FTS Capture file Fixes

Also covers breakout capture files

30 November 2004

TABLE OF CONTENTS

Section **Page**

[1] `$Header: d:/txb-p/Captures/RCS/DocFix.tex,v 1.16 2004-11-30 09:41:04-08 Hamilton Exp Hamilton $`

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`
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

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 begining with a per-cent sign "%", have been deleted in the examples to save paper.

2 Fix1.l

2.1 Description

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

2.2 Calling Sequence

```
fix1 infile outfile
```

2.3 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
```

2.4 Internal Processing

This routine will make comment lines, by prepending a percent 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␣".

2.5 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
```

2.6 Listing

```
%{
// $Header: d:/txb-p/Captures/RCS/Fix1.l,v 1.16 2004-11-30 15:07:13-08 Hamilton Exp Hamilton $
// FIX1 is used to get rid of some unused data in an FTS comm line dump.
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#undef yywrap
%}
%%
"%".*\n        ECHO;   // Print only comments starting with a % sign

^"FTS".*\n   |
^"Event".*\n |
^\n                 {
                     fprintf(yyout,"%% %s     %5d %s",__FILE__,__LINE__,
                                   yytext);
                    }

^"Timestamp".*\n     ;
^"Sd".*\n            ;

^"DCE "              ECHO;
^"DTE "              ECHO;

^[0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;
[0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;

[0-9]{1,2}":"[0-9]{2}"."[0-9]{3}" "{1,5}$ ;

" PM"                ;
" AM"                ;

.|\n     ECHO;
%%

// This rule:
// ^[0-9]{1,2}"/"[0-9]{1,2}"/"[0-9]{4}" "[0-9]{1,2}":" ;
// Means:
// Starting in the first column
// 1 or 2 numbers from 0 to 9
// followed by a /
// then 1 or 2 numbers
// followed by a /
// and 4 numbers
// with a blank following
// then 1 or two numbers from 0 to 9
// and a trailing colon/blank ": ".

// yywrap is for End Of File processing, 1 = done
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: Fix1.l,v 1.16 2004-11-30 15:07:13-08 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%% %s     %5d $Id: Fix1.l,v 1.16 2004-11-30 15:07:13-08 Hamilton Exp
 Hamilton $\n",
102                         __FILE__,__LINE__);
```

```
103         t = time(NULL);
104         thistime = localtime(&t);
105         fprintf(yyout,"%% %s     %5d %s",
106                         __FILE__,__LINE__,asctime(thistime));
107         fprintf(yyout,"%% %s     %5d Reading from \"%s\"\n",
108                         __FILE__,__LINE__,infilename);
109         fprintf(yyout,"%% %s     %5d Writing into \"%s\"\n%%\n",
110                         __FILE__,__LINE__,outfilename);
111
112         yylex();
113         exit(EXIT_SUCCESS);
114 }
115
116 // $Log: Fix1.l,v $
117 // Revision 1.16  2004-11-30 15:07:13-08  Hamilton
118 // Normal end of day data saving
119 //
120 // Revision 1.15  2004-11-30 13:06:10-08  Hamilton
121 // Normal end of day data saving
122 //
123 // Revision 1.14  2004-11-29 15:41:23-08  Hamilton
124 // Normal end of day data saving
125 //
126 // Revision 1.13  2004-11-11 15:53:16-08  Hamilton
127 // Normal end of day data saving
128 //
129 // Revision 1.12  2004-11-11 08:45:50-08  Hamilton
130 // Normal end of day data saving
131 //
132 // Revision 1.11  2004-11-11 07:45:20-08  Hamilton
133 // Normal end of day data saving
134 //
135 // Revision 1.10  2004-11-11 07:38:03-08  Hamilton
136 // Normal end of day data saving
137 //
138 // Revision 1.9  2004-11-11 07:14:22-08  Hamilton
139 // Normal end of day data saving
140 //
141 // Revision 1.8  2004-11-04 12:17:49-08  Hamilton
142 // Normal end of day data saving
143 //
144
```

3 Fix2.l

3.1 Description

This routine is used to condense the "one byte per line" format that FIX1.L generates into something more useful.

3.2 Calling Sequence

```
fix2 infile outfile
```

3.3 Input Data Format

This routine is designed to take the output from FIX1.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
```

3.4 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.

3.5 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 ;
```

