Display



Normal Mode, Full Screen Display



Multiview Mode, 32 Digitized Frames


Frames leave display screen
Frame Scroll Pattern, All colums active
Frames enter display screen

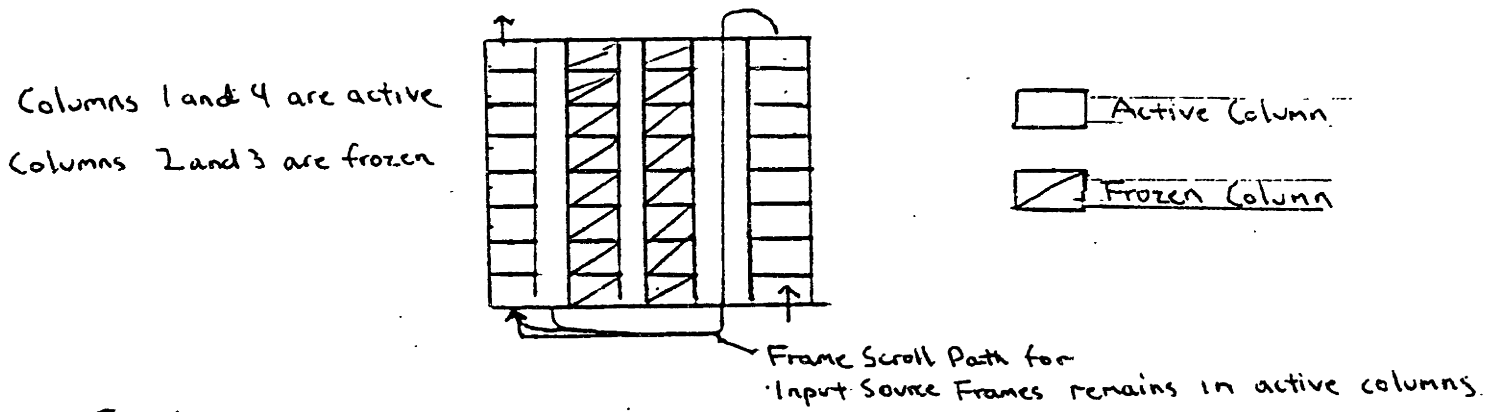
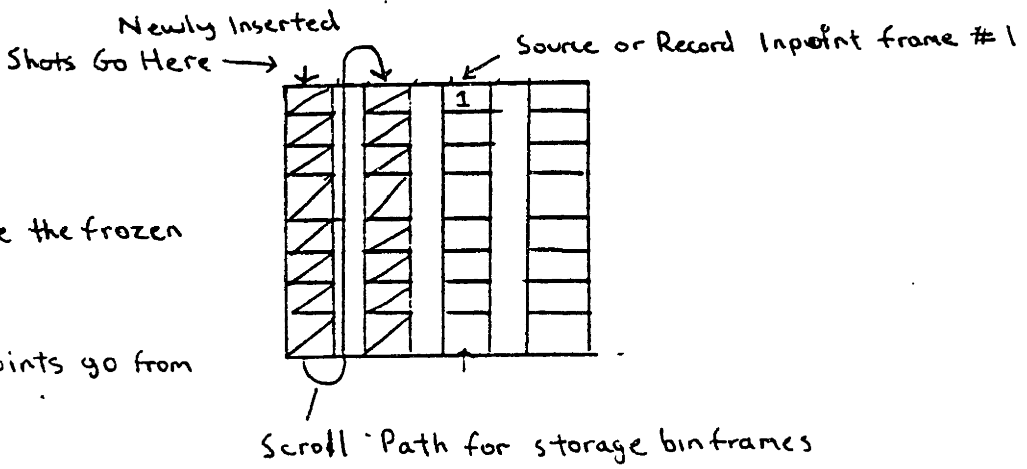

Columns 1 and 4 are active
Columns 2 and 3 are frozen
Active Column
Frozen Column
Frame Scroll Path for Input Source Frames remains in active columns

Diagram 5 : Active and Frozen Columns


Newly Inserted Shots Go Here
Source or Record Inpoint frame #1
1
Scroll Path for storage binframes

Column 3 is active

Columns 1 and 2 are the frozen frame storage bin.

