Capture file Fixes

17 February 2005

TABLE OF CONTENTS

Section **Page**

Note

1. In all of the following sample outputs/inputs the actual file has been edited. This has been done to change the format of the `RCS` keywords by adding a blank ("␣") after the leading dollar sign ("`$`").
2. All comments, lines beginning with a per-cent sign "%", have been deleted in the examples to save paper.

[1] `$Header: d:/txb-p/Captures/RCS/DocFix.tex,v 1.25 2005-02-17 12:26:12-08 Hamilton Exp Hamilton $`

[2] `tocdepth = 2`

1 How to save serial communications data

1.1 How to save data with FTS

Utilizing the new asynchronous data capture program from FrontLine Test Systems' named "Serialtest Async" requires that it be set up as follows to capture data from a Panasonic WV-CU161 keyboard in RS-485 four wire mode:

1. `Configuration: Monitor Both 19200,N,8,1 19200,N,8,1 DTE=COM4, DCE=COM5`
2. `Capture file:E:\Capture\test`, or where ever you want to put it.
3. `Bit Order: LSB first (normal)`
4. `Hardware Settings: Use FTS Cables`
5. `System Settings: Wrap Buffer is selected`
6. `System Settings: Capture Buffer Size (in K): 2052`
7. `System Settings: File Size (in K): 4102`
8. `Program Start Up Options: Prompt for a filename, then immediately start capturing.`
9. `Advanced System Options: <all at defaults>`

After getting every thing turned on, click on the correct Icon, fool around a bit and then select "`Live: Start Capture to Disk...`".

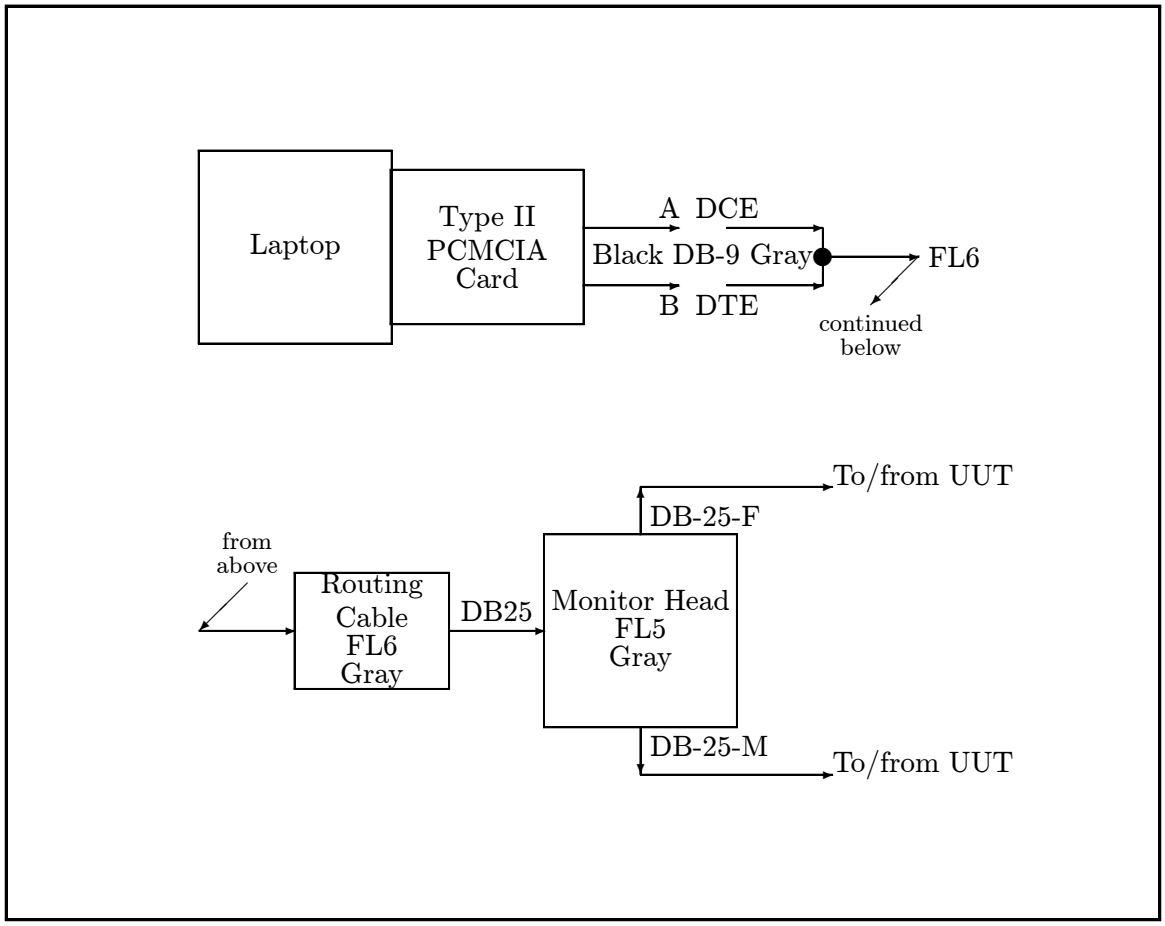
To output the captured data:

1. `Export Events...`
2. `Apply Template: Sample Events`
3. `Displayed Fields: Side Timestamp Hexadecimal`
4. `Text Output`
5. `Separate Records With CR/LF`
6. `Output Header`
7. `Output Field Name Record`
8. `Align Field Names With Data`
9. `Timestamp Format: Native`

[3] `$Header: d:/txb-p/Captures/RCS/FtsSave.inc,v 1.2 2005-02-17 12:26:13-08 Hamilton Exp Hamilton $`

10. `Character Set: ASCII`
11. `Signals Characters: 1/0`
12. `Errors Characters: X/<sp>`
13. `Field Delimiter: Space`
14. `Filter Out: Neither Side`
15. `Export: Entire Buffer`
16. `As: Serialtest Async Events.txt` or other name of your choice

[4] `$Header: d:/txb-p/Captures/RCS/InterCnt.inc,v 1.1 2005-02-17 12:15:22-08 Hamilton Exp Hamilton $`



$RCSfile: InterCnt.inc,v $

Figure 1. Connections for use with FrontLine Systems' version of Breakout `[intercnt]`

1.2 Saving capture files with BREAKOUT

The BREAKOUT program may be used to save communications line data captures. To do this:

1. Start BREAKOUT running.
2. Verify that the communication parameters are correct:
   - 2.1. Inside COMMUNICATIONS (Alt-C):
     - 2.1.1. MONITOR is selected
     - 2.1.2. NORMAL is selected
     - 2.1.3. SINGLE PORT SETTINGS is selected
     - 2.1.4. Inside PORT SETTINGS...
       - 2.1.4.1. BAUD RATE... is correct.
       - 2.1.4.2. DATA BITS... is correct. Default is eight or "8".
       - 2.1.4.3. PARITY... is correct. Default is None or "N".
       - 2.1.4.4. STOP BITS... is correct. Default is one or "1".
   - 2.2. Inside VIEW (Alt-V):
     - 2.2.1. ONLINE is selected.
     - 2.2.2. SINGLE is selected. Other choices are acceptable.
     - 2.2.3. DISPLAY is selected.
     - 2.2.4. UNDERLINE CURSOR is selected. Don't really what this does.
     - 2.2.5. Inside DISPLAY CHARACTER SET... Select something. These don't effect the capture data, but they do make the display meaningful.
   - 2.3. Inside OPTIONS (Alt-O):
     - 2.3.1. Select CAPTURE
     - 2.3.2. Do not select TIMESTAMP, it causes BREAKOUT to crash after awhile and DB does not process its data correctly.
3. Collect a bunch of data. As this is happening the data will be written to the screen and saved in memory.
4. When you want to make a disk copy of your captured data:
   - 4.1. Inside VIEW (Alt-V):
     - 4.1.1. Select Editor
       - 4.1.1.1. Select FILE (Alt-F):
       - 4.1.1.2. While in FILE select SAVE AS

[5] `$Header: d:/txb-p/Captures/RCS/BreakSav.inc,v 1.2 2004-12-01 09:35:27-08 Hamilton Exp Hamilton $`

4.1.1.3. Then take the time tag that shows up in ENTER DESCRIPTOR.

4.1.1.4. Now select SAVE WITHOUT CHANGE.

4.1.1.5. Enter in a file name to use in saving the data. I prefer to use the data and an incrementing last letter. BREAKOUT will append a `.CAP` to your name.
At this point the data will be written to the hard disk. If there is a lot of data this may take a minute or so.

5. There are other options which I do not need to use and thus don't know how to use them.

6. Be sure to copy your data from the hard disk to a floppy for analyzing.

1.3 Saving Oscilloscope Pictures

With a Tektronix TDS220 (and other similar Tektronix Oscilloscopes) configure the printer port as follows:

1. Hard Copy Setup:
   - 1.1. LAYOUT Portrait
   - 1.2. FORMAT EPSIMAGE
   - 1.3. PORT RS232
2. RS-232 setup:
   - 2.1. BAUD 19,200
   - 2.2. FLOW CONTROL None
   - 2.3. EOL STRING CR/LF
   - 2.4. PARITY None
3. Computer setup.
   - 3.1. Boot the computer into "real DOS" not "DOS in a window" if using Kermit to collect data.
   - 3.2. On the computer set it up to receive serial data.
   - 3.3. Use a "null modem" or "Laplink" type cable to connect the two together.
   - 3.4. Using this configuration, it will take 1 to 2 minuets to transfer the data from the oscilloscope to the picture.

[6] `$Header: d:/txb-p/Captures/RCS/ScopeSav.inc,v 1.2 2004-12-01 10:20:34-08 Hamilton Exp Hamilton $`

2 General use routines

2.1 FormatFx.l

2.1.1 Description

This routine is used to reformat ASCII records written by FrontLine Test Systems' serial data line capture software named "Serialtest Async".