3.6 Listing

```
%{
// $Header: d:/txb-p/Captures/RCS/Fix2.l,v 1.15 2004-11-30 15:23:17-08 Hamilton Exp Hamilton $
// FIX2 is used as one step in reformatting Panasonic protocol data
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#undef yywrap

char timetag[100];

int makenumb(void);
void makeascii(void);

%}
%x INMSG IN OUT OUTMSG EITHER EITHERMSG
%%
^"%".*\n        ECHO;   // Print only comments starting with a % sign


^"DCE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " {strcpy(timetag,yytext);
                                               BEGIN IN;
                                              }

<IN>[0-9a-f]{2}     {// Look for an STX or ACK data byte
                     if (yytext[0] == '0' && yytext[1] == '2') {
                         fprintf(yyout,"%sSTX  ",timetag);
                         BEGIN INMSG;
                      } else
                      if (yytext[0] == '0' && yytext[1] == '6') {
                          fprintf(yyout,"%sACK\n",timetag);
                          BEGIN 0;
                      }
                     }

<INMSG>[0-9a-f]{2} {// While in the message look for an ETX byte
                    if (yytext[0] == '0' && yytext[1] == '3') {
                        fprintf(yyout,"ETX\n");
                        BEGIN 0;
                    }
                    else {// Not an ETX, make it printable
                        makeascii();
                    }
                   }

<INMSG>^"DCE "[0-9]{1,2}":"[0-9]{1,2}"."[0-9]{3}" " ;
<INMSG>.|\n        ;



^"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: Fix2.l,v 1.15 2004-11-30 15:23:17-08 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%% %s     %5d $Id: Fix2.l,v 1.15 2004-11-30 15:23:17-08 Hamilton Exp
 Hamilton $\n",__FILE__,__LINE__);
204         fprintf(yyout,"%% %s     %5d Reading from \"%s\"\n",
205                         __FILE__,__LINE__,infilename);
206         fprintf(yyout,"%% %s     %5d Writing into \"%s\"\n",
```

```
207                         __FILE__,__LINE__,outfilename);
208
209         t = time(NULL);
210         thistime = localtime(&t);
211         fprintf(yyout,"%% %s     %5d %s",
212                         __FILE__,__LINE__,asctime(thistime));
213
214         yylex();
215         exit(EXIT_SUCCESS);
216 }
217
218 // $Log: Fix2.l,v $
219 // Revision 1.15  2004-11-30 15:23:17-08  Hamilton
220 // Normal end of day data saving
221 //
222 // Revision 1.14  2004-11-30 15:07:13-08  Hamilton
223 // Normal end of day data saving
224 //
225 // Revision 1.13  2004-11-30 13:06:10-08  Hamilton
226 // Normal end of day data saving
227 //
228 // Revision 1.12  2004-11-15 15:31:39-08  Hamilton
229 // Normal end of day data saving
230 //
231 // Revision 1.11  2004-11-11 13:27:22-08  Hamilton
232 // Normal end of day data saving
233 //
234 // Revision 1.10  2004-11-11 08:45:51-08  Hamilton
235 // Normal end of day data saving
236 //
237 // Revision 1.9  2004-11-11 07:45:21-08  Hamilton
238 // Normal end of day data saving
239 //
240 // Revision 1.8  2004-11-11 07:24:17-08  Hamilton
241 // Normal end of day data saving
242 //
243 // Revision 1.7  2004-11-11 07:14:22-08  Hamilton
244 // Normal end of day data saving
245 //
246 // Revision 1.6  2004-11-04 10:47:03-08  Hamilton
247 // Normal end of day data saving
248 //
249
```

4 UnBlank.l

4.1 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.

4.2 Calling Sequence

```
UnBlank infile outfile
```

4.3 Input Data Format

This routine is designed to take the output from FIX2.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 ;
```

4.4 Internal Processing

The input file is scanned for items of interest. They are transformed (Section 4.1, page 19) or passed through (Section 4.1, page 19). Other items are dumped.

4.5 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;
```

4.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/UnBlank.l,v 1.11 2004-11-30 15:07:15-08 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 1.11 2004-11-30 15:07:15-08 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%% %s  %5d $Id: UnBlank.l,v 1.11 2004-11-30 15:07:15-08 Hamilton Exp
Hamilton $\n",__FILE__,__LINE__);
85
86          t = time(NULL);
87          thistime = localtime(&t);
88          fprintf(yyout,"%% %s  %5d %s",
89                          __FILE__,__LINE__,asctime(thistime));
90          fprintf(yyout,"%% %s  %5d Reading from \"%s\"\n",
91                          __FILE__,__LINE__,infilename);
92          fprintf(yyout,"%% %s  %5d Writing into \"%s\"\n",
93                          __FILE__,__LINE__,outfilename);
94
95          yylex();
96          exit(EXIT_SUCCESS);
97  }
98
99  // $Log: UnBlank.l,v $
100 // Revision 1.11  2004-11-30 15:07:15-08  Hamilton
```

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

5 GcProc.l

5.1 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:

```
fix1 <infile | fix2 | unblank | GcProc >outfile
```

5.2 Calling Sequence

```
GcProc infile outfile
```

5.3 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;
```

5.4 Internal Processing

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

5.5 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
```

5.6 Listing

```
 1 %{
 2 // $Header: d:/txb-p/Captures/RCS/GcProc.l,v 1.7 2004-11-30 15:07:14-08 Hamilton Exp Hamilton $
 3 // GCPROC is used to decode Panasonic Protocol commands.
 4 #include <stdio.h>
 5 #include <stdlib.h>
 6 #include <strings.h>
 7 #include <time.h>
 8
 9 #undef yywrap
10
11 // This logic is used to print out additional data for debugging the
12 // software. This is only used with compile time debugging.
13 #define DEBUGRUN 1
14 #ifdef DEBUGRUN
15 #define DM(comment,message) strcat(decodedcmnd,comment ## message);
16 #else
17 #define DM(comment,message) strcat(decodedcmnd,message);
18 #endif // #ifdef DEBUGRUN
19
20 int linecount = 0;
21 int type;
22
23 char decodedcmnd[1000];
24
25 char from[20];
26 char *end_P;
27
28 void decode002(void);
29 void decode202(void);
30 void decode902(void);
31 void decode903(void);
32 void decoderon(void);
33 void decodeer(void);
34 void decodealm(void);
35
36 void donecmnd(void);
37
38 %}
39 %x DOCMND DO002 DO202 DO902 DO903 DORON DOER DOALM
40 %%
41 "%".*              strcpy(decodedcmnd,yytext); donecmnd();
42
43 "DTE "             strcat(decodedcmnd," ---> ");
44 "DCE "             strcat(decodedcmnd,"<---  ");
45
46 [0-9]{2}":"[0-9]{2}"."[0-9]{3}" " strcat(decodedcmnd,yytext);
47
48 "ACK"              strcat(decodedcmnd,"ACK\n"); donecmnd();
49 " ACK"             strcat(decodedcmnd," ACK\n"); donecmnd();
50
51 "ETX"              ;
```