Source or Record Inpoints go from Active frame 31 to frozen frame 11

Diagram 6 : Edlview.c and the multiview mode

LISTING 1: "SUBMAIN.C"

```
#define SCOPE
#include <stdio.h>
#include <fcntl.h>
#include <gened.h>
#include <funcs.h>
#include  "/usr/reza/global.h"


long tcno;


submain(file)   char *file;                                                              submain

{ int    result;        /* Result of control routine operation */

  int    time = 400;  /* Argument for sleep call */

  int    offset;        /* Offset into input command string. Only CMX commands //
                              are passed to control routines. */

  char temp[9];         /* location for cueing timecodes */


 offset = 0;                    /* Until a special command is received. "Ctl-E" /
                                         and "Ctl-G" are the special commands */

 bufptr = cmdlist;
 endbuffer = &(cmdlist[MAXCMDLIST]);
 if( (result = open(file, 0)) < 0 ){ printf("Bad command file: %s\n", file);
                                              control("^B");         return(-1);
                                                                  }
 if ( (result = read(result,cmdlist, MAXCMDLIST)) < 0 ) return(-1);
 edl = TRUE;
 close(result);

while( result != 66)

   {  offset = 0;
      endbuffer = &(cmdlist[MAXCMDLIST]);
      edl = TRUE;
      printf("\nEnter command: ");
      databuf[0] = getachar();

      switch( *databuf)
          {

            case   5:  checkscreen(); offset = 1; break;
                                                        /* Set mv to normal   //
                                                  sceen, no timecode displays */
```

```
        case  7:  printf("\nframe: "); scanf("%7ld" , &tcno);
                  convert(outframe); checkframe(); offset = 1; break;
                                                        /* Get destination    //
                 timecode value; when value is reached stop device motion */


       case 'n' : printf("\nlocation: ");
                  break;

        case 'N' : printf("\nlocation: ");
                  break;
        case 27: databuf[1] = getachar(); break;
        default: break;
       }
    if ( (result = control(databuf + offset)) < 0 )
                                                        /* "Control" executes /
action named by the given CMX command. It returns -1 for errors.See Appendix */

       printf("command execution error.\n");


  }
    printf("got q\n"); rtnkbd(); return(0);  /** Quit **/


}
```

```
#define SCOPE extern

#include <funcs.h>
#include <gened.h>
#include <string.h>
#include <sgtty.h>
#include <fcntl.h>
#include "/usr/reza/global.h"


 checkscreen()                  /* Setup normal screen, timecodes off. */        checkscreen
     { extern struct DEVTABLE devtable[];

       dispstat(MV1);
       if ( (devtable[MV1].status & BIT_5) == BIT_5)
             control("^[q");
                                                       /* Normal screen display
                                                       for mv */

       if ( (devtable[MV1].status & BIT_6) != BIT_6)
             control("^[s");
                                                         /* No timecode super—
                                                                imposition */

       if ( (devtable[MV1].status & BIT_2) != BIT_2)
              control("^[t");                           /* No timecode for  /
                                                          multiscreen cursor */
     }

 checkframe()                                                                      checkframe
     { extern TBYTE inframe[];
       extern long tcno;
       extern TBYTE outframe[];
       extern TBYTE curdevice;

       while( strcmp(outframe, inframe))        /* If current frame is not dest—
                                                       ination frame...      */
          getframe(curdevice, inframe);      /* update current frame (as device
                                                    motion continues) */
       control("V");       /** still device **/
     }


 convert(ptr) TBYTE *ptr;     /* John Barbour's handy timecode to framenumber/      convert
                                 conversion utility.*/
    { extern int  debug;
      extern long tcno;
             long x;
             long frame;

      tcno -= (long) 1000000;
      x  = tcno / (long) 10000;
      tcno = tcno - ( (long) 10000 * x);
```

LISTING 2: A decpro command file.

```
 This file was the end credit sequence at NAB. It cues the NAB disks to each
group member's credit, and moves a picture of that member to column 1 or 2 of
the multiview display mode. It also  writes their name alongside their picture
on the display.

The general command sequence to do this goes:
s n timecode ^[q ^[q 4 K 4

' s' or ' d' commands  selects a laser disk player.
' n' or ' N' cue to the immediately  following timecode location.
' ^[q' exits mutiview mode to show a full screen view of the group member.
' ^[q' a second time retuns to multiview mode.
' 4' freezes column 4 so that one frame may be copied.
' K' copies a shot of the group member into multiview display column 1 or 2.
' 4' reactivates column 4 as the active diaplay column for the next cue.

This command  sequence sort of repeats until everyone gets credit, and then the
write command sequence writes everyone' s names on the monitor screen.

Here is the complete command file:

*^Esn1213000^c^[q1v
dN1213429^^[q^c^[q2v
sn1213929^^[q^c^[q3v
dN1214514^^[q^c^[q4vK4112^4
sn1215604^^[q^c^[q4vK4113^4
dN1220600^^[q^c^[q4vK4114^4
sn1221602^^[q^c^[q4vK4115^4
dN1222529^^[q^c^[q4vK4116^4
sn1223608^^[q^c^[q4vK4117^4
dN1224604^^[q^c^[q4vK4124^4
sn1225602^^[q^c^[q4vK4125^4
dN1230602^^[q^c^[q4vK4126^4
sn1231604^^[q^c^[q4vK4127^4
^[j11^[j21^[j22^[j23^[j18^[j28
WMIT FILM/VIDEO
4,4
W<RICKY
6,12
W<GLORIANNA
10,12
W<KEISHI
14,12
W<RUSS
18,12
W<ANDRIA
22,12
W<REZA
26,12
W<JOHN
14,27
```

```
W<WIL
18,27
WKARL
22,27
W<MARK
26,27^^
^[q1234sc^G1295414q                                                60
```