2.1.2 Calling Sequence

```
FormatFx infile outfile
```

2.1.3 Input Data Format

The capture data must be saved with the following fields in the following order:

1. **Date/Time:** "8/26/2004␣12:06:35.060␣PM"
2. **Data:** "02"

Typical first few lines of the capture file **must** be as follows:

```
 1  FTS capture file: E:\Capture\newtest.cfa (8/26/2004 12:34:27 PM)
 2  Event 1 (8/26/2004 12:06:06.434 PM) through
 3  Event 30,910 (8/26/2004 12:32:44.595 PM)
 4
 5  Timestamp                 Hx
 6  8/26/2004 12:06:06.434 PM
 7  8/26/2004 12:06:35.060 PM 02
 8  8/26/2004 12:06:35.060 PM 41
 9  8/26/2004 12:06:35.060 PM 44
10  8/26/2004 12:06:35.060 PM 30
11  8/26/2004 12:06:35.060 PM 31
12  8/26/2004 12:06:35.060 PM 3b
13  8/26/2004 12:06:35.060 PM 52
14  8/26/2004 12:06:35.066 PM 4f
15  8/26/2004 12:06:35.066 PM 4e
16  8/26/2004 12:06:35.066 PM 3a
17  8/26/2004 12:06:35.066 PM 30
18  8/26/2004 12:06:35.066 PM 03
19  8/26/2004 12:06:35.106 PM 02
20  8/26/2004 12:06:35.106 PM 41
21  8/26/2004 12:06:35.106 PM 44
22  8/26/2004 12:06:35.106 PM 30
23  8/26/2004 12:06:35.112 PM 31
24  8/26/2004 12:06:35.112 PM 3b
25  8/26/2004 12:06:35.112 PM 47
26  8/26/2004 12:06:35.112 PM 43
27  8/26/2004 12:06:35.112 PM 37
28  8/26/2004 12:06:35.112 PM 3a
```

```
29  8/26/2004 12:06:35.112 PM 32
30  8/26/2004 12:06:35.112 PM 30
31  8/26/2004 12:06:35.112 PM 32
32  8/26/2004 12:06:35.112 PM 30
33  8/26/2004 12:06:35.112 PM 30
34  8/26/2004 12:06:35.112 PM 30
35  8/26/2004 12:06:35.119 PM 30
36  8/26/2004 12:06:35.119 PM 03
37  8/26/2004 12:06:36.141 PM 02
38  8/26/2004 12:06:36.141 PM 41
39  8/26/2004 12:06:36.148 PM 44
40  8/26/2004 12:06:36.148 PM 30
41  8/26/2004 12:06:36.148 PM 31
42  8/26/2004 12:06:36.148 PM 3b
43  8/26/2004 12:06:36.148 PM 47
44  8/26/2004 12:06:36.148 PM 43
45  8/26/2004 12:06:36.148 PM 37
46  8/26/2004 12:06:36.148 PM 3a
47  8/26/2004 12:06:36.148 PM 32
48  8/26/2004 12:06:36.148 PM 30
49  8/26/2004 12:06:36.148 PM 32
50  8/26/2004 12:06:36.154 PM 30
51  8/26/2004 12:06:36.154 PM 30
52  8/26/2004 12:06:36.154 PM 32
53  8/26/2004 12:06:36.154 PM 30
54  8/26/2004 12:06:36.154 PM 03
55  8/26/2004 12:06:37.177 PM 02
56  8/26/2004 12:06:37.177 PM 41
57  8/26/2004 12:06:37.177 PM 44
58  8/26/2004 12:06:37.177 PM 30
59  8/26/2004 12:06:37.177 PM 31
60  8/26/2004 12:06:37.183 PM 3b
```

An alternate file format is:

```
 1  DTE 00:00.000 02
 2  DTE 00:00.000 41
 3  DTE 00:00.000 44
 4  DTE 00:00.000 30
 5  DTE 00:00.000 31
 6  DTE 00:00.000 3b
 7  DTE 00:00.000 52
 8  DTE 00:00.000 4f
 9  DTE 00:00.000 4e
10  DTE 00:00.000 3a
11  DTE 00:00.000 30
12  DTE 00:00.000 03
13  DCE 00:00.001 06
14  DCE 00:00.001 02
15  DCE 00:00.001 41
16  DCE 00:00.001 44
17  DCE 00:00.001 30
```

```
18  DCE 00:00.001 31
19  DCE 00:00.001 3b
20  DCE 00:00.001 52
21  DCE 00:00.001 4f
22  DCE 00:00.001 4e
23  DCE 00:00.001 03
24  DTE 00:00.002 02
25  DTE 00:00.002 41
26  DTE 00:00.002 44
27  DTE 00:00.002 30
28  DTE 00:00.002 31
29  DTE 00:00.002 3b
30  DTE 00:00.002 47
31  DTE 00:00.002 43
32  DTE 00:00.002 37
33  DTE 00:00.002 3a
34  DTE 00:00.002 32
35  DTE 00:00.002 30
36  DTE 00:00.002 32
37  DTE 00:00.002 30
38  DTE 00:00.002 30
39  DTE 00:00.002 30
40  DTE 00:00.002 30
41  DTE 00:00.002 03
42  DCE 00:00.003 06
```

2.1.4 Internal Processing

This routine will make comment lines, by prepending a pound sign ("#") to each input line, out of the first three lines, stick a blank line out and delete line five.

Once the first few lines have been processed, all of the following lines will have their date and the hours part of the time fields deleted.

Month, day and hour fields may be one or two digits long. The year field **must** be four digits long.

With the alternate file format the first three characters may be "DCE␣" or "DTE␣".

2.1.5 Output

This routine identifies its self on the screen and in the output file. The output file has the program's name written as a comment line.

Typical results of this step of processing are:

```
1  06:06.434
2  06:35.060 02
3  06:35.060 41
4  06:35.060 44
5  06:35.060 30
6  06:35.060 31
```

```
 7  06:35.060 3b
 8  06:35.060 52
 9  06:35.066 4f
10  06:35.066 4e
11  06:35.066 3a
12  06:35.066 30
13  06:35.066 03
14  06:35.106 02
15  06:35.106 41
16  06:35.106 44
17  06:35.106 30
18  06:35.112 31
19  06:35.112 3b
20  06:35.112 47
21  06:35.112 43
22  06:35.112 37
23  06:35.112 3a
24  06:35.112 32
25  06:35.112 30
26  06:35.112 32
27  06:35.112 30
28  06:35.112 30
29  06:35.112 30
30  06:35.119 30
31  06:35.119 03
32  06:36.141 02
33  06:36.141 41
34  06:36.148 44
35  06:36.148 30
36  06:36.148 31
37  06:36.148 3b
38  06:36.148 47
39  06:36.148 43
40  06:36.148 37
41  06:36.148 3a
42  06:36.148 32
43  06:36.148 30
44  06:36.148 32
45  06:36.154 30
46  06:36.154 30
47  06:36.154 32
48  06:36.154 30
49  06:36.154 03
50  06:37.177 02
51  06:37.177 41
52  06:37.177 44
53  06:37.177 30
54  06:37.177 31
55  06:37.183 3b
```

2.1.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/FormatFx.l,v 1.17 2004-12-01 07:11:14-08 Hamilton Exp
Hamilton $
 3  // FIX1 is used to get rid of some unused data in an FTS comm line dump.
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <time.h>
 7
 8  #undef yywrap
 9  %}
10  %%
11  "#".*\n        ECHO;   // Print out comments starting with a # sign
12
13  ^"FTS".*\n   |
14  ^"Event".*\n |
15  ^\n                 {
16                       fprintf(yyout,"# %12s, %4d: %s",__FILE__,__LINE__,
17                                     yytext);
18                      }
19
20  ^"Timestamp".*\n     ;
21  ^"Sd".*\n            ;
22
23  ^"DCE "              ECHO;
24  ^"DTE "              ECHO;
25
26  ^[0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;
27  [0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;
28
29  [0-9]{1,2}":"[0-9]{2}"."[0-9]{3}" "{1,5}$ ;
30
31  " PM"                ;
32  " AM"                ;
33
34  .|\n     ECHO;
35  %%
36
37  // This rule:
38  // ^[0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;
39  // Means:
40  // Starting in the first column
41  // 1 or 2 numbers from 0 to 9
42  // followed by a /
43  // then 1 or 2 numbers
44  // followed by a /
45  // and 4 numbers
46  // with a blank following
47  // then 1 or two numbers from 0 to 9
48  // and a trailing colon/blank ": ".
49
50  // yywrap is for End Of File processing, 1 = done
```

```
51  int yywrap(void)
52  {
53          return 1;
54  }
55
56  main(int argc, char *argv[])
57  {
58  char infilename[100];
59  char outfilename[100];
60
61  time_t thistime;
62  struct tm *t;
63
64          fprintf(stderr,"$Id: FormatFx.l,v 1.17 2004-12-01 07:11:14-08 Hamilton Exp Hamilton
$\n");
65
66          strcpy(infilename,"stdin");
67          strcpy(outfilename,"stdout");
68
69          // Do we have to deal with stdin/stdout?
70          // 3 = one for the program name and one each for input and output names
71          if (argc == 3) {
72              // Nope
73              // Get over the program, etc., name
74              argc--;
75              argv++;
76              yyin = fopen(argv[0],"r");
77              if (yyin == NULL) {
78                  fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
79                                  argv[0]);
80                  exit(EXIT_FAILURE);
81              }
82              strcpy(infilename,argv[0]);
83
84              // Now get over the input file name and get to the output file name
85              argc--;
86              argv++;
87              yyout = fopen(argv[0],"w");
88              if (yyout == NULL) {
89                  fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
90                                  argv[0]);
91                  exit(EXIT_FAILURE);
92              }
93              strcpy(outfilename,argv[0]);
94          }
95          else {
96              // Woops we are using std... type io
97              yyin  = stdin;
98              yyout = stdout;
99          }
100
```