```
52  "AD01; "            ;
53  "STX"               ;
54
55  "GC7: "             BEGIN DOCMND;
56  "RON"               BEGIN DORON;
57  "ER"                BEGIN DOER;
58  "ALM"               BEGIN DOALM;
59
60  <DOCMND>"002"       BEGIN DO002;
61  <DOCMND>"202"       BEGIN DO202;
62  <DOCMND>"902"       BEGIN DO902;
63  <DOCMND>"903"       BEGIN DO903;
64
65
66  <DO002>[0-9A-F]{4} decode002(); BEGIN 0;
67  <DO202>[0-9A-F]{4} decode202(); BEGIN 0;
68  <DO902>[0-9A-F]{4} decode902(); BEGIN 0;
69  <DO903>[0-9A-F]{4} decode903(); BEGIN 0;
70  <DORON>.*\n         decoderon(); BEGIN 0;
71  <DOER>.*\n          decodeer();  BEGIN 0;
72  <DOALM>.*\n         decodealm(); BEGIN 0;
73
74
75
76  .|\n         ;
77
78  %%
79  // This is used to decode ALM type messages
80  //
81  // Format of an ALM message is:
82  //
83  //  From the equipment it is:
84  //
85  //      STX ADaa;ALM:nn ETX (with no blanks and the binary
86  //                           representation of STX/ETX}
87  //
88  //  When it gets to GCPROC.L it has been reformatted to become:
89  //
90  //      "STX ADaa; ALM: nn ETX<cr>" (With blanks, etc.)
91  //
92  //  The following adbreviatons were used in the above examples:
93  //
94  //      aa = Devide address
95  //      cr = End of line
96  //      nn = Alarm number
97  //
98  //  There is no documentation that relates alarm numbers to what
99  //  they are trying to represent. (I.e. their cause.)
100 //
101 void decodealm(void)
102 {
103 char temp[10];
```

```
104
105     from[0] = '0';
106     from[1] = 'x';
107     if (yytext[0] == ':') {
108         from[2] = yytext[2];
109         from[3] = yytext[3];
110         from[4] = '\0';
111         type = strtol(from,&end_P,16);
112         sprintf(temp,"%02d",type);
113         strcat(decodedcmnd,"ALM: ");
114         strcat(decodedcmnd,temp);
115     }
116     donecmnd();
117 }
118
119
120
121 // This is used to decode RON type messages
122 void decoderon(void)
123 {
124     from[0] = '0';
125     from[1] = 'x';
126     if (yytext[0] == ':') {
127         from[2] = yytext[2];
128         from[3] = '\0';
129         type = strtol(from,&end_P,16);
130         switch (type) {
131             case 0x7: DM("RON 7 <newcc>"," Suppress: Acks, Answers, Errors"); break;
132             case 0x6: DM("RON 6 <newcc>"," Suppress: Acks, Answers"); break;
133             case 0x5: DM("RON 5 <newcc>"," Suppress: Acks, Errors"); break;
134             case 0x4: DM("RON 4 <newcc>"," Suppress: Acks,"); break;
135             case 0x3: DM("RON 3 <newcc>"," Suppress: Answers, Errors"); break;
136             case 0x2: DM("RON 2 <newcc>"," Suppress: Answers"); break;
137             case 0x1: DM("RON 1 <newcc>"," Suppress: Errors"); break;
138             case 0x0: DM("RON 0 <newcc>"," Suppress: None"); break;
139             default:
140                    DM(" <newcc> ","Illegal RON code of <");
141                    strcat(decodedcmnd,yytext);
142                    strcat(decodedcmnd,">");
143                    break;
144         }
145     }
146     else {
147         DM("RON <newcc> ","");
148         strcat(decodedcmnd,yytext);
149     }
150     donecmnd();
151 }
152
153
154 // This is used to decode ER type messages.
155 void decodeer(void)
```

```
156  {
157      from[2] = yytext[0];
158      from[3] = yytext[1];
159      from[4] = yytext[2];
160      from[5] = '\0';
161      type = strtol(from,&end_P,16);
162      switch (type) {
163          case 0x001: DM("ER001 <newccc> ","Invalid command: General reasons"); break;
164          case 0x002: DM("ER002 <newccc> ","Invalid parameters"); break;
165          case 0x301: DM("ER301 <newccc> ","Invalid command: Cannot execute depending on camera
 mode"); break;
166          case 0x305: DM("ER305 <newccc> ","Invalid command: Cannot answer for execution
 waiting"); break;
167          case 0x306: DM("ER306 <newccc> ","Invalid command: Cannot execute for execution
 waiting"); break;
168          case 0x601: DM("ER601 <newccc> ","Wrong command: Cannot execute depending on menu
 setting"); break;
169          default:
170                 DM("<newccc> ","Illegal ER code of <");
171                 strcat(decodedcmnd,yytext);
172                 strcat(decodedcmnd,">");
173                 break;
174      }
175      donecmnd();
176  }
177
178
179  // This is used to decode a four hex digit command.
180  void decode002(void)
181  {
182      from[2] = yytext[0];
183      from[3] = yytext[1];
184      from[4] = yytext[2];
185      from[5] = yytext[3];
186      from[6] = '\0';
187
188      type = strtol(from,&end_P,16);
189      switch (type) {
190          case 0x0102: DM("0020102 <csro> ","ALC/BLC?"); break;
191          case 0x0110: DM("0020110 <csro> ","Shutter?"); break;
192          case 0x0112: DM("0020112 <csro> ","Sens Up Auto (Set-up)?"); break;
193          case 0x0114: DM("0020114 <csro> ","Sens Up Auto (Condition)?"); break;
194          case 0x0116: DM("0020116 <csro> ","Sens Up Manual (Set-up)?"); break;
195          case 0x0121: DM("0020121 <csro> ","AGC?"); break;
196          case 0x0122: DM("0020122 <csro> ","Gain Up?"); break;
197          case 0x0124: DM("0020124 <csro> ","ELC?"); break;
198          case 0x0130: DM("0020130 <csro> ","White Balance?"); break;
199          case 0x0140: DM("0020140 <csro> ","Line Lock?"); break;
200          case 0x0176: DM("0020176 <csro> ","Motion Detect?"); break;
201          case 0x01A2: DM("00201A2 <csro> ","Auto Focus Area ?"); break;
202          case 0x1002: DM("0021002 <cfcn> ","Iris (camera) Open"); break;
203          case 0x1003: DM("0021003 <cfcn> ","Iris (camera) Close"); break;
```