LISTING 3: a bobcat command file.
Note that "/////" is just a place holder, not data.

```
q     /////  00:00
^[P   /////  00:05
n     24528  00:10
n     00001  00:20
c     /////  00:25
v     /////  00:35
d     /////  00:37
^[P   /////  00:40
n     24559  00:45
n     34000  00:50
c     /////  00:55
v     /////  01:05
s     /////  01:10
b     /////  01:15
v     /////  01:20
x     /////  01:30
z     /////  01:40
v     /////  01:50
s     /////  02:00
X     /////  02:10
```

LISTING 4: "DEMO.C", Version 1.

```
/* needs an argument file to initialize correctly ! */

#include <stdio.h>
#include <time.h>
#include <strings.h>
#include <sgtty.h>
#include <sys/ioctl.h>



#define MAXEVENTS 60
#define LENGTH 30          /*  event file line maximum no. of characters */
#define FIRST  0          /*  Index of first event in the array event_table */



struct event{

    char  cmd[2];
    char  time[8];
    char  options[12];

              };




init(file,table)          /* Gets CMX command characters from argument file and puts them into init_table */

char *file ;
struct event *table;

{ FILE *fp, *fopen() ;
  int i, j, k ;
  extern int total ;
  char  buff[LENGTH];

  if (( fp = fopen(file, "r")) == NULL)
     { printf(" can't open %s \n", file) ;
       exit(-1) ;
     }

  printf( "reading %s...\n",file) ;
  while (fgets(buff, LENGTH, fp) == buff)  /* Read one line from command file */
      {
         sscanf(buff,"%s %s %s %d", &table->cmd, &table->options,&table->time, ); /* Load values into the even

         printf("cmd: %s  time: %s ",table->cmd, table->time); /* Diagnostic printouts */
         printf("options:%s " , table->options);
         printf("no: %d\n", );
         table++; total++;                  /* Next event structure, increment running count of event structur
      }

   printf("last event was no.%d\n", (--table)->);  /* More dianostics */
   printf("%d events read\n", total) ;
   fclose(fp) ;
 }



int total = 0;                     /* Total number of command characters in the sequence */
```

main

```
main(argc,argv)
int argc ; char *argv[] ;

{ long  clock ;
  int   current;
  int  row, fd;
  FILE *fp;
  char *tstring, stime[5], key ;
  char *loc;
  struct event event_table[MAXEVENTS];
  struct sgttyb fdbuff;

  if(( fd = open("/dev/tty00", 2)) < 0) fprintf(stderr,"open error\n");

  ioctl(fd, TIOCGETP, &fdbuff);
  fdbuff.sg_ispeed = B9600;
  fdbuff.sg_ospeed = B9600;
  fdbuff.sg_flags |= 0;
  ioctl(fd, TIOCSETP, &fdbuff);
  fp = fdopen(fd, "w");

  init( argv[1], event_table);
  if (fp = fopen("/dev/tty00", "r") == NULL) {printf( "open error, tty00\n");
                                              exit(-1);
                                             }

  for(current = 0; current < total - 1; current++) /*Send command characters at their time*/
```

```
    {
       fprintf(stderr, "cmd: %s\n", event_table[current].cmd);

       fprintf(fp, "%s\n", event_table[current].cmd);  /* Transmit the command */

       switch(event_table[current].cmd[0])  /* Some CMX commands have options */
        {
             case  7 :
             case 'I' :
             case 'W' :
             case 'K' :
             case  20:
             case 'T' :
             case 'E' :
             case 'n' :  fprintf(stderr, "options: %s\n", event_table[current].options);
             case 'N' :  fprintf(fp,"%s\n",event_table[current].options); */ break;      /* Transmit the options, when
             default:    break;
       }
    }
    fclose(fp);
    close(fd);

}
```