```
101         fprintf(yyout,"#\n# %12s, %4d: $Id: FormatFx.l,v 1.17 2004-12-01 07:11:14-08 Hamilton
 Exp Hamilton $\n",
102                         __FILE__,__LINE__);
103         t = time(NULL);
104         thistime = localtime(&t);
105         fprintf(yyout,"# %12s, %4d: %s",
106                         __FILE__,__LINE__,asctime(thistime));
107         fprintf(yyout,"# %12s, %4d: Reading from \"%s\"\n",
108                         __FILE__,__LINE__,infilename);
109         fprintf(yyout,"# %12s, %4d: Writing into \"%s\"\n#\n",
110                         __FILE__,__LINE__,outfilename);
111
112         yylex();
113         exit(EXIT_SUCCESS);
114 }
115
116 // $Log: FormatFx.l,v $
117 // Revision 1.17  2004-12-01 07:11:14-08  Hamilton
118 // Normal end of day data saving
119 //
120 // Revision 1.16  2004-11-30 15:07:13-08  Hamilton
121 // Normal end of day data saving
122 //
123 // Revision 1.15  2004-11-30 13:06:10-08  Hamilton
124 // Normal end of day data saving
125 //
126 // Revision 1.14  2004-11-29 15:41:23-08  Hamilton
127 // Normal end of day data saving
128 //
129 // Revision 1.13  2004-11-11 15:53:16-08  Hamilton
130 // Normal end of day data saving
131 //
132 // Revision 1.12  2004-11-11 08:45:50-08  Hamilton
133 // Normal end of day data saving
134 //
135 // Revision 1.11  2004-11-11 07:45:20-08  Hamilton
136 // Normal end of day data saving
137 //
138 // Revision 1.10  2004-11-11 07:38:03-08  Hamilton
139 // Normal end of day data saving
140 //
141 // Revision 1.9  2004-11-11 07:14:22-08  Hamilton
142 // Normal end of day data saving
143 //
144 // Revision 1.8  2004-11-04 12:17:49-08  Hamilton
145 // Normal end of day data saving
146 //
147
```

2.2 DB.c

2.2.1 Description

DB, Dump Breakout, is used to convert saved BREAKOUT files into usable ASCII files.

2.2.2 Calling Sequence

There are three runtime options to DB:

1. 2: This modifies the operation of DB by putting out a pair of line feed/carriage returns at each change of direction of data instead of the more usual signal pair.
2. A: Prints out an ASCII representation of each input character as shown in this example:

```
 1   O-02(.) O-41(A) O-44(D) O-30(0) O-31(1) O-3B(;) O-52(R) O-4F(O) O-4E(N) O-3A(:) O-30(0)
O-03(.)
 2   I-06(.) I-02(.) I-41(A) I-44(D) I-30(0) I-31(1) I-3B(;) I-52(R) I-4F(O) I-4E(N) I-03(.)
 3   O-02(.) O-41(A) O-44(D) O-30(0) O-31(1) O-3B(;) O-47(G) O-43(C) O-37(7) O-3A(:) O-32(2)
O-30(0) O-32(2) O-30(0) O-30(0) O-30(0) O-30(0) O-03(.)
 4   I-06(.)
 5   O-06(.)
```

3. R: Reverses the direction indicators. I.e. the "O-" indicators for "output" become "I-" indicators to show input data. When this is done the above example becomes:

```
 1   I-02(.) I-41(A) I-44(D) I-30(0) I-31(1) I-3B(;) I-52(R) I-4F(O) I-4E(N) I-3A(:) I-30(0)
I-03(.)
 2   O-06(.) O-02(.) O-41(A) O-44(D) O-30(0) O-31(1) O-3B(;) O-52(R) O-4F(O) O-4E(N) O-03(.)
 3   I-02(.) I-41(A) I-44(D) I-30(0) I-31(1) I-3B(;) I-47(G) I-43(C) I-37(7) I-3A(:) I-32(2)
I-30(0) I-32(2) I-30(0) I-30(0) I-30(0) I-30(0) I-03(.)
 4   O-06(.)
 5   I-06(.)
```

All options may be entered in upper or lower case values.

```
db infile outfile
```

2.2.3 Input Data Format

The file format of a breakout file is detailed in the BREAKOUT documentation. It is in binary and a partial capture file dump is included here:

```
1  000000 0F A5 66 20 53 61 76 65 64 20 46 72 69 20 4E 6F
2  000010 76 20 31 39 20 31 32 3A 35 36 3A 30 33 20 32 30
3  000020 30 34 00 20 00 C0 00 02 00 C0 00 41 00 C0 00 44
4  000030 00 C0 00 30 00 C0 00 31 00 C0 00 3B 00 C0 00 52
5  000040 00 C0 00 4F 00 C0 00 4E 00 C0 00 3A 00 C0 00 30
```

```
 6  000050 00 C0 00 03 00 C1 00 06 00 C1 00 02 00 C1 00 41
 7  000060 00 C1 00 44 00 C1 00 30 00 C1 00 31 00 C1 00 3B
 8  000070 00 C1 00 52 00 C1 00 4F 00 C1 00 4E 00 C1 00 03
 9  000080 00 C0 00 02 00 C0 00 41 00 C0 00 44 00 C0 00 30
10  000090 00 C0 00 31 00 C0 00 3B 00 C0 00 47 00 C0 00 43
11  0000A0 00 C0 00 37 00 C0 00 3A 00 C0 00 32 00 C0 00 30
12  0000B0 00 C0 00 32 00 C0 00 30 00 C0 00 30 00 C0 00 30
13  0000C0 00 C0 00 30 00 C0 00 03 00 C1 00 06 00 C0 00 06
```

2.2.4 Internal Processing

When called DB reads the command line for any runtime options, it then reads the BREAKOUT capture's file header and generates its own header: (Note that the RCS control line has been intentionally modified.)

```
#         db.c,  166: $ Id: db.c,v 1.18 2004-12-01 06:52:26-08 Hamilton Exp Hamilton $
#
#         db.c,  169: Dumping "DocSampl\19NOV04A.cap" a 88888 byte file.
#
#         db.c,  179: The .CAP file's header is:
#         db.c,  181: "Saved Fri Nov 19 12:56:03 2004 "
#
#         db.c,  190: Format of this output is as follows:
#         db.c,  191: There are two directions of data flow.
#         db.c,  192: Data direction is indicated with either
#         db.c,  193: a "1" or "0" followed by a "-".
#         db.c,  194: It is intended that "1" will
#         db.c,  195: be used to indicate data that is coming in,
#         db.c,  196: and that "0" is used for data
#         db.c,  197: going out. If this is not true, then use
#         db.c,  198: the /R option to reverse the logic
#         db.c,  199: of these indicators
#         db.c,  200: Following the "-" is the value of the
#         db.c,  201: data byte in hexidecimal.
#         db.c,  202: Whenever a direction change is detected
#         db.c,  203: a new line is output.
#         db.c,  205: Data source directions are reversed.
#         db.c,  208: Each direction-data item is preceeded
#         db.c,  209: by a single blank.
#
```

It then reads the capture file and translates it into ASCII. Translation consists of marking input data with "I-" and output data with "Ø-" indicators. In reality the directions are totally arbitrary and may be changed by using the "R" runtime flag. When a change from input to output, or visa versa, occurs a carriage return/line feed is output.

This processing continues until the end of data when a short summary of the work is output:

```
#
#         db.c,  227: There were 22213 data bytes converted
#         db.c,  229: There were 1412 lines output
#
```

2.2.5 Output

Typical results of this step of processing are:

```
1
2   O-02 O-41 O-44 O-30 O-31 O-3B O-52 O-4F O-4E O-3A O-30 O-03
3   I-06 I-02 I-41 I-44 I-30 I-31 I-3B I-52 I-4F I-4E I-03
4   O-02 O-41 O-44 O-30 O-31 O-3B O-47 O-43 O-37 O-3A O-32 O-30 O-32 O-30 O-30 O-30 O-30 O-03
5   I-06
```

2.2.6 Listing

```
 1  // $Header: d:/txb-p/Captures/RCS/DB.c,v 1.19 2004-12-01 07:11:13-08 Hamilton Exp Hamilton $
 2  // Copyright (c) by Pelco, 1998, 1999, 2001, 2004
 3
 4  // Dumps the contents of Breakout's .CAP files.
 5  //
 6  // Options are:
 7  //
 8  //      /2 = Put out two LF-CRs on change in direction of data.
 9  //      /A = Print out an ASCII equivelent of the data.
10  //      /R = Reverse the meaning of the direction indicators "0"- and "1-"
11  //
12  // 1DEC04 HAM Changed the first output character on comments from % to #.
13  //
14
15  #include <ctype.h>
16  #include <stdio.h>
17  #include <string.h>
18  #include <time.h>
19
20  #define FROM         0           // First data direction
21  #define TO           1           // Other data direction
22
23  #define NO           0           // No
24  #define YES          1           // Yes
25
26  void    DumpFrame(void);         // Interpertes a Breakout data frame
27  long    Flength(char *fname);    // Returns the length of a file
28  int     GetDirection(void);      // What is the current data direction
29  void    PrintHelp(void);         // Prints out a help file for bad argruments
30
31  // Defines for use with breakout's .CAP file
32  #define SIGNALS     0            // State of the RS-232 control lines
33  #define DTR         0x01         // Data Terminal Ready bit
34  #define RTS         0x02         // Request To Send bit
35  #define SB          0x04         // Set Break bit
36  #define BI          0x08         // Break Interrupt bit
37  #define CTS         0x10         // Clear To Send bit
38  #define DSR         0x20         // Data Set Ready bit
39  #define RI          0x40         // Ring Indicator bit
40  #define DCD         0x80         // Data Carrier Detect bit
41
42  #define SOURCE      1            // Direction/status of the data
43  #define DTE         0x01         // DTE data event bit
44  #define OER         0x02         // Overflow error bit
45  #define PER         0x04         // Parity error bit
46  #define FER         0x08         // Frame error bit
47  #define TSL         0x10         // Long Time Stamp Enabled bit
48  #define TSE         0x20         // Time Stamp Enabled bit
49  #define VE          0x40         // Valid Event bit
50  #define DP          0x80         // Data Present bit
51  #define DATA_OK     (VE|DP)      // Indicates that the data byte is valid
```