```
204          case 0x1004: DM("0021004 <cfcn> ","Iris (camera) Stop"); break;
205          case 0x1005: DM("0021005 <cfcn> ","Iris (camera) Reset"); break;
206          case 0x1032: DM("0021032 <cfcn> ","ALC On"); break;
207          case 0x1033: DM("0021033 <cfcn> ","ELC On"); break;
208          case 0x1034: DM("0021034 <cfcn> ","Manual Iris On"); break;
209          case 0x1040: DM("0021040 <cfcn> ","BW On"); break;
210          case 0x1041: DM("0021041 <cfcn> ","BW Off"); break;
211          case 0x1042: DM("0021042 <cfcn> ","BW Auto"); break;
212          case 0x1050: DM("0021050 <cfcn> ","BW Burst On"); break;
213          case 0x1051: DM("0021051 <cfcn> ","BW Burst Off"); break;
214          case 0x1100: DM("0021100 <cfcn> ","Shutter On"); break;
215          case 0x1101: DM("0021101 <cfcn> ","Shutter Off"); break;
216          case 0x1102: DM("0021102 <cfcn> ","Shutter Inc"); break;
217          case 0x1103: DM("0021103 <cfcn> ","Shutter Dec"); break;
218          case 0x1109: DM("0021109 <cfcn> ","Shutter 1/100"); break;
219          case 0x110A: DM("002110A <cfco> ","Shutter 1/120"); break;
220          case 0x110C: DM("002110C <cfcn> ","Shutter 1/250"); break;
221          case 0x110D: DM("002110D <cfcn> ","Shutter 1/500"); break;
222          case 0x110E: DM("002110E <cfcn> ","Shutter 1/1000"); break;
223          case 0x110F: DM("002110F <cfcn> ","Shutter 1/2000"); break;
224           case 0x1119: DM("0021119 <cfcn> ","Shutter 1/4000"); break;
225          case 0x111A: DM("002111A <cfcn> ","Shutter 1/10000"); break;
226           case 0x1120: DM("0021120 <cfcn> ","Sens Up Auto On"); break;
227           case 0x1121: DM("0021121 <cfcn> ","Sens Up Auto Off"); break;
228           case 0x1122: DM("0021122 <cfcn> ","Sens Up Auto Inc"); break;
229           case 0x1123: DM("0021123 <cfcn> ","Sens Up Auto Dec"); break;
230          case 0x1150: DM("0021150 <cfcn> ","Sens Up Manual On"); break;
231          case 0x1151: DM("0021151 <cfcn> ","Sens Up Manual Off"); break;
232          case 0x1152: DM("0021152 <cfcn> ","Sens Up Manual Inc"); break;
233          case 0x1153: DM("0021153 <cfcn> ","Sens Up Manual Dec"); break;
234          case 0x1200: DM("0021200 <cfcn> ","AGC On"); break;
235          case 0x1201: DM("0021201 <cfcn> ","AGC Off"); break;
236          case 0x1208: DM("0021208 <cfcn> ","AGC Gain Up Low"); break;
237          case 0x1209: DM("0021209 <cfcn> ","AGC Gain Up Mid"); break;
238          case 0x120A: DM("002120A <cfcn> ","AGC Gain Up High"); break;
239          case 0x1224: DM("0021224 <rfco> ","Zoom [stop(f)] + Focus [stop(f)]"); break;
240          case 0x1228: DM("0021228 <cfcn> ","Manual Gain Up Low OR <rfco> Zoom [tele(f)]");
 break;
241          case 0x1229: DM("0021229 <cfcn> ","Manual Gain Up Mid OR <rfco> Zoom [tele(f)] + Focus
 [far(f)]"); break;
242          case 0x122A: DM("002122A <cfcn> ","Manual Gain Up High OR <rfco> Focus [far(f)]");
 break;
243          case 0x122B: DM("002122B <rfco> ","Zoom [wide(f)] + Focus [far(f)]"); break;
244          case 0x122C: DM("002122C <rfco> ","Zoom [wide(f)]"); break;
245          case 0x122D: DM("002122D <rfco> ","Zoom [wide(f)] + Focus [near(f)]"); break;
246          case 0x122E: DM("002122E <rfco> ","Focus [near(f)]"); break;
247          case 0x122F: DM("002122F <rfco> ","Zoom [wide(f)] + Focus [near(f)]"); break;
248          case 0x12A2: DM("00212A2 <cfcn> ","BLC Mode Auto"); break;
249          case 0x12A3: DM("00212A3 <cfcn> ","BLC Mode Preset"); break;
250          case 0x12B0: DM("00212B0 <cfcn> ","BLC Setup [start]"); break;
251          case 0x12B1: DM("00212B1 <cfcn> ","BLC Setup [end]"); break;
252          case 0x12C0: DM("00212C0 <cfcn> ","BLC Setup [mask on]"); break;
```