LISTING 6: "EDLVIEW.C"

```
#include <stdio.h>

long tcno;

main ()                                                                    main

{ FILE *fp;                                         /* edl file pointer */

  short device;                           /* device  = 1 for ldp1, 2 for ldp2 */

  short i ;                             /* keeps count of edl line number */

  char  source_in[9];                    /* Source inpoint timecode string */

  char  record_in[9];                    /* Record    "       "       "    */

  extern char inframe[];       /* Global device control variable used to execute
                                 CMX "Cue to inframe" command */

  extern char outframe[];      /* Same as above, for outframe */


  if ( (fp = fopen( "my.edl", "r")) == NULL ) /* Open argument edl file. */
     {printf("open eror\n");
      return(-1);
     }

  if ( setup() ) {printf("ERROR IN INITIALIZATION\n"); exit(-1);}
                                                  /* Initialize devices et al. */
  allstop();
  submain("black1");


  for ( i = 0; i < 8 ; i++)  /* Edl files accessed line by line. Each
                                edl line access increments the loop
                                index, i, for a maximum 8 iterations, since
                                the  storage bin holds 16 frames.  */


     {

        while( fscanf(fp, "%3d %3d %*s %*s %*s  0 %1s : %2s : %2s : %2s %*s" ,
        &eventno, &device, source_in, souce_in, source_in + 1, source_in + 3,
                                                          source_in  + 5 ) == 6)


        /* Perform the multiview mode storage while the edl file
         provides valid information i.e. until end of file or edl data error */
      {
        fscanf(fp, " %2s : %2s : %2s : %2s %*s", record_in, record_in + 2,
                                              record_in + 4, record_in + 6);

    printf("points: 1 %s 2 %s     device: %d\n", source_in, record_in + 1, device);

    sscanf(source_in, "%7ld", &tcno);
    convert(inframe);

    sscanf(record_in + 1, "%7ld", &tcno);
    convert(outframe);

    submain("black");            /* Black out all columns in multiscreen */

    if (device == 2) submain("ldp2");    /* Which ldp is source ? */
    else             submain("ldp1");

    submain("qinframe");
    submain("column_3");                 /* Cue up ldp to source_in. Display
                                            8 source_in frames in column_3 */
    submain("insert");          /* Insert first frame of source_in at scene 11 */


    control("a");                        /* Select rvtr */
    submain("qoutframe");
    submain("column_3");                 /* Cue up rvtr to record_in    */
    submain("insert");    /* Insert first frame of record_in at scene 11 */
    submain("reset");
                              /* Reestablish loop starting conditions */

   }                  /* } while */
  }                   /* } for   */

 if ( i = 0) printf("%s was not quite recognizeable.\n", argv[1]);

/* No iterations means something wrong in edl file */

control("^[q"); /* Multiview mode to see bin display */
}
```

APPENDIX II

Documentaion for "control.c," prepared by Reza Jalili '89.

CONTROL

Control is the interface program between the keyboard and the various routines that drive the various devices. The following is detailed information about the structure of the program. Refer to the header files listed below for information about pre-defined values, structures, and types:

```
        <stdio.h>
        <fcntl.h>
        <sgtty.h>
        <gened.h>
        <funcs.h>
```

Three other header files hold defines for indexes into device command tables:

```
        "/usr/reza/ldpcmds.h"
        "/usr/reza/mvcmds.h"
        "/usr/reza/aswcmds.h"
```

The program, control(), defines several global variables that are used by functions in funcs.c. The following is a list of these global variables:

```
        1) STRUCTURE struct DEVTABLE devtable[MAXDEVICES]
        2) TBYTE cmdtbl[MAXDEVICES][128][4]
        3) TBYTE command[10],expect[10]
        4) TBYTE stdexp[] = {0,0x10,1}
        5) TBYTE TBYTE ssresult[15];
        6) LDPFLAG ldp1flag = 0;
        7) LDPFLAG ldp2flag = 0;
        8) MVFLAG mvflag;
        9) ASWFLAG aswflag = 0;
        10) TBYTE curdevice;
        11) TBYTE devices[MAXDEVICES];
        12) int kbd;
        13) TBYTE inframe[5],outframe[5];
        14) EDIT1FLAG ef1;
        15) EDIT2FLAG ef2;
```

The array of DEVTABLEs, called devtable, is used to store information about each device that is opened. Refer to the header file funcs.h for information on the structure itself.

cmdtbl[] is an array that holds strings of commands for each of the opened devices. The array is allowed a maximum length of 128 4 byte strings. Information about the structure of the table can be found in loadtbl() in funcs.c.

The two global arrays, command[] and expect[] are used to send and receive strings of bytes. The array stdexp[] holds the standard expected reply from a multiviewer. The reply is common enough to justify the array.

The array ssresult[] is filled in by sndstring() and holds the bytes sent by a device. Only the first 15 bytes are kept. Only 15 bytes are read.

A description of the flags can be found in funcs.h.

curdevice holds the value of the current playing device. That is, it holds the value of the device that will be sent any commands such as play, rec, forward, rewind,etc. The value of curdevice is one of the values defined in funcs.h for the various devices(LDP1, LDP2, MV1,...)

devices[] is an array that links the devices' values to the channels to which they are connected on the switcher. A device's value is it's offset into the devtable[] array. For example, if laser disc player 1 is connected to channel 3 of the switcher, and it's information is in the structure devtable[1], then devices[1] = 3; more generally devices[LDP1] = BVTR.

inframe and outframe hold the frame numbers sent by the playing device.

ef1 and ef2 are the general flags about the system as a whole. see funcs.h.

The program begins by opening the keyboard at 9600 baud and with the CBREAK flag turned on. This flag lets charcaters be read without waiting for a <CR>:
Next, the laser disc player is opened at the correct baud rate and parity settings:
The multiviewer is opened at 9600 baud and with no parity:
With all the devices open, the devtable array is filled:

```
        devtable[LDP1].fd = ldp;
        devtable[MV1].fd = mv;
        devtable[SWTR].fd = mv;                /* for now, share same line */
        devtable[LDP1].status = 0;             /* nothing for now */
        devtable[MV1].status = 0;              /* nothing for now */
        devtable[SWTR].status = 0;             /* nothing for now */
        devtable[LDP1].table = 0;
        devtable[MV1].table = 1;
        devtable[SWTR].table = 2;
```

The command tables are loaded in to the correct arrays:

```
        loadtbl("mvcmds",&cmdtbl[devtable[MV1].table][0][0]);
        loadtbl("ldpcmds",&cmdtbl[devtable[LDP1].table][0][0]);
        loadtbl("swtrcmds",&cmdtbl[devtable[SWTR].table][0][0]);
```

The program then simply loops, getting keys, testing them against set values, and calling the appropriate functions in fucns.c. Refer to the individual function headers in funcs.c for information about functions.