```
52  #define TIME_STAMP  (VE|TSE|TSL) // Might indicate a valid time stamp field
53
54  #define TS          2            // Time stamp
55  #define DATA        3            // The data for this frame
56
57  // For long time stamp records the following holds:
58  //
59  // SIGNALS Same as for data records
60  // SOURCE  Top 4 bits identify a time record, bottom 4 bits are the
61  //         4 MSBs of the time tag
62  // TS      Same as for data records, i.e. all 8 bits are the 8 LSBs of the
63  //         time tag
64  // DATA    Middle 8 bits of the time tag
65  //
66  // Thus we have:
67  //
68  // TimeTag = SOURCE[3,2,1,0] + DATA[7,6,5,4,3,2,1,0] + TS[7,6,5,4,3,2,1,0]
69  //           [bits involved]
70
71  long          ByteCounter = 0;          // Number of data bytes dumped
72  unsigned int  DataFrame[4];             // Contents of a Breakout data frame
73  long          FileLength;               // Input file length, in bytes
74  long          InputByte = 0;            // Current data frame being worked on
75  char         *InputPointer = NULL;      // Input file name from the cmnd line
76  FILE         *InFile;                   // Pointer to the input file
77  long          LineCounter = 23;         // How many lines have been output
78                                          // 23 = 18 lines of header and
79                                          //       5 lines of trailer
80  int           MsgTime;                  // This message's time
81  int           OkToPrint = YES;          // YES or NO
82  int           PrintASCII = NO;          // Print ASCII data equivelents
83  int           ReverseDirection = NO;    // Swap input flags with output flags
84  int           TwoLineFeeds = NO;        // 1 or 2 LFs on direction change
85
86
87  int main(int argc, char *argv[])
88  {
89  char   *ptr2 = NULL;
90  char    temp[100];   // Dont expect an argument to be larger than 100 chars
91  long    i;
92  int     data;
93
94      fprintf(stderr,"\n$Header: d:/txb-p/Captures/RCS/DB.c,v 1.19 2004-12-01 07:11:13-08
 Hamilton Exp Hamilton $\n");
95
96      // Process the args
97      while (--argc) {
98          strcpy(temp, *(++argv));
99          strupr(temp);
100
101         switch (temp[0]) {
102             // For Unix comment out the following line "case '/':"
```

```
103             // to enable root directory access, then use '-' for all switches
104             case '/':
105             case '-':
106                 if (temp[2] != NULL) {
107                     fprintf(stderr,"\a%12s, %4d: Invalid Switch => \"%s\"\n",
108                                     __FILE__,__LINE__,*(argv));
109                     PrintHelp();
110                     return 0;
111                 }
112                 else {
113                     switch (temp[1]) {
114                         case '2':
115                             TwoLineFeeds = YES;
116                             break;
117                         case 'A':
118                             PrintASCII = YES;
119                             break;
120                         case 'R':
121                             ReverseDirection = YES;// Reverse direction
122                             break;
123                         case '?':
124                             PrintHelp();
125                             return 0;
126                         default:
127                             fprintf(stderr,"\a%12s, %4d: Invalid Switch => \"%s\"\n",
128                                             __FILE__,__LINE__,*(argv));
129                             PrintHelp();
130                             return 0;
131                     }
132                 }
133                 break;
134
135             default:
136                 if (InputPointer == NULL)    // 1st file will be the input file (InFile)
137                     FileLength = Flength(InputPointer = *(argv));
138                 else if (ptr2 == NULL)       // 2nd file will be the output file to get stdout
139                     ptr2 = *(argv);
140                 else {                       // 3rd file is too many, quit
141                     fprintf(stderr,"\a%12s, %4d: Too many arguments, error with \"%s\"\n",
142                                     __FILE__,__LINE__,*(argv));
143                     PrintHelp();
144                     return 0;
145                  }
146         } //end switch
147     } // end while
148
149     if (ptr2)
150         if (freopen(ptr2, "w", stdout) == NULL) {
151             fprintf(stderr, "%12s, %4d: Error redirecting stdout to \"%s\"\n",
152                             __FILE__,__LINE__,ptr2);
153             return 0;
154         }
```

```
155
156     if (-1L == FileLength) {
157         fprintf(stdout,"# %12s, %4d: Unable to dump \"%s\"\n",
158                         __FILE__,__LINE__,InputPointer);
159         fprintf(stderr,"\a%12s, %4d: Unable to dump \"%s\"\n",
160                         __FILE__,__LINE__,InputPointer);
161         fclose(stdout);
162     }
163
164     fprintf(stderr,"%12s, %4d: Dumping \"%s\" a %ld byte file.\n",
165                     __FILE__,__LINE__,InputPointer, FileLength);
166     fprintf(stdout,"# %12s, %4d: $Id: DB.c,v 1.19 2004-12-01 07:11:13-08 Hamilton Exp Hamilton
 $\n",__FILE__,__LINE__);
167     fprintf(stdout,"#\n");
168     fprintf(stdout,"# %12s, %4d: Dumping \"%s\" a %ld byte file.\n",
169                     __FILE__,__LINE__,InputPointer, FileLength);
170     InFile = fopen(InputPointer,"rb");
171
172     // First check for Breakout's signature
173     if ((InFile != NULL) &&
174         (fgetc(InFile) == 0x0F) &&
175         (fgetc(InFile) == 0xA5) &&
176         (fgetc(InFile) == 0x66) &&
177         (fgetc(InFile) == 0x20)) {
178             fprintf(stdout,"#\n");
179             fprintf(stdout,"# %12s, %4d: The .CAP file's header is:\n",__FILE__,__LINE__);
180             // Then printout the breakout header
181             fprintf(stdout,"# %12s, %4d: \"",__FILE__,__LINE__);
182             for (i = 4; i < 36; i++) {
183                 data = fgetc(InFile);
184                 if (data == 0x13) // If there is a CR then start a new line
185                     fprintf(stdout,"# %12s, %4d: ",__FILE__,__LINE__);
186                 fprintf(stdout,"%c",data);
187             }
188             fprintf(stdout,"\"\n");
189             fprintf(stdout,"#\n");
190         fprintf(stdout,"# %12s, %4d: Format of this output is as
 follows:\n",__FILE__,__LINE__);
191         fprintf(stdout,"# %12s, %4d: There are two directions of data
 flow.\n",__FILE__,__LINE__);
192         fprintf(stdout,"# %12s, %4d: Data direction is indicated with
 either\n",__FILE__,__LINE__);
193         fprintf(stdout,"# %12s, %4d: a \"1\" or \"0\" followed by a
 \"-\".\n",__FILE__,__LINE__);
194         fprintf(stdout,"# %12s, %4d: It is intended that \"1\" will\n",__FILE__,__LINE__);
195         fprintf(stdout,"# %12s, %4d: be used to indicate data that is coming
 in,\n",__FILE__,__LINE__);
196         fprintf(stdout,"# %12s, %4d: and that \"0\" is used for data\n",__FILE__,__LINE__);
197         fprintf(stdout,"# %12s, %4d: going out. If this is not true, then
 use\n",__FILE__,__LINE__);
198         fprintf(stdout,"# %12s, %4d: the /R option to reverse the logic\n",__FILE__,__LINE__);
199         fprintf(stdout,"# %12s, %4d: of these indicators\n",__FILE__,__LINE__);
```

```
200          fprintf(stdout,"# %12s, %4d: Following the \"-\" is the value of
 the\n",__FILE__,__LINE__);
201          fprintf(stdout,"# %12s, %4d: data byte in hexidecimal.\n",__FILE__,__LINE__);
202          fprintf(stdout,"# %12s, %4d: Whenever a direction change is
 detected\n",__FILE__,__LINE__);
203          fprintf(stdout,"# %12s, %4d: a new line is output.\n",__FILE__,__LINE__);
204          if (ReverseDirection == YES)
205              fprintf(stdout,"# %12s, %4d: Data source directions are
 reversed.\n",__FILE__,__LINE__);
206          else
207              fprintf(stdout,"# %12s, %4d: Data source directions are
 normal.\n",__FILE__,__LINE__);
208          fprintf(stdout,"# %12s, %4d: Each direction-data item is
 preceeded\n",__FILE__,__LINE__);
209          fprintf(stdout,"# %12s, %4d: by a single blank.\n",__FILE__,__LINE__);
210          fprintf(stdout,"#\n");
211
212          // Dump the frames
213          for (InputByte = 0; InputByte < FileLength; InputByte += 4) {
214              DataFrame[SIGNALS] = fgetc(InFile);
215              DataFrame[SOURCE]  = fgetc(InFile);
216              DataFrame[TS]      = fgetc(InFile);
217              DataFrame[DATA]    = fgetc(InFile);
218              DumpFrame();
219          }
220          fprintf(stderr,"%12s, %4d: There were %ld data bytes converted\n",
221                          __FILE__,__LINE__,ByteCounter);
222          fprintf(stderr,"%12s, %4d: There were %ld lines output\n",
223                          __FILE__,__LINE__,LineCounter);
224
225          fprintf(stdout,"\n#\n");
226          fprintf(stdout,"# %12s, %4d: There were %ld data bytes converted\n",
227                          __FILE__,__LINE__,ByteCounter);
228          fprintf(stdout,"# %12s, %4d: There were %ld lines output\n",
229                          __FILE__,__LINE__,LineCounter);
230          fprintf(stdout,"#\n");
231      }
232      else {
233              fprintf(stdout,"\n\n# %12s, %4d: File \"%s\" is not a Breakout capture file\n",
234                              __FILE__,__LINE__,InputPointer);
235              fprintf(stderr,"%12s, %4d: File \"%s\" is not a Breakout capture file\n",
236                              __FILE__,__LINE__,InputPointer);
237      }
238      fclose(stdout);
239      return 0;
240  }
241
242
243  // DumpFrame is used to dump the contents of one Breakout frame of data.
244  // It is assumed that the array DataFrame has been filled up with data
245  // from the .CAP file in the following manner:
246  //
```