```
253          case 0x12C1: DM("00212C1 <cfcn> ","BLC Setup [mask off]"); break;
254          case 0x12C2: DM("00212C2 <cfcn> ","BLC Setup [mask clear]"); break;
255          case 0x12C3: DM("00212C3 <cfcn> ","BLC Setup [mask reverse]"); break;
256          case 0x12C8: DM("00212C8 <cfcn> ","BLC Setup [cursor up]"); break;
257          case 0x12C9: DM("00212C9 <cfcn> ","BLC Setup [cursor right]"); break;
258          case 0x12CA: DM("00212CA <cfcn> ","BLC Setup [cursor down]"); break;
259          case 0x12CB: DM("00212CB <cfcn> ","BLC Setup [cursor left]"); break;
260          case 0x1300: DM("0021300 <cfco> ","AWC On OR <rfco> Auto Pan On OR <cfco> AWC Set Up
 1"); break;
261          case 0x1301: DM("0021301 <rfco> ","Auto Pan Off"); break;
262          case 0x1305: DM("0021305 <cfcn> ","AWC Reset"); break;
263          case 0x1306: DM("0021306 <cfco> ","AWC Set Up 2"); break;
264          case 0x130E: DM("002130E <rfco> ","Auto Pan Endless On"); break;
265          case 0x130F: DM("002130F <rfco> ","Auto Pan Endless Off"); break;
266          case 0x1310: DM("0021310 <cfcn> ","ATW On"); break;
267          case 0x1312: DM("0021312 <cfcn> ","R Offset Up"); break;
268          case 0x1313: DM("0021313 <cfcn> ","R Offset Down"); break;
269          case 0x1314: DM("0021314 <cfcn> ","R Offset Stop"); break;
270          case 0x1315: DM("0021315 <cfcn> ","R Offset Reset"); break;
271          case 0x1316: DM("0021316 <cfcn> ","B Offset Up"); break;
272          case 0x1317: DM("0021317 <cfcn> ","B Offset Down"); break;
273          case 0x1318: DM("0021318 <cfcn> ","B Offset Stop"); break;
274          case 0x1319: DM("0021319 <cfcn> ","B Offset Reset"); break;
275          case 0x1324: DM("0021324 <rfco> ","Pan [stop(f)] + Tilt [stop(f)]"); break;
276          case 0x1328: DM("0021328 <rfco> ","Pan [left(f)]"); break;
277          case 0x1329: DM("0021329 <rfco> ","Pan [left(f)] + Tilt [up(f)]"); break;
278          case 0x132A: DM("002132A <rfco> ","Tilt [up(f)]"); break;
279          case 0x132B: DM("002132B <rfco> ","Pan [right(f)] + Tilt [up(f)]"); break;
280          case 0x132C: DM("002132C <rfco> ","Pan [right(f)]"); break;
281          case 0x132D: DM("002132D <rfco> ","Pan [right(f)] + Tilt [down(f)]"); break;
282          case 0x132E: DM("002132E <rfco> ","Tilt [down(f)]"); break;
283          case 0x132F: DM("002132F <rfco> ","Pan [left(f)] + Tilt [down(f)]"); break;
284          case 0x1382: DM("0021382 <cfcn> ","Burst Phase Up (+)"); break;
285          case 0x1383: DM("0021383 <cfcn> ","Burst Phase Down (-)"); break;
286          case 0x1384: DM("0021384 <cfcn> ","Burst Phase Stop"); break;
287          case 0x1385: DM("0021385 <cfcn> ","Burst Phase Reset"); break;
288          case 0x1452: DM("0021452 <cfcn> ","Line Lock On"); break;
289          case 0x1453: DM("0021453 <cfcn> ","Line Lock Off (int)"); break;
290          case 0x1454: DM("0021454 <cfcn> ","LL Phase Up"); break;
291          case 0x1455: DM("0021455 <cfcn> ","LL Phase Down"); break;
292          case 0x1456: DM("0021456 <cfcn> ","LL Phase Stop"); break;
293          case 0x1560: DM("0021560 <cfcn> ","EL-Zoom On"); break;
294          case 0x1561: DM("0021561 <cfcn> ","EL-Zoom Off"); break;
295          case 0x1640: DM("0021640 <cfcn> ","Privacy Zone On"); break;
296          case 0x1641: DM("0021641 <cfcn> ","Privacy Zone Off"); break;
297          case 0x1690: DM("0021690 <cfcn> ","Motion Detect On"); break;
298          case 0x1691: DM("0021691 <cfcn> ","Motion Detect Off"); break;
299          case 0x16B0: DM("00216B0 <cfcn> ","D-Range On"); break;
300          case 0x16B1: DM("00216B1 <cfcn> ","D-Range Off"); break;
301           case 0x1760: DM("0021760 <cfcn> ","All Reset"); break;
302           case 0x1761: DM("0021761 <cfcn> ","Restart"); break;
303          case 0x1902: DM("0021902 <rfco> ","Preset Position Mode [ID all clear]"); break;
```