Control is the interface program between the keyboard and the various devices. Devices include laser disc players, the Multi Viewer machine, VTRs, switchers, and any other video equipment. Currently, the program can access two laser disc players (LDP1 and LDP2), one Multi Viewer (MV), and one switcher (ASW). Of course, the program could handle more, but it has been "configured" to run those three devices for now.

Before running the program, turn on the Multi Viewer, both 600 baud LDP (the one on top) and 4800 baud LDP, and the ASACA switcher.
Turn on the Toshiba monitor and set it to Video 1 for the small screen, and RGB2 on the large screen.

The set-up of the keyboard has been modeled after that of The Grass Valley Group's Super Edit Keyboard. Function keys are used for many effects. On keyboards without these function keys, a two-key sequence of an escape <ESC> character and a regular character (e.g. <ESC>q) will result in an identical action.
The keys implemented so far are the following: (in no order!)

KEY	ACTION
===	======
'c'	Put LDP in forward play mode.
'V'	Stop LDP.
'v'	Put LDP in still mode. (does not work)
'B'	Put LDP in reverse slow motion mode.
'b'	Put LDP in forward slow motion mode.
'X'	Put LDP in forward fast scan mode. (does not work)
'x'	Put LDP in fast forward mode. (does not work)
'Z'	Put LDP in reverse slow motion mode.