```
247 //                DataFrame[SIGNALS] = Status of most of the control lines
248 //                DataFrame[SOURCE]  = Flags as to where it came from
249 //                DataFrame[TS]      = The current time stamp
250 //                DataFrame[DATA]    = The most recent data byte
251 //
252 void DumpFrame(void)
253 {
254 static int lastwas = TO;
255 long   timetag;
256
257     // Is this a time stamped entry? Then dump the time stamp data.
258     if ((DataFrame[SOURCE] & TIME_STAMP) == TIME_STAMP) {
259             timetag = DataFrame[SOURCE] & 0x0F;
260             timetag = (timetag << 8) + DataFrame[DATA];
261             timetag = (timetag << 8) + DataFrame[TS];
262     }
263     else {
264         if ((OkToPrint == YES) &&
265             (DataFrame[DATA] != 0xFFFFFFFF)) {
266
267             if (GetDirection() == TO) {
268                 if (lastwas == FROM) {
269                     fprintf(stdout,"\n");
270                     LineCounter++;
271                     if (TwoLineFeeds == YES) {
272                         fprintf(stdout,"\n");
273                         LineCounter++;
274                     }
275                 }
276                 lastwas = TO;
277                 fprintf(stdout," I-");
278             }
279             else {
280                 if (lastwas == TO) {
281                     fprintf(stdout,"\n");
282                     LineCounter++;
283                     if (TwoLineFeeds == YES) {
284                         fprintf(stdout,"\n");
285                         LineCounter++;
286                     }
287                 }
288                 lastwas = FROM;
289                 fprintf(stdout," O-");
290             }
291             fprintf(stdout,"%02X",DataFrame[DATA]);
292             if (PrintASCII == YES) {
293                 if (isprint(DataFrame[DATA]))
294                     fprintf(stdout,"(%c)",DataFrame[DATA]);
295                 else
296                     fprintf(stdout,"(.)");
297             }
298         }
```

```
299      } // !if ((DataFrame[SOURCE] & TIME_STAMP) == TIME_STAMP)
300      return;
301  }
302
303
304  // Calculate the length of the file.
305  long Flength(char *fname)
306  {
307  long length = -1L;
308  FILE *fptr;
309
310     fptr = fopen(fname, "rb");
311
312     if (fptr != NULL) {
313         fseek(fptr, 0L, SEEK_END);
314         length = ftell(fptr);
315         fclose(fptr);
316     }
317     return length;
318  }
319
320
321  int  GetDirection(void)
322  {
323      ByteCounter++;
324      if (ReverseDirection == YES) {
325          if (DataFrame[SOURCE] & DTE)
326                  return(TO);
327          else
328                  return(FROM);
329      }
330      else {
331          if (DataFrame[SOURCE] & DTE)
332                  return(FROM);
333          else
334                  return(TO);
335      }
336  }
337
338  void PrintHelp(void)
339  {
340      fprintf(stderr,"\n%12s, %4d: 2 = Two line feeds on data direction
 change\n",__FILE__,__LINE__);
341      fprintf(stderr,"%12s, %4d: A = Print ASCII equivalents of the data in
 ()s\n",__FILE__,__LINE__);
342      fprintf(stderr,"%12s, %4d: R = Reverse direction of data-source
 flags\n\n",__FILE__,__LINE__);
343      return;
344  }
345
```

2.3 NC.l

2.3.1 Description

NC is used to strip comments out of a file.

2.3.2 Calling Sequence

```
nc infile outfile
```

2.3.3 Input Data Format

NC is used to delete everything from the comment identifier, "#", to the end of the line.

```
 1  #
 2  #     GcProc.L, 1007: $Id: GcProc.l,v 1.8 2004-12-01 07:11:14-08 Hamilton Exp Hamilton $
 3  #     GcProc.L, 1011: Wed Dec  1 11:32:22 2004
 4  #     GcProc.L, 1013: Reading from "test4.out"
 5  #     GcProc.L, 1015: Writing into "test3.out"
 6  #     GcProc.L, 1016:
 7  #     GcProc.L, 1018:  ' --->' = DTE, from keyboard
 8  #     GcProc.L, 1020:  '<--- ' = DCE, from camera
 9  #     GcProc.L, 1021:
10  #
11  #    UnBlank.L,   84: $Id: UnBlank.l,v 1.12 2004-12-01 07:11:15-08 Hamilton Exp Hamilton $
12  #    UnBlank.L,   89: Wed Dec  1 11:32:22 2004
13  #    UnBlank.L,   91: Reading from "test2.out"
14  #    UnBlank.L,   93: Writing into "test3.out"
15  #
16  #   PCmdMake.L,  203: $Id: PCmdMake.l,v 1.16 2004-12-01 07:11:14-08 Hamilton Exp Hamilton $
17  #   PCmdMake.L,  205: Reading from "test1.out"
18  #   PCmdMake.L,  207: Writing into "test2.out"
19  #   PCmdMake.L,  212: Wed Dec  1 11:32:22 2004
20  #
21  #   FormatFx.L,  102: $Id: FormatFx.l,v 1.17 2004-12-01 07:11:14-08 Hamilton Exp Hamilton $
22  #   FormatFx.L,  106: Wed Dec  1 11:32:22 2004
23  #   FormatFx.L,  108: Reading from "test0.out"
24  #   FormatFx.L,  110: Writing into "test1.out"
25  #
26  #   FormatFx.L,   16: FTS capture file: E:\Capture\newtest.cfa (8/26/2004 12:34:27 PM)
27  #   FormatFx.L,   16: Event 1 (8/26/2004 12:06:06.434 PM) through
28  #   FormatFx.L,   16: Event 30,910 (8/26/2004 12:32:44.595 PM)
29  #   FormatFx.L,   16:
30  06:35.060 RON 0 <newcc> Suppress: None
31  06:35.106 2020000 <rsro><rfco>  Unknown command
32  06:36.141 2020020 <rsro><rfco>  Unknown command
```

2.3.4 Internal Processing

When a comment identifier, in any column, is detected, it and everything to the end of the line is deleted.

2.3.5 Output

This will be identical to the input file without any inserted comments.

```
1  06:35.060 RON 0 <newcc> Suppress: None
2  06:35.106 2020000 <rsro><rfco>  Unknown command
3  06:36.141 2020020 <rsro><rfco>  Unknown command
```

2.3.6 Listing

```
%{
// $Header: d:/txb-p/Captures/RCS/NC.l,v 1.2 2004-12-01 07:11:15-08 Hamilton Exp Hamilton $
// LMAIN is used as a "universal" starter skeleton for FLEX programs
#include <stdio.h>
#include <stdlib.h>
#undef yywrap
%}
%%
"#".*\n    ; // strip all comment lines starting with a #

.|\n        ECHO; // keep the rest
%%
// yywrap is for End Of File processing, 1 = done
int yywrap(void)
{
        return 1;
}

main(int argc, char *argv[])
{
        fprintf(stderr,"$Id: NC.l,v 1.2 2004-12-01 07:11:15-08 Hamilton Exp Hamilton $\n");

        // Do we have to deal with stdin/stdout?
        // 3 = one for the program name and one each for input and output names
        if (argc == 3) {
            // Nope
            // Get over the program, etc., name
            argc--;
            argv++;
            yyin = fopen(argv[0],"r");
            if (yyin == NULL) {
                fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
                                argv[0]);
                exit(EXIT_FAILURE);
            }

            // Now get over the input file name and get to the output file name
            argc--;
            argv++;
            yyout = fopen(argv[0],"w");
            if (yyout == NULL) {
                fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
                                argv[0]);
                exit(EXIT_FAILURE);
            }
        }
        else {
            // Woops we are using std... type io
            yyin  = stdin;
            yyout = stdout;
        }
```

```
52
53          yylex();
54          exit(EXIT_SUCCESS);
55  }
56
57  // $Log: NC.l,v $
58  // Revision 1.2  2004-12-01 07:11:15-08  Hamilton
59  // Normal end of day data saving
60  //
61  // Revision 1.1  2004-11-30 14:35:26-08  Hamilton
62  // Initial revision
63  //
64  // Revision 1.6  1999-07-09 07:54:17-07  Hamilton
65  // Deleted references to NOGETOPT, as I am not using it with the new vesion
66  // of FLEX.
67  //
68  // Revision 1.5  1997-01-12 21:12:50-08  ham
69  // Added in define for NO_GETOPT.  When set this will inhibit the inclusion
70  // of GNU's getopt routines, kinda.
71  //
72  // Revision 1.4  1996-11-14 19:36:28-0800  ham
73  // Add in end of file processing by adding the yywrap() function.
74  //
75
```

3 Panasonic Specific Routines

3.1 PCmdMake.l

3.1.1 Description

This routine is used to condense the Panasonic command format "one byte per line" format that FORMATFX.L generates into something more useful.

3.1.2 Calling Sequence

```
PCmdMake infile outfile
```

3.1.3 Input Data Format

This routine is designed to take the output from FORMATFX.L. This output is as follows:

```
 1  06:06.434
 2  06:35.060 02
 3  06:35.060 41
 4  06:35.060 44
 5  06:35.060 30
 6  06:35.060 31
 7  06:35.060 3b
 8  06:35.060 52
 9  06:35.066 4f
10  06:35.066 4e
11  06:35.066 3a
12  06:35.066 30
13  06:35.066 03
14  06:35.106 02
15  06:35.106 41
16  06:35.106 44
17  06:35.106 30
18  06:35.112 31
19  06:35.112 3b
20  06:35.112 47
21  06:35.112 43
22  06:35.112 37
23  06:35.112 3a
24  06:35.112 32
25  06:35.112 30
26  06:35.112 32
27  06:35.112 30
28  06:35.112 30
29  06:35.112 30
30  06:35.119 30
31  06:35.119 03
32  06:36.141 02
33  06:36.141 41
34  06:36.148 44
```