```
304          case 0x1940: DM("0021940 <cfcn> ","Setup Menu On"); break;
305          case 0x1941: DM("0021941 <cfcn> ","Setup Menu Off"); break;
306          case 0x1942: DM("0021942 <cfcn> ","Setup Menu [cursor up]"); break;
307          case 0x1943: DM("0021943 <cfcn> ","Setup Menu [cursor right]"); break;
308          case 0x1944: DM("0021944 <cfcn> ","Setup Menu [cursor down]"); break;
309          case 0x1945: DM("0021945 <cfcn> ","Setup Menu [cursor left]"); break;
310          case 0x194A: DM("002194A <cfcn> ","Setup Menu Set1"); break;
311          case 0x194B: DM("002194B <cfcn> ","Setup Menu Set2"); break;
312          case 0x194C: DM("002194C <cfcn> ","Left + Right Button Pressed (F2) OR <cfco> Set Up
 Menu SW (reset)"); break;
313          case 0x194D: DM("002194D <cfcn> ","Left + Centre + Right Button Pressed (F3) OR <cfco>
 Set Up Menu SW (all reset)"); break;
314          case 0x194E: DM("002194E <cfcn> ","Setup Special Menu"); break;
315          case 0x194F: DM("002194F <cfcn> ","Setup Menu [cursor stop]"); break;
316          case 0x19A0: DM("00219A0 <cfcn> ","Digital Flip On"); break;
317          case 0x19A1: DM("00219A1 <cfcn> ","Digital Flip Off"); break;
318          case 0x19C0: DM("00219C0 <cfcn> ","Menu On"); break;
319          case 0x19C1: DM("00219C1 <cfcn> ","Menu Off"); break;
320          case 0x19C8: DM("00219C8 <cfcn> ","Area Title 1 Set"); break;
321          case 0x19C9: DM("00219C9 <cfcn> ","Area Title 2 Set"); break;
322          case 0x19CA: DM("00219CA <cfcn> ","Area Title 3 Set"); break;
323          case 0x19CB: DM("00219CB <cfcn> ","Area Title 4 Set"); break;
324          case 0x19CC: DM("00219CC <cfcn> ","Area Title 5 Set"); break;
325          case 0x19CD: DM("00219CD <cfcn> ","Area Title 6 Set"); break;
326          case 0x19CE: DM("00219CE <cfcn> ","Area Title 7 Set"); break;
327          case 0x19CF: DM("00219CF <cfcn> ","Area Title 8 Set"); break;
328          case 0x19D0: DM("00219D0 <cfcn> ","Area Title Default On"); break;
329          case 0x19D1: DM("00219D1 <cfcn> ","Area Title Off"); break;
330          case 0x19E0: DM("00219E0 <cfcn> ","Area Title 1 Read Start"); break;
331          case 0x19E1: DM("00219E1 <cfcn> ","Area Title 1 Write Start"); break;
332          case 0x19E2: DM("00219E2 <cfcn> ","Area Title 2 Read Start"); break;
333          case 0x19E3: DM("00219E3 <cfcn> ","Area Title 2 Write Start"); break;
334          case 0x19E4: DM("00219E4 <cfcn> ","Area Title 3 Read Start"); break;
335          case 0x19E5: DM("00219E5 <cfcn> ","Area Title 3 Write Start"); break;
336          case 0x19E6: DM("00219E6 <cfcn> ","Area Title 4 Read Start"); break;
337          case 0x19E7: DM("00219E7 <cfcn> ","Area Title 4 Write Start"); break;
338          case 0x19E8: DM("00219E8 <cfcn> ","Area Title 5 Read Start"); break;
339          case 0x19E9: DM("00219E9 <cfcn> ","Area Title 5 Write Start"); break;
340          case 0x19EA: DM("00219EA <cfcn> ","Area Title 6 Read Start"); break;
341          case 0x19EB: DM("00219EB <cfcn> ","Area Title 6 Write Start"); break;
342          case 0x19EC: DM("00219EC <cfcn> ","Area Title 7 Read Start"); break;
343          case 0x19ED: DM("00219ED <cfcn> ","Area Title 7 Write Start"); break;
344          case 0x19EE: DM("00219EE <cfcn> ","Area Title 8 Read Start"); break;
345          case 0x19EF: DM("00219EF <cfcn> ","Area Title 8 Write Start"); break;
346          case 0x19F0: DM("00219F0 <cfcn> ","Function Request"); break;
347          case 0x1A06: DM("0021A06 <cfcn> ","One Shot Auto Focus On OR <cfco> Auto Focus On");
 break;
348          case 0x1A18: DM("0021A18 <cfcn> ","Auto Focus Area (small)"); break;
349          case 0x1A19: DM("0021A19 <cfcn> ","Auto Focus Area (middle)"); break;
350          case 0x1A1A: DM("0021A1A <cfcn> ","Auto Focus Area (large)"); break;
351
352          case 0x1020: DM("0021020 <csro> ","ALC/BLC off"); break;
```