'z'	Put LDP in fast reverse mode.
control-b	Reset LDP.
'+'	Clear LDP error flags.
'P' or 'p'	Have switcher cut from A to B and vice-versa.
'A' or 'a'	Tell MV to switch to no VTR.
'S' or 's'	Tell MV to switch to VTR A.
'D' or 'd'	Tell MV to switch to VTR B.
'r'	Reset the time to 00:00:00:01. (does not work)
'w'	Get current frame number from LDP. (does not work)
'^'	Pause
'q'	Close all devices and exit from program.
ESC-'q'	Toggle between normal and multi-frame screen.
'r'	Toggle LDP motor on and off. (does not work)
's'	Toggle super-imposing of time code on and off.
't'	Toggle the use of time code on and off.
'A'	Move the cursor up.
'B'	Move the cursor down.
'C'	Move the cursor left.
'D'	Move the cursor right.
'P'	Toggle indexing of LDP on and off. (does not work)
'Q'	Toggle between LDP segment and frame mode.

```
/* R. Jalili 4-10-86 */


/* FILM/VIDEO MEDIA LAB */

#define SCOPE extern

#include <stdio.h>
#include <fcntl.h>
#include <gened.h>
#include <funcs.h>
#include "include/ldpcmds.h"
#include "include/mvcmds.h"
#include "include/rvtrcmds.h"
#include "include/aswcmds.h"
#include "global.h"

extern TBYTE *sndstring();

void quitall();

/********** CONTROL EQUIPMENT
 *
 * Call:        result = control(string holding command          TBYTE *;);
 *-
 * Function:    Calls the appropriate routine for the given command.
 *              A command string is only one character.  IF the character
 *              is ESC (27), then the next character is used.
 *              ESC-r is a command, wherer the first byte is 27 and the
 *              second byte is 18.  This is to allow escape combinations.
 *
 * Returns:     0 if ok, -1 if not.  66 if quit is received and connections
 *                                  are closed.
 *
*/
int
FUNCTION control(s)
TBYTE *s;
{ int key,key2,c,result;

  result = 0;
  key = s[0];
  if ( key == 27 ) key2 = s[1];
  if ( curdevice == RVTR1)
      switch (key)                            /* RVTR commands */
          {case 'C' :    result = rvshtl(curdevice,-PLAYSPD);
                         result += setmvdir(MV1, REVERSE);
                         ACTIVITY(curdevice, BIT_14 | BIT_15, BIT_13 | BIT_12)
/* speed flag = norm */  ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                         break;
           case 'c' :     CALLSS(curdevice,RVPLAY)  /*macrocall sndstring()*/
                         result += setmvdir(MV1, FORWARD);
                         ACTIVITY(curdevice, BIT_12 | BIT_15| BIT_14, BIT_13)
```

```
/* speed flag = norm */ ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                        break;
        case 'V' :      CALLSS(curdevice,RVSTOP)       /* no ; */
                        ACTIVITY(curdevice, BIT_14|BIT_15|BIT_13|BIT_12, 0)
/* speed flag = stop */ ACTIVITY(curdevice,BIT_1 | BIT_2 | BIT_3,0 );
                        break;
        case 'v' :      CALLSS(curdevice,RVSTOP);
                        ACTIVITY(curdevice, BIT_14|BIT_15|BIT_13|BIT_12,0)
/* speed flag = stop */ ACTIVITY(curdevice,BIT_1 | BIT_2 | BIT_3,0 );
                        break;
        case 'B' :      result = rvjog(curdevice,-1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_14, BIT_12|BIT_15|BIT_13 )
                         break;
        case 'b' :      result = rvjog(curdevice,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_14 | BIT_12, BIT_13|BIT_15 )
                        break;
        case 'X' :      result = rvshtl(curdevice,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_15 | BIT_12,BIT_13 | BIT_14)
                        break;
        case 'x' :      CALLSS(curdevice,RVFF)
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_15 | BIT_12, BIT_13 | BIT_14)
/* speed flag = norm */ ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                        break;
        case 'Z' :      result = rvshtl(curdevice,-1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_15, BIT_12| BIT_13 | BIT_14)
                        break;
        case 'z' :      CALLSS(curdevice,RVREW)
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_15,BIT_12| BIT_13 | BIT_14)
/* speed flag = norm */ ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                        break;
        case 'm' :      result = rvmark(curdevice,inframe);
                        break;
        case ',' :      result = rvmark(curdevice,outframe);
                        break;
        case 'n' :      result = rvcue(curdevice,inframe);
                        break;
        case 'N' :      result = rvcue(curdevice,outframe);
                        break;
        case 'r' :      CALLSS(curdevice,RVPREROLL)
                        break;
        case 'i' :      result = rvsetin(curdevice);
                        break;
        case 'o' :      result = rvsetout(curdevice);
                        break;
        case 'u' :      result = rvvideo(curdevice);
                        break;
        case 'y' :      result = rvaudio(curdevice);
```

```
                        break;
        case   9:       CALLSS(curdevice,RVEDITON);
                        break;
        case '@' :      result = rvsetspeed(curdevice);break;
        case   2:       break;
        case '+' :      break;
        case  27:       {switch (key2)
                            {case 'r' : if ( devtable[curdevice].status
                                              & BIT_0 )
                                             {CALLSS(curdevice,RVSBON)
                                             }
                                        else {CALLSS(curdevice,RVSBOFF)
                                              }
                                        if ( result == 0 ) TOGGLE(
                                           devtable[curdevice].status,BIT_0)
                                        break;
                            }
                         break;
                        }
        default:        break;
      }
  if ( curdevice != RVTR1)
     switch (key)                               /* LDP1 and LDP2 commands */
        {case 'C' :     result = send(curdevice, RPLAY,1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_14 | BIT_15, BIT_13 | BIT_12)
                        break;
         case 'c' :     result = send(curdevice, FPLAY,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_12 | BIT_15| BIT_14, BIT_13)
                        break;
         case 'V' :     result = send(curdevice,STOP,1);
                        ACTIVITY(curdevice, BIT_14|BIT_15|BIT_13|BIT_12, 0)
                        break;
         case 'v' :     result = send(curdevice, STILL,1);
                        ACTIVITY(curdevice, BIT_14|BIT_15|BIT_13|BIT_12,0)
                        break;
         case 'B' :     result = send(curdevice, RSLOW,1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_14, BIT_12|BIT_15|BIT_13 )
                        break;
         case 'b' :     result = send(curdevice, FSLOW,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_14 | BIT_12, BIT_13|BIT_15 )
                        break;
         case '{' :     result = send(curdevice, RSTEP,1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_14, BIT_12|BIT_13|BIT_15 )
                        break;
         case '}' :     result = send(curdevice, FSTEP,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_12|BIT_14, BIT_13|BIT_15 )
                        break;
        case 'X' :      result = send(curdevice, FSCAN,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_15 | BIT_12,BIT_13 | BIT_14)
                        break;
        case 'x' :      result = send(curdevice, FFAST,1);
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_15 | BIT_12, BIT_13 | BIT_14)
                        break;
        case 'Z' :      result = send(curdevice, RSCAN,1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_15, BIT_12| BIT_13 | BIT_14)
                        break;
        case 'z' :      result = send(curdevice, RFAST,1);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_15,BIT_12| BIT_13 | BIT_14)
                        break;
        case 'm' :      result = mark(curdevice,inframe);
                        break;
        case ',' :      result = mark(curdevice,outframe);
                        break;
        case 'n' :      result = cue(curdevice,inframe);break;
        case 'N' :      result = cue(curdevice,outframe);break;
        case   2:       result = send(curdevice, CL,1);break;
        case '+' :      result = send(curdevice, CE,1);break;
        case  27:       {switch (key2)
                            {case 'r' : result = mtrcntl(curdevice);
                                        break;
                             case 'P' : result = indxctl(curdevice);break;
                                                /* indx on/off */
                             case 'Q' : result = modefs(curdevice);break;
                                                /* seg/frm mode */
                            }
                         break;
                        }
        default:        break;
    }
    switch (key)                                /* other commands */
       {case '^' :      snore(10); break;
        case '*' :       debug = !debug; break;    /* toggle tty output */
        case '?' :      statprint();break;
        case ' ' :      result = allstop();break;
        case '&' :      result = setframe();break;
        case 'E' :      result = scnxchng(MV1); break;
        case 'W' :      result = scrnwrite(MV1,15);break;
        case '(' :      result = chngcolor(MV1);break;
        case '1' :      result = rollcntl(MV1,1);break;
        case '2' :      result = rollcntl(MV1,2);break;
        case '3' :      result = rollcntl(MV1,3);break;
        case '4' :      result = rollcntl(MV1,4);break;
        case '!' :      result = charclear(MV1); break;
        case 'R' :      result = scnreplace(MV1,ROLL_1);  /* roll 1 */
                        result += scnreplace(MV1,ROLL_2);       /* roll 2 */
                        result += scnreplace(MV1,ROLL_3);       /* roll 3 */
```

```
/* R. Jalili 4-10-86 */


/* FILM/VIDEO MEDIA LAB */

#define SCOPE extern

#include <stdio.h>
#include <fcntl.h>
#include <gened.h>
#include <funcs.h>
#include "include/ldpcmds.h"
#include "include/mvcmds.h"
#include "include/rvtrcmds.h"
#include "include/aswcmds.h"
#include "global.h"

extern TBYTE *sndstring();

void quitall();

/********** CONTROL EQUIPMENT
 *
 * Call:          result = control(string holding command          TBYTE *;);
 *
 * Function:      Calls the appropriate routine for the given command.
 *                A command string is only one character.  IF the character
 *                is ESC (27), then the next character is used.
 *                ESC-r is a command, wherer the first byte is 27 and the
 *                second byte is 18.  This is to allow escape combinations.
 *
 * Returns:       0 if ok, -1 if not.  66 if quit is received and connections
 *                                        are closed.
 *
*/
int
FUNCTION control(s)
TBYTE *s;
{ int key,key2,c,result;

  result = 0;
  key = s[0];
  if ( key == 27 ) key2 = s[1];
  if ( curdevice == RVTR1)
      switch (key)                              /* RVTR commands */
         {case 'C' :    result = rvshtl(curdevice,-PLAYSPD);
                        result += setmvdir(MV1, REVERSE);
                        ACTIVITY(curdevice, BIT_14 | BIT_15, BIT_13 | BIT_12)
/* speed flag = norm */ ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                        break;
          case 'c' :     CALLSS(curdevice,RVPLAY)  /*macrocall sndstring()*/
                        result += setmvdir(MV1, FORWARD);
                        ACTIVITY(curdevice, BIT_12 | BIT_15| BIT_14, BIT_13)
```

```
/* R. Jalili 4-10-86 */


/* FILM/VIDEO MEDIA LAB */

#define SCOPE extern

#include <stdio.h>
#include <fcntl.h>
#include <gened.h>
#include <funcs.h>
#include "include/ldpcmds.h"
#include "include/mvcmds.h"
#include "include/rvtrcmds.h"
#include "include/aswcmds.h"
#include "global.h"

extern TBYTE *sndstring();

void quitall();

/********** CONTROL EQUIPMENT
 *
 * Call:        result = control(string holding command        TBYTE *;);
 *
 * Function:    Calls the appropriate routine for the given command.
 *              A command string is only one character.  IF the character
 *              is ESC (27), then the next character is used.
 *              ESC-r is a command, wherer the first byte is 27 and the
 *              second byte is 18.  This is to allow escape combinations.
 *
 * Returns:     0 if ok, -1 if not.  66 if quit is received and connections
 *                                   are closed.
 *
*/
int
FUNCTION control(s)
TBYTE *s;
{ int key,key2,c,result;

  result = 0;
  key = s[0];
  if ( key == 27 ) key2 = s[1];
  if ( curdevice == RVTR1)
      switch (key)                         /* RVTR commands */
         {case 'C' :  result = rvshtl(curdevice,-PLAYSPD);
                      result += setmvdir(MV1, REVERSE);
                      ACTIVITY(curdevice, BIT_14 | BIT_15, BIT_13 | BIT_12)
/* speed flag = norm */ ACTIVITY(curdevice,0,BIT_1 | BIT_2 | BIT_3 );
                      break;
          case 'c' :   CALLSS(curdevice,RVPLAY)  /*macrocall sndstring()*/
                      result += setmvdir(MV1, FORWARD);
                      ACTIVITY(curdevice, BIT_12 | BIT_15| BIT_14, BIT_13)
```

```
        {
          fscanf(fp, " %2s : %2s : %2s : %2s %*s", record_in, record_in + 2,
                                                   record_in + 4, record_in + 6);

      printf("points: 1 %s 2 %s     device: %d\n", source_in, record_in + 1, device);

      sscanf(source_in, "%7ld", &tcno);
      convert(inframe);

      sscanf(record_in + 1, "%7ld", &tcno);
      convert(outframe);

      submain("black");            /* Black out all columns in multiscreen */

      if (device == 2) submain("ldp2");     /* Which ldp is source ? */
      else             submain("ldp1");

      submain("qinframe");
      submain("column_3");                 /* Cue up ldp to source_in. Display
                                              8 source_in frames in column_3 */
      submain("insert");         /* Insert first frame of source_in at scene 11 */


      control("a");                        /* Select rvtr */
      submain("qoutframe");
      submain("column_3");                 /* Cue up rvtr to record_in    */
      submain("insert");   /* Insert first frame of record_in at scene 11 */
      submain("reset");
                              /* Reestablish loop starting conditions */

    }                      /* } while */
  }                        /* } for   */

  if ( i = 0) printf("%s was not quite recognizeable.\n", argv[1]);

/* No iterations means something wrong in edl file */

control("^[q"); /* Multiview mode to see bin display */
}
```

```
        {
          fscanf(fp, " %2s : %2s : %2s : %2s %*s", record_in, record_in + 2,
                                                  record_in + 4, record_in + 6);

      printf("points: 1 %s 2 %s      device: %d\n", source_in, record_in + 1, device);

      sscanf(source_in, "%7ld", &tcno);
      convert(inframe);

      sscanf(record_in + 1, "%7ld", &tcno);
      convert(outframe);

      submain("black");            /* Black out all columns in multiscreen */

      if (device == 2) submain("ldp2");     /* Which ldp is source ? */
      else             submain("ldp1");

      submain("qinframe");
      submain("column_3");                  /* Cue up ldp to source_in. Display
                                                8 source_in frames in column_3 */
      submain("insert");        /* Insert first frame of source_in at scene 11 */


      control("a");                         /* Select rvtr */
      submain("qoutframe");
      submain("column_3");                  /* Cue up rvtr to record_in     */
      submain("insert");  /* Insert first frame of record_in at scene 11 */
      submain("reset");
                                /* Reestablish loop starting conditions */

    }                     /* } while */
  }                       /* } for   */

 if ( i = 0) printf("%s was not quite recognizeable.\n", argv[1]);

/* No iterations means something wrong in edl file */

control("^[q"); /* Multiview mode to see bin display */
}
```