```
35  06:36.148 30
36  06:36.148 31
37  06:36.148 3b
38  06:36.148 47
39  06:36.148 43
40  06:36.148 37
41  06:36.148 3a
42  06:36.148 32
43  06:36.148 30
44  06:36.148 32
45  06:36.154 30
46  06:36.154 30
47  06:36.154 32
48  06:36.154 30
49  06:36.154 03
50  06:37.177 02
51  06:37.177 41
52  06:37.177 44
53  06:37.177 30
54  06:37.177 31
55  06:37.183 3b
```

3.1.4 Internal Processing

Each line is scanned, the routine looks for an ASCII `STX` (`0x02`) character. When the `STX` is found the current time tag is saved "`STX`" is output. All following time characters are ignored until the next data byte is found. If the next data byte is not an ASCII `ETX` (`0x03`) byte, then the data byte is output as an ASCII character with a leading blank as a delimiter, i.e. `0x33` is output as a "␣3". When an `ETX` is found "`ETX`" is output and a new line is started.

3.1.5 Output

This routine identifies its self on the screen and in the output file. The output file has the program's name written as a comment line.

Typical results of this step of processing are:

```
 1  06:35.060  STX A D 0 1 ; R O N : 0 ETX
 2  06:35.106  STX A D 0 1 ; G C 7 : 2 0 2 0 0 0 0 ETX
 3  06:36.141  STX A D 0 1 ; G C 7 : 2 0 2 0 0 2 0 ETX
 4  06:37.177  STX A D 0 1 ;
```

3.1.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/PCmdMake.l,v 1.16 2004-12-01 07:11:14-08 Hamilton Exp
Hamilton $
 3  // FIX2 is used as one step in reformatting Panasonic protocol data
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <time.h>
 7
 8  #undef yywrap
 9
10  char timetag[100];
11
12  int makenumb(void);
13  void makeascii(void);
14
15  %}
16  %x INMSG IN OUT OUTMSG EITHER EITHERMSG
17  %%
18  "#".*\n        ECHO;   // Print out comments starting with a # sign
19
20
21  ^"DCE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " {strcpy(timetag,yytext);
22                                                 BEGIN IN;
23                                                }
24
25  <IN>[0-9a-f]{2}     {// Look for an STX or ACK data byte
26                       if (yytext[0] == '0' && yytext[1] == '2') {
27                           fprintf(yyout,"%sSTX  ",timetag);
28                           BEGIN INMSG;
29                        } else
30                        if (yytext[0] == '0' && yytext[1] == '6') {
31                            fprintf(yyout,"%sACK\n",timetag);
32                            BEGIN 0;
33                        }
34                      }
35
36  <INMSG>[0-9a-f]{2} {// While in the message look for an ETX byte
37                      if (yytext[0] == '0' && yytext[1] == '3') {
38                          fprintf(yyout,"ETX\n");
39                          BEGIN 0;
40                      }
41                      else {// Not an ETX, make it printable
42                          makeascii();
43                      }
44                     }
45
46  <INMSG>^"DCE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " ;
47  <INMSG>.|\n        ;
48
49
50
```

```
51  ^"DTE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " {strcpy(timetag,yytext);
52                                                 BEGIN OUT;
53                                                }
54
55  <OUT>[0-9a-f]{2}     {// Look for an STX data byte
56                        if (yytext[0] == '0' && yytext[1] == '2') {
57                            fprintf(yyout,"%s STX ",timetag);
58                            BEGIN OUTMSG;
59                        } else
60                        if (yytext[0] == '0' && yytext[1] == '6') {
61                            fprintf(yyout,"%s ACK\n",timetag);
62                            BEGIN 0;
63                        }
64                       }
65
66  <OUTMSG>[0-9a-f]{2} {// While in the message look for an ETX byte
67                       if (yytext[0] == '0' && yytext[1] == '3') {
68                           fprintf(yyout,"ETX\n");
69                           BEGIN 0;
70                       }
71                       else {// Not an ETX, make it printable
72                           makeascii();
73                       }
74                      }
75
76  <OUTMSG>^"DTE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " ;
77  <OUTMSG>.|\n        ;
78
79
80
81  ^[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}"  "   ;
82  ^[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}"   "  ;
83
84  ^[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " {strcpy(timetag,yytext);
85                                          BEGIN EITHER;
86                                         }
87
88  <EITHER>[0-9a-f]{2}     {// Look for an STX data byte
89                           if (yytext[0] == '0' && yytext[1] == '2') {
90                               fprintf(yyout,"%s STX ",timetag);
91                               BEGIN EITHERMSG;
92                            } else
93                            if (yytext[0] == '0' && yytext[1] == '6') {
94                                fprintf(yyout,"%s ACK\n",timetag);
95                                BEGIN 0;
96                            }
97                           }
98
99  <EITHERMSG>[0-9a-f]{2} {// While in the message look for an ETX byte
100                          if (yytext[0] == '0' && yytext[1] == '3') {
101                              fprintf(yyout,"ETX\n");
102                              BEGIN 0;
```

```
103                          }
104                          else {// Not an ETX, make it printable
105                              makeascii();
106                          }
107                         }
108
109  <EITHERMSG>^[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " ;
110  <EITHERMSG>.|\n         ;
111
112
113  .|\n                ;
114
115  %%
116
117  // makeascii translates what is in yytext into printable ASCII.
118  void makeascii(void)
119  {
120  int result;
121
122      result = makenumb();
123           if (result == 0x02) { fprintf(yyout,"STX ");    }
124      else if (result == 0x03) { fprintf(yyout,"ETX\n");   }
125      else if (result == 0x06) { fprintf(yyout,"ACK\n");   }
126      else if (result == 0x15) { fprintf(yyout,"NAK \n");  }
127      else if (result <  0x20 ||
128               result >  0x7E) { fprintf(yyout,"--%d--",result);
129      }
130      else {
131          fprintf(yyout,"%c ",result);
132      }
133
134      return;
135  }
136
137  // makenumb is used to convert an input ASCII hex number to binary.
138  //
139  // yytext must contain the two digit ASCII hex value for conversation.
140  //
141  int makenumb(void)
142  {
143  char *pointer,string[4];
144  int number;
145
146      strcpy(string,yytext);
147      string[2] = 0;
148      number = strtol(string,&pointer,16);
149      return number;
150  }
151
152  // yywrap is for End Of File processing, 1 = done
153  int yywrap(void)
154  {
```

```
155         return 1;
156 }
157
158 main(int argc, char *argv[])
159 {
160 char infilename[100];
161 char outfilename[100];
162
163 time_t thistime;
164 struct tm *t;
165
166         fprintf(stderr,"$Id: PCmdMake.l,v 1.16 2004-12-01 07:11:14-08 Hamilton Exp Hamilton
 $\n");
167
168         strcpy(infilename,"stdin");
169         strcpy(outfilename,"stdout");
170
171         // Do we have to deal with stdin/stdout?
172         // 3 = one for the program name and one each for input and output names
173         if (argc == 3) {
174             // Nope
175             // Get over the program, etc., name
176             argc--;
177             argv++;
178             yyin = fopen(argv[0],"r");
179             if (yyin == NULL) {
180                 fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
181                                 argv[0]);
182                 exit(EXIT_FAILURE);
183             }
184             strcpy(infilename,argv[0]);
185
186             // Now get over the input file name and get to the output file name
187             argc--;
188             argv++;
189             yyout = fopen(argv[0],"w");
190             if (yyout == NULL) {
191                 fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
192                                 argv[0]);
193                 exit(EXIT_FAILURE);
194             }
195             strcpy(outfilename,argv[0]);
196         }
197         else {
198             // Woops we are using std... type io
199             yyin  = stdin;
200             yyout = stdout;
201         }
202
203         fprintf(yyout,"#\n# %12s, %4d: $Id: PCmdMake.l,v 1.16 2004-12-01 07:11:14-08 Hamilton
 Exp Hamilton $\n",__FILE__,__LINE__);
204         fprintf(yyout,"# %12s, %4d: Reading from \"%s\"\n",
```

```
205                         __FILE__,__LINE__,infilename);
206         fprintf(yyout,"# %12s, %4d: Writing into \"%s\"\n",
207                         __FILE__,__LINE__,outfilename);
208
209         t = time(NULL);
210         thistime = localtime(&t);
211         fprintf(yyout,"# %12s, %4d: %s",
212                         __FILE__,__LINE__,asctime(thistime));
213
214         yylex();
215         exit(EXIT_SUCCESS);
216 }
217
218 // $Log: PCmdMake.l,v $
219 // Revision 1.16  2004-12-01 07:11:14-08  Hamilton
220 // Normal end of day data saving
221 //
222 // Revision 1.15  2004-11-30 15:23:17-08  Hamilton
223 // Normal end of day data saving
224 //
225 // Revision 1.14  2004-11-30 15:07:13-08  Hamilton
226 // Normal end of day data saving
227 //
228 // Revision 1.13  2004-11-30 13:06:10-08  Hamilton
229 // Normal end of day data saving
230 //
231 // Revision 1.12  2004-11-15 15:31:39-08  Hamilton
232 // Normal end of day data saving
233 //
234 // Revision 1.11  2004-11-11 13:27:22-08  Hamilton
235 // Normal end of day data saving
236 //
237 // Revision 1.10  2004-11-11 08:45:51-08  Hamilton
238 // Normal end of day data saving
239 //
240 // Revision 1.9  2004-11-11 07:45:21-08  Hamilton
241 // Normal end of day data saving
242 //
243 // Revision 1.8  2004-11-11 07:24:17-08  Hamilton
244 // Normal end of day data saving
245 //
246 // Revision 1.7  2004-11-11 07:14:22-08  Hamilton
247 // Normal end of day data saving
248 //
249 // Revision 1.6  2004-11-04 10:47:03-08  Hamilton
250 // Normal end of day data saving
251 //
252
```

3.2 UnBlank.l

3.2.1 Description

This routine is a filter that is designed to pass only the characters that are actually wanted in the output. It passes the following character/character strings only, all others are disposed of. (Except for the contents of comment lines.):

"ACK␣"
"␣ETX"
"NAK␣"
"STX␣"
":␣"
";␣"
"␣"

`[passedthrough]`

Processing occurs in the following order so that double blanks get translated into single blanks and all other blanks are deleted. `[changed]`

1. All double blanks ("␣␣") are turned into single blanks ("␣") so as to bypass the general blank dumping and remain as single blanks.
2. Single blanks ("␣") are dumped, except for those in comments.