```
353         case 0x1021: DM("0021021 <csro> ","ALC/BLC ALC"); break;
354         case 0x1022: DM("0021022 <csro> ","ALC/BLC ELC"); break;
355         case 0x1023: DM("0021023 <csro> ","ALC/BLC Manual"); break;
356         case 0x1110: DM("0021110 <csro> ","Shutter off"); break;
357         case 0x1111: DM("0021111 <csro> ","Shutter 1/100 NTSC only"); break;
358         case 0x1112: DM("0021112 <csro> ","Shutter 1/120 PAL only"); break;
359         case 0x1114: DM("0021114 <csro> ","Shutter 1/250"); break;
360         case 0x1115: DM("0021115 <csro> ","Shutter 1/500"); break;
361         case 0x1116: DM("0021116 <csro> ","Shutter 1/1000"); break;
362         case 0x1117: DM("0021117 <csro> ","Shutter 1/2000"); break;
363         case 0x1118: DM("0021118 <csro> ","Shutter 1/4000"); break;
364       //case 0x1119: DM("0021119 <csro> ","Shutter 1/10000"); break;
365       //case 0x1120: DM("0021120 <csro> ","Sens Up Auto (Set-up) off"); break;
366       //case 0x1121: DM("0021121 <csro> ","Sens Up Auto (Set-up) Auto x1"); break;
367       //case 0x1122: DM("0021122 <csro> ","Sens Up Auto (Set-up) Auto x2"); break;
368       //case 0x1123: DM("0021123 <csro> ","Sens Up Auto (Set-up) Auto x4"); break;
369         case 0x1124: DM("0021124 <csro> ","Sens Up Auto (Set-up) Auto x6"); break;
370         case 0x1126: DM("0021126 <csro> ","Sens Up Auto (Set-up) Auto x10"); break;
371         case 0x1129: DM("0021129 <csro> ","Sens Up Auto (Set-up) Auto x16"); break;
372         case 0x1133: DM("0021133 <csro> ","Sens Up Auto (Set-up) Auto x32"); break;
373         case 0x1140: DM("0021140 <csro> ","Sens Up Auto (Condition) off"); break;
374         case 0x1141: DM("0021141 <csro> ","Sens Up Auto (Condition) Auto x1"); break;
375         case 0x1142: DM("0021142 <csro> ","Sens Up Auto (Condition) Auto x2"); break;
376         case 0x1143: DM("0021143 <csro> ","Sens Up Auto (Condition) Auto x4"); break;
377         case 0x1144: DM("0021144 <csro> ","Sens Up Auto (Condition) Auto x6"); break;
378         case 0x1146: DM("0021146 <csro> ","Sens Up Auto (Condition) Auto x10"); break;
379         case 0x1149: DM("0021149 <csro> ","Sens Up Auto (Condition) Auto x16"); break;
380         case 0x114A: DM("002114A <csro> ","Sens Up Auto (Condition) Auto x32"); break;
381         case 0x1160: DM("0021160 <csro> ","Sens Up Manual (Set-up) off"); break;
382         case 0x1161: DM("0021161 <csro> ","Sens Up Manual (Set-up) Auto x1"); break;
383         case 0x1162: DM("0021162 <csro> ","Sens Up Manual (Set-up) Auto x2"); break;
384         case 0x1163: DM("0021163 <csro> ","Sens Up Manual (Set-up) Auto x4"); break;
385         case 0x1164: DM("0021164 <csro> ","Sens Up Manual (Set-up) Auto x6"); break;
386         case 0x1166: DM("0021166 <csro> ","Sens Up Manual (Set-up) Auto x10"); break;
387         case 0x1169: DM("0021169 <csro> ","Sens Up Manual (Set-up) Auto x16"); break;
388         case 0x116A: DM("002116A <csro> ","Sens Up Manual (Set-up) Auto x32"); break;
389         case 0x1210: DM("0021210 <csro> ","AGC off"); break;
390         case 0x1211: DM("0021211 <csro> ","AGC low"); break;
391         case 0x1212: DM("0021212 <csro> ","AGC mid"); break;
392         case 0x1213: DM("0021213 <csro> ","AGC high"); break;
393         case 0x1220: DM("0021220 <csro> ","Gain Up off"); break;
394         case 0x1221: DM("0021221 <csro> ","Gain Up low"); break;
395         case 0x1222: DM("0021222 <csro> ","Gain Up mid"); break;
396         case 0x1223: DM("0021223 <csro> ","Gain Up high"); break;
397         case 0x1241: DM("0021241 <csro> ","ELC auto"); break;
398         case 0x1242: DM("0021242 <csro> ","ELC reset"); break;
399         case 0x1243: DM("0021243 <csro> ","ELC normal"); break;
400       //case 0x1301: DM("0021301 <csro> ","White Balance ATW"); break;
401         case 0x1302: DM("0021302 <csro> ","White Balance AWC"); break;
402         case 0x1303: DM("0021303 <csro> ","White Balance Manual"); break;
403         case 0x1400: DM("0021400 <csro> ","Line Lock on"); break;
404         case 0x1401: DM("0021401 <csro> ","Line Lock off"); break;
```

```
405       //case 0x1760: DM("0021760 <csro> ","Motion Detect off"); break;
406       //case 0x1761: DM("0021761 <csro> ","Motion Detect on"); break;
407         case 0x1A20: DM("0021A20 <csro> ","Auto Focus Area small"); break;
408         case 0x1A21: DM("0021A21 <csro> ","Auto Focus Area middle"); break;
409         case 0x1A22: DM("0021A22 <csro> ","Auto Focus Area large"); break;
410
411         default:
412                 strcat(decodedcmnd,"002");
413                 strcat(decodedcmnd,yytext);
414                 DM(" <cfcn><cfco><csro><rfco> "," Unknown command");
415                 break;
416         }
417     donecmnd();
418 }
419
420
421 // This is used to decode a four hex digit command.
422 void decode202(void)
423 {
424     from[2] = yytext[0];
425     from[3] = yytext[1];
426     from[4] = yytext[2];
427     from[5] = yytext[3];
428     from[6] = '\0';
429
430     type = strtol(from,&end_P,16);
431     switch (type) {
432         case 0x0100: DM("2020100 <rsro> ","Wiper?"); break;
433         case 0x0102: DM("2020102 <rsro> ","Def?"); break;
434         case 0x0110: DM("2020110 <rsro> ","AUX 1/2?"); break;
435         case 0x0118: DM("2020118 <rsro> ","AUX 1/2 Mode?"); break;
436         case 0x0120: DM("2020120 <rsro> ","Camera Power?"); break;
437         case 0x0130: DM("2020130 <rsro> ","Random Pan?"); break;
438         case 0x0132: DM("2020132 <rsro> ","Auto Pan?"); break;
439         case 0x0136: DM("2020136 <rsro> ","Pan/Tilt Variable?"); break;
440         case 0x0138: DM("2020138 <rsro> ","Auto Pan Endless?"); break;
441         case 0x0146: DM("2020146 <rsro> ","Preset Mode?"); break;
442
443         case 0x1130: DM("2021130 <rfcn> ","AUX All On"); break;
444         case 0x1131: DM("2021131 <rfcn> ","AUX All Off"); break;
445         case 0x1160: DM("2021160 <rfcn> ","AUX 1 On"); break;
446         case 0x1161: DM("2021161 <rfcn> ","AUX 1 Off"); break;
447         case 0x1162: DM("2021162 <rfcn> ","AUX 2 On"); break;
448         case 0x1163: DM("2021163 <rfcn> ","AUX 2 Off"); break;
449          case 0x1180: DM("2021180 <rfcn> ","AUX 1 Momentary"); break;
450          case 0x1181: DM("2021181 <rfcn> ","AUX 1 Latch"); break;
451          case 0x1182: DM("2021182 <rfcn> ","AUX 2 Momentary"); break;
452          case 0x1183: DM("2021183 <rfcn> ","AUX 2 Latch"); break;
453         case 0x1224: DM("2021224 <rfcn> ","Zoom [stop(f)] + Focus [stop(f)]"); break;
454         case 0x1228: DM("2021228 <rfcn> ","Zoom [tele(f)]"); break;
455         case 0x1229: DM("2021229 <rfcn> ","Zoom [tele(f)] + Focus [far(f)]"); break;
456         case 0x122A: DM("202122A <rfcn> ","Focus [far(f)]"); break;
```