LISTING 6: "EDLVIEW.C"

```
#include <stdio.h>

long tcno;

main ()                                                                    main

{ FILE *fp;                                          /* edl file pointer */

  short device;                         /* device  = 1 for ldp1, 2 for ldp2 */

  short i ;                          /* keeps count of edl line number */

  char   source_in[9];                /* Source inpoint timecode string */

  char   record_in[9];                /* Record    "       "        "    */

  extern char inframe[];        /* Global device control variable used to execute
                                   CMX "Cue to inframe" command */

  extern char outframe[];       /* Same as above, for outframe */



  if ( (fp = fopen( "my.edl", "r")) == NULL ) /* Open argument edl file. */
     {printf("open eror\n");
      return(-1);
     }

  if ( setup() ) {printf("ERROR IN INITIALIZATION\n"); exit(-1);}
                                              /* Initialize devices et al. */
  allstop();
  submain("black1");


  for ( i = 0; i < 8 ; i++)  /* Edl files accessed line by line. Each
                                 edl line access increments the loop
                                 index, i, for a maximum 8 iterations, since
                                 the  storage bin holds 16 frames.  */


     {

        while( fscanf(fp, "%3d %3d %*s %*s %*s  0 %1s : %2s : %2s : %2s %*s" ,
        &eventno, &device, source_in, souce_in, source_in + 1, source_in + 3,
                                                        source_in   + 5 ) == 6)


        /* Perform the multiview mode storage while the edl file
         provides valid information i.e. until end of file or edl data error */
```