3.2.2 Calling Sequence

```
UnBlank infile outfile
```

3.2.3 Input Data Format

This routine is designed to take the output from PCmdMake.l. This output is as follows:

```
1  06:35.060  STX A D 0 1 ; R O N : 0 ETX
2  06:35.106  STX A D 0 1 ; G C 7 : 2 0 2 0 0 0 0 ETX
3  06:36.141  STX A D 0 1 ; G C 7 : 2 0 2 0 0 2 0 ETX
4  06:37.177  STX A D 0 1 ;
```

3.2.4 Internal Processing

The input file is scanned for items of interest. They are transformed (Section 3.2.1, page 36) or passed through (Section 3.2.1, page 36). Other items are dumped.

3.2.5 Output

This routine identifies its self on the screen and in the output file. The output file has the program's name written as a comment line.

Typical results of this step of processing are:

```
1  06:35.060  STX AD01; RON: 0 ETX
2  06:35.106  STX AD01; GC7: 2020000 ETX
3  06:36.141  STX AD01; GC7: 2020020 ETX
4  06:37.177  STX AD01;
```

3.2.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/UnBlank.l,v 1.12 2004-12-01 07:11:15-08 Hamilton Exp Hamilton
$
 3  // UNBLANK is used to remove unneeded blanks and pass through good stuff only.
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <time.h>
 7
 8  #undef yywrap
 9  %}
10  %%
11  "#".*\n        ECHO;   // Print out comments starting with a # sign
12
13  "  ACK"   |
14  " ACK"    |
15  "  ACK "        ECHO;
16  " ETX"          ECHO;
17  " NAK"    |
18  "NAK "          ECHO;
19  "  STX "  |
20  " STX  "        ECHO;
21
22  "; "            ECHO;
23  ": "            ECHO;
24  ^"DCE "         ECHO;
25  ^"DTE "         ECHO;
26
27  "   " |
28  "  "            fprintf(yyout," ");
29  " "             ;
30
31  .|\n            ECHO;
32  %%
33  // yywrap is for End Of File processing, 1 = done
34  int yywrap(void)
35  {
36          return 1;
37  }
38
39  main(int argc, char *argv[])
40  {
41  char infilename[100];
42  char outfilename[100];
43
44  time_t thistime;
45  struct tm *t;
46
47          fprintf(stderr,"$Id: UnBlank.l,v 1.12 2004-12-01 07:11:15-08 Hamilton Exp Hamilton
$\n");
48
49          strcpy(infilename,"stdin");
```

```
50          strcpy(outfilename,"stdout");
51
52          // Do we have to deal with stdin/stdout?
53          // 3 = one for the program name and one each for input and output names
54          if (argc == 3) {
55              // Nope
56              // Get over the program, etc., name
57              argc--;
58              argv++;
59              yyin = fopen(argv[0],"r");
60              if (yyin == NULL) {
61                  fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
62                                  argv[0]);
63                  exit(EXIT_FAILURE);
64              }
65              strcpy(infilename,argv[0]);
66
67              // Now get over the input file name and get to the output file name
68              argc--;
69              argv++;
70              yyout = fopen(argv[0],"w");
71              if (yyout == NULL) {
72                  fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
73                                  argv[0]);
74                  exit(EXIT_FAILURE);
75              }
76              strcpy(outfilename,argv[0]);
77          }
78          else {
79              // Woops we are using std... type io
80              yyin  = stdin;
81              yyout = stdout;
82          }
83
84          fprintf(yyout,"#\n# %12s, %4d: $Id: UnBlank.l,v 1.12 2004-12-01 07:11:15-08 Hamilton
Exp Hamilton $\n",__FILE__,__LINE__);
85
86          t = time(NULL);
87          thistime = localtime(&t);
88          fprintf(yyout,"# %12s, %4d: %s",
89                          __FILE__,__LINE__,asctime(thistime));
90          fprintf(yyout,"# %12s, %4d: Reading from \"%s\"\n",
91                          __FILE__,__LINE__,infilename);
92          fprintf(yyout,"# %12s, %4d: Writing into \"%s\"\n",
93                          __FILE__,__LINE__,outfilename);
94
95          yylex();
96          exit(EXIT_SUCCESS);
97  }
98
99  // $Log: UnBlank.l,v $
100 // Revision 1.12  2004-12-01 07:11:15-08  Hamilton
```

```
101  // Normal end of day data saving
102  //
103  // Revision 1.11  2004-11-30 15:07:15-08  Hamilton
104  // Normal end of day data saving
105  //
106  // Revision 1.10  2004-11-30 13:06:11-08  Hamilton
107  // Normal end of day data saving
108  //
109  // Revision 1.9  2004-11-11 15:53:18-08  Hamilton
110  // Normal end of day data saving
111  //
112  // Revision 1.8  2004-11-11 08:45:52-08  Hamilton
113  // Normal end of day data saving
114  //
115  // Revision 1.7  2004-11-11 07:45:22-08  Hamilton
116  // Normal end of day data saving
117  //
118  // Revision 1.6  2004-11-11 07:24:19-08  Hamilton
119  // Normal end of day data saving
120  //
121  // Revision 1.5  2004-11-04 10:47:05-08  Hamilton
122  // Normal end of day data saving
123  //
124
```

3.3 GcProc.l

3.3.1 Description

This routine is a filter that translates Panasonic protocol commands into short english statements. It is assumed that the file has been processed by many of the other programs in this series. The following "batch" file is a typical example of its normal calling sequence:

```
Formatfx <infile | PCmdMake | unblank | GcProc >outfile
```

3.3.2 Calling Sequence

```
GcProc infile outfile
```

3.3.3 Input Data Format

This routine is designed to take the output from UnBlank.l. This output is as follows:

```
1  06:35.060  STX AD01; RON: 0 ETX
2  06:35.106  STX AD01; GC7: 2020000 ETX
3  06:36.141  STX AD01; GC7: 2020020 ETX
4  06:37.177  STX AD01;
```

3.3.4 Internal Processing

Internally this routine matches up different translation strings with input strings and outputs them.

3.3.5 Output

This routine identifies its self on the screen and in the output file. The output file has the program's name written as a comment line.

Typical results of this step of processing are:

```
1  06:35.060 RON 0 <newcc> Suppress: None
2  06:35.106 2020000 <rsro><rfco>  Unknown command
3  06:36.141 2020020 <rsro><rfco>  Unknown command
```

3.3.6 Listing

`\listing{GcProc.l}` File is too long to list

4 Breakout Specific Routines

4.1 P2cmnds.l

4.1.1 Description

DB breaks sequences of commands whenever the direction of data flow changes. Sometimes additional breaks are desirable and P2CMNDS makes them.

4.1.2 Calling Sequence

```
P2cmnds infile outfile
```

4.1.3 Input Data Format

P2CMNDS expects the output of DB to be its input.

```
1
2   O-02 O-41 O-44 O-30 O-31 O-3B O-52 O-4F O-4E O-3A O-30 O-03
3   I-06 I-02 I-41 I-44 I-30 I-31 I-3B I-52 I-4F I-4E I-03
4   O-02 O-41 O-44 O-30 O-31 O-3B O-47 O-43 O-37 O-3A O-32 O-30 O-32 O-30 O-30 O-30 O-30 O-03
5   I-06
```

4.1.4 Internal Processing

Whenever any of several data patterns are discovered, they are written out with with a carriage return/line feed inserted in them. The patterns that are "broken" are: (\n indicates the location of the carriage return/line feed pair.)

1. ␣I-03␣I-02 is translated to: ␣I-03\n␣I-02.
2. ␣O-03␣O-02 is translated to: ␣O-03\n␣O-02.
3. ␣O-06␣O-02 is translated to: ␣O-06\n␣O-02.
4. ␣I-06␣I-02 is translated to: ␣I-06\n␣I-02.

4.1.5 Output

Typical results of this step of processing are:

```
1
2   O-02 O-41 O-44 O-30 O-31 O-3B O-52 O-4F O-4E O-3A O-30 O-03
3   I-06
4   I-02 I-41 I-44 I-30 I-31 I-3B I-52 I-4F I-4E I-03
5   O-02 O-41 O-44 O-30 O-31 O-3B O-47 O-43 O-37 O-3A O-32 O-30 O-32 O-30 O-30 O-30 O-30 O-03
6   I-06
```

4.1.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/P2cmnds.l,v 1.4 2004-12-01 07:11:13-08 Hamilton Exp Hamilton
$
 3  // FIX0 is used as a "universal" starter skeleton for FLEX programs
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <time.h>
 7
 8  #undef yywrap
 9  %}
10  %%
11  "#".*\n        ECHO;   // Print out comments starting with a # sign
12
13  " I-03 I-02 " fprintf(yyout," I-03\n I-02 ");
14  " O-03 O-02 " fprintf(yyout," O-03\n O-02 ");
15  " O-06 O-02 " fprintf(yyout," O-06\n O-02 ");
16  " I-06 I-02 " fprintf(yyout," I-06\n I-02 ");
17
18  .|\n     ECHO;
19  %%
20  // yywrap is for End Of File processing, 1 = done
21  int yywrap(void)
22  {
23          return 1;
24  }
25
26  main(int argc, char *argv[])
27  {
28  char infilename[100];
29  char outfilename[100];
30
31  time_t thistime;
32  struct tm *t;
33
34          fprintf(stderr,"$Id: P2cmnds.l,v 1.4 2004-12-01 07:11:13-08 Hamilton Exp Hamilton
$\n");
35          strcpy(infilename,"stdin");
36          strcpy(outfilename,"stdout");
37
38          // Do we have to deal with stdin/stdout?
39          // 3 = one for the program name and one each for input and output names
40          if (argc == 3) {
41              // Nope
42              // Get over the program, etc., name
43              argc--;
44              argv++;
45              yyin = fopen(argv[0],"r");
46              if (yyin == NULL) {
47                  fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
48                                  argv[0]);
49                  exit(EXIT_FAILURE);
```

```
50              }
51              strcpy(infilename,argv[0]);
52
53              // Now get over the input file name and get to the output file name
54              argc--;
55              argv++;
56              yyout = fopen(argv[0],"w");
57              if (yyout == NULL) {
58                  fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
59                                  argv[0]);
60                  exit(EXIT_FAILURE);
61              }
62              strcpy(outfilename,argv[0]);
63          }
64          else {
65              // Woops we are using std... type io
66              yyin  = stdin;
67              yyout = stdout;
68          }
69
70          fprintf(yyout,"#\n# %12s, %4d: $Id: P2cmnds.l,v 1.4 2004-12-01 07:11:13-08 Hamilton Exp
Hamilton $\n",
71                          __FILE__,__LINE__);
72          t = time(NULL);
73          thistime = localtime(&t);
74          fprintf(yyout,"# %12s, %4d: %s",
75                          __FILE__,__LINE__,asctime(thistime));
76          fprintf(yyout,"# %12s, %4d: Reading from \"%s\"\n",
77                          __FILE__,__LINE__,infilename);
78          fprintf(yyout,"# %12s, %4d: Writing into \"%s\"\n#\n",
79                          __FILE__,__LINE__,outfilename);
80
81          yylex();
82          exit(EXIT_SUCCESS);
83  }
84
85  // $Log: P2cmnds.l,v $
86  // Revision 1.4  2004-12-01 07:11:13-08  Hamilton
87  // Normal end of day data saving
88  //
89  // Revision 1.3  2004-11-30 15:07:13-08  Hamilton
90  // Normal end of day data saving
91  //
92  // Revision 1.2  2004-11-30 13:06:10-08  Hamilton
93  // Normal end of day data saving
94  //
95  // Revision 1.1  2004-11-12 07:10:11-08  Hamilton
96  // Initial revision
97  //
98
```

4.2 PFormat.l

4.2.1 Description

This routine is used to "translate" capture files from BREAKOUT format files into FTS format capture files. Prior to running this filter the BREAKOUT capture file must have been run through DB, which converts the raw breakout capture file into an ASCII character representation of the data. And through P2CMNDS to complete the clean up.

This has been done so that all processing of Panasonic protocol data may use a common set of analysis filters.

4.2.2 Calling Sequence

```
PFormat infile outfile
```

4.2.3 Input Data Format

This routine is designed to take a BREAKOUT capture file that has been processed by P2CMNDS. The output of P2CMNDS is as follows:

```
1
2   O-02 O-41 O-44 O-30 O-31 O-3B O-52 O-4F O-4E O-3A O-30 O-03
3   I-06
4   I-02 I-41 I-44 I-30 I-31 I-3B I-52 I-4F I-4E I-03
5   O-02 O-41 O-44 O-30 O-31 O-3B O-47 O-43 O-37 O-3A O-32 O-30 O-32 O-30 O-30 O-30 O-30 O-03
6   I-06
```

4.2.4 Internal Processing

Before the file is read, a fake header is written out. While processing the file, any lines that are comment lines are deleted. Then each five character ASCII representation of a BREAKOUT captured protocol byte is translated by outputting:

1. A data source ID code, "`DCE␣`" or "`DCD␣`".
2. A fake date of "`11/22/3333␣`".
3. A fake time tag which is the message number. The message number is incremented on each `ETX`
4. Followed by a blank and the actual data byte.

4.2.5 Output

This routine identifies its self on the screen and in the output file. The output file has the program's name written as a comment line.

Typical results of this step of processing are:

```
 1  FTS capture file, generated from a breakout file.
 2  Event <none>
 3  Event <none>
 4
 5  Timestamp <none>
 6  Sd Timestamp <none>
 7
 8  DTE 11/22/3333 44:00:00.000 PM 02
 9  DTE 11/22/3333 44:00:00.000 PM 41
10  DTE 11/22/3333 44:00:00.000 PM 44
11  DTE 11/22/3333 44:00:00.000 PM 30
12  DTE 11/22/3333 44:00:00.000 PM 31
13  DTE 11/22/3333 44:00:00.000 PM 3b
14  DTE 11/22/3333 44:00:00.000 PM 52
15  DTE 11/22/3333 44:00:00.000 PM 4f
16  DTE 11/22/3333 44:00:00.000 PM 4e
17  DTE 11/22/3333 44:00:00.000 PM 3a
18  DTE 11/22/3333 44:00:00.000 PM 30
19  DTE 11/22/3333 44:00:00.000 PM 03
20  DCE 11/22/3333 44:00:00.001 PM 06
21  DCE 11/22/3333 44:00:00.001 PM 02
22  DCE 11/22/3333 44:00:00.001 PM 41
23  DCE 11/22/3333 44:00:00.001 PM 44
24  DCE 11/22/3333 44:00:00.001 PM 30
25  DCE 11/22/3333 44:00:00.001 PM 31
26  DCE 11/22/3333 44:00:00.001 PM 3b
27  DCE 11/22/3333 44:00:00.001 PM 52
28  DCE 11/22/3333 44:00:00.001 PM 4f
29  DCE 11/22/3333 44:00:00.001 PM 4e
30  DCE 11/22/3333 44:00:00.001 PM 03
31  DTE 11/22/3333 44:00:00.002 PM 02
32  DTE 11/22/3333 44:00:00.002 PM 41
33  DTE 11/22/3333 44:00:00.002 PM 44
34  DTE 11/22/3333 44:00:00.002 PM 30
35  DTE 11/22/3333 44:00:00.002 PM 31
36  DTE 11/22/3333 44:00:00.002 PM 3b
37  DTE 11/22/3333 44:00:00.002 PM 47
38  DTE 11/22/3333 44:00:00.002 PM 43
39  DTE 11/22/3333 44:00:00.002 PM 37
40  DTE 11/22/3333 44:00:00.002 PM 3a
41  DTE 11/22/3333 44:00:00.002 PM 32
42  DTE 11/22/3333 44:00:00.002 PM 30
43  DTE 11/22/3333 44:00:00.002 PM 32
44  DTE 11/22/3333 44:00:00.002 PM 30
45  DTE 11/22/3333 44:00:00.002 PM 30
46  DTE 11/22/3333 44:00:00.002 PM 30
```

```
47  DTE 11/22/3333 44:00:00.002 PM 30
48  DTE 11/22/3333 44:00:00.002 PM 03
49  DCE 11/22/3333 44:00:00.003 PM 06
```

4.2.6 Listing

```
\listing{PFormat.l}
```

5 Oscilloscope Specific Routines

5.1 FixPs.l

FixPs is used to deconcatenate concatenate files. Concatenationing occurs when oscilloscope encapsulated postscript image pictures are sent to a PC without saving each one individually as they come in. This is a common event when Windows' HyperTerm utility.

5.1.1 Description

FixPs reads the input file and writes output files. Each output file starts with a "`%!PS-Adobe`" marker and ends with a "`%%EOF`" marker.

5.1.2 Calling Sequence

```
FixPs infile
```

5.1.3 Input Data Format

The input file consists of one, or more, Postscript picture records.

5.1.4 Internal Processing

The input file is read. When FixPs starts to run, it opens an output file of "`test00.out`" and writes the input file data into it. When it detects and end of file marker ("`%%EOF`"), the file is closed. If FixPs now reads a subsequent start of file mark ("`%!PS-Adobe`"), increments the last two digits of the output file name (`test00.out` becomes `test01.out` etc.) and starts to copy the input data into it until it comes to an end of file marker. The process then repeats.

5.1.5 Output

Each output file consists of one Postscript picture record.

5.1.6 Listing

```
%{
// $Header: d:/txb-p/Captures/RCS/FixPs.l,v 1.2 2004-12-01 11:06:35-08 Hamilton Exp Hamilton $
// FIXPS is used to "deconcatenate" concatenated encapsulated postscript files.
//
#include <stdio.h>
#include <stdlib.h>
#undef yywrap

int thisfile = 0;
int outvalid = 0;
FILE *outfile;
char outname[100];

void openfile(void);
void closefile(void);
void mightdump(void);

%}
x START END
%%

"%!PS-Adobe"    openfile();
"%%EOF"         closefile();

.|\n            mightdump();
%%
// mightdump is used to conditionally dump data instead of writing it to disk
void mightdump(void)
{
        if (outvalid == 1) {
            fprintf(outfile,"%s",yytext);
        }
}

// openfile is used to open output files
void openfile(void)
{
        outvalid = 0;
        thisfile++;
        sprintf(outname,"test%02d.ps",thisfile);
        fprintf(stderr,"Open file  %s\n",outname);
        outfile = fopen(outname,"wb");
        if (outfile == NULL) {
            fprintf(stderr,"Can not open %s for output, quitting",outname);
            exit(EXIT_FAILURE);
        }
        outvalid = 1;
        mightdump();
}

// closefile is used to close output files
```

```
52  void closefile(void)
53  {
54          mightdump();
55          fprintf(stderr,"Close file %s\n\n",outname);
56          fclose(outfile);
57          outvalid = 0;
58  }
59
60  // yywrap is for End Of File processing, 1 = done
61  int yywrap(void)
62  {
63          return 1;
64  }
65
66  main(int argc, char *argv[])
67  {
68          fprintf(stderr,"$Id: FixPs.l,v 1.2 2004-12-01 11:06:35-08 Hamilton Exp Hamilton $\n");
69
70          // 2 = one for the program name and each for the input file name
71          if (argc == 2) {
72              // Nope
73              // Get over the program, etc., name
74              argc--;
75              argv++;
76              yyin = fopen(argv[0],"rb");
77              if (yyin == NULL) {
78                  fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
79                                  argv[0]);
80                  exit(EXIT_FAILURE);
81              }
82          }
83          else {
84              fprintf(stderr,"\aIncorrect number of argruments (%d instead of 1) "
85                             " provided, qutting",
86                               argc-1);
87              exit(EXIT_FAILURE);
88          }
89
90          yylex();
91          exit(EXIT_SUCCESS);
92  }
93
94  // $Log: FixPs.l,v $
95  // Revision 1.2  2004-12-01 11:06:35-08  Hamilton
96  // Normal end of day data saving
97  //
98  // Revision 1.1  2004-08-19 07:26:25-07  Hamilton
99  // Initial revision
100 //
101 // Revision 1.1  2004-08-17 09:40:24-07  Hamilton
102 // Normal end of day data saving
103 //
```

```
104
```

Index