LISTING 6: "EDLVIEW.C"

```
#include <stdio.h>

long tcno;

main ()                                                              main

{ FILE *fp;                                  /* edl file pointer */

  short device;                      /* device  = 1 for ldp1, 2 for ldp2 */

  short i ;                        /* keeps count of edl line number */

  char  source_in[9];               /* Source inpoint timecode string */

  char  record_in[9];               /* Record    "        "        "      */

  extern char inframe[];     /* Global device control variable used to execute
                                CMX "Cue to inframe" command */

  extern char outframe[];   /* Same as above, for outframe */

       ,

  if ( (fp = fopen( "my.edl", "r")) == NULL ) /* Open argument edl file. */
     {printf("open eror\n");
      return(-1);
     }

  if ( setup() ) {printf("ERROR IN INITIALIZATION\n"); exit(-1);}
                                                /* Initialize devices et al. */
  allstop();
  submain("black1");


  for ( i = 0; i < 8 ; i++)  /* Edl files accessed line by line. Each
                                  edl line access increments the loop
                                  index, i, for a maximum 8 iterations, since
                                  the  storage bin holds 16 frames.  */


      {

         while( fscanf(fp, "%3d %3d %*s %*s %*s  0 %1s : %2s : %2s : %2s %*s" ,
         &eventno, &device, source_in, souce_in, source_in + 1, source_in + 3,
                                                         source_in   + 5 ) == 6)


         /* Perform the multiview mode storage while the edl file
          provides valid information i.e. until end of file or edl data error */
```