```
457         case 0x122B: DM("202122B <rfcn> ","Zoom [wide(f)] + Focus [far(f)]"); break;
458         case 0x122C: DM("202122C <rfcn> ","Zoom [wide(f)]"); break;
459         case 0x122D: DM("202122D <rfcn> ","Zoom [wide(f)] + Focus [near(f)]"); break;
460         case 0x122E: DM("202122E <rfcn> ","Focus [near(f)]"); break;
461         case 0x122F: DM("202122F <rfcn> ","Zoom [wide(f)] + Focus [near(f)]"); break;
462         case 0x1264: DM("2021264 <rfcn> ","Zoom [stop(v)] + Focus [stop(v)]"); break;
463         case 0x1268: DM("2021268 <rfcn> ","Zoom [tele(v)]"); break;
464         case 0x1269: DM("2021269 <rfcn> ","Zoom [tele(v)] + Focus [far(v)]"); break;
465         case 0x126A: DM("202126A <rfcn> ","Focus [far(v)]"); break;
466         case 0x126B: DM("202126B <rfcn> ","Zoom [wide(v)] + Focus [far(v)]"); break;
467         case 0x126C: DM("202126C <rfcn> ","Zoom [wide(v)]"); break;
468         case 0x126D: DM("202126D <rfcn> ","Zoom [wide(v)] + Focus [near(v)]"); break;
469         case 0x126E: DM("202126E <rfcn> ","Focus [near(v)]"); break;
470         case 0x126F: DM("202126F <rfcn> ","Zoom [wide(v)] + Focus [near(v)]"); break;
471
472          case 0x1300: DM("2021300 <rfcn> ","Auto Pan On"); break;
473          case 0x1301: DM("2021301 <rfcn> ","Auto Pan Off"); break;
474         case 0x1305: DM("2021305 <rfcn> ","Auto Pan Reverse"); break;
475         case 0x1306: DM("2021306 <rfcn> ","Auto Pan Left End Set"); break;
476         case 0x1307: DM("2021307 <rfcn> ","Auto Pan Right End Set"); break;
477         case 0x1308: DM("2021308 <rfcn> ","Auto Pan Speed Inc"); break;
478         case 0x1309: DM("2021309 <rfcn> ","Auto Pan Speed Dec"); break;
479         case 0x130E: DM("202130E <rfcn> ","Auto Pan End Less On"); break;
480         case 0x130F: DM("202130F <rfcn> ","Auto Pan End Less Off"); break;
481         case 0x1324: DM("2021324 <rfcn> ","Pan [stop(f)] + Tilt [stop(f)]"); break;
482         case 0x1328: DM("2021328 <rfcn> ","Pan [left(f)]"); break;
483         case 0x1329: DM("2021329 <rfcn> ","Pan [left(f)] + Tilt [up(f)]"); break;
484         case 0x132A: DM("202132A <rfcn> ","Tilt [up(f)]"); break;
485         case 0x132B: DM("202132B <rfcn> ","Pan [right(f)] + Tilt [up(f)]"); break;
486         case 0x132C: DM("202132C <rfcn> ","Pan [right(f)]"); break;
487         case 0x132D: DM("202132D <rfcn> ","Pan [right(f)] + Tilt [down(f)]"); break;
488         case 0x132E: DM("202132E <rfcn> ","Tilt [down(f)]"); break;
489         case 0x132F: DM("202132F <rfcn> ","Pan [left(f)] + Tilt [down(f)]"); break;
490         case 0x1334: DM("2021334 <rfcn> ","Pan [stop(f)] + Tilt [stop(s)]"); break;
491         case 0x1338: DM("2021338 <rfcn> ","Pan [left(f)]"); break;
492         case 0x1339: DM("2021339 <rfcn> ","Pan [left(f)] + Tilt [up(s)]"); break;
493         case 0x133A: DM("202133A <rfcn> ","Tilt [up(s)]"); break;
494         case 0x133B: DM("202133B <rfcn> ","Pan [right(f)] + Tilt [up(s)]"); break;
495         case 0x133C: DM("202133C <rfcn> ","Pan [right(f)]"); break;
496         case 0x133D: DM("202133D <rfcn> ","Pan [right(f)] + Tilt [down(s)]"); break;
497         case 0x133E: DM("202133E <rfcn> ","Tilt [down(s)]"); break;
498         case 0x133F: DM("202133F <rfcn> ","Pan [left(f)] + Tilt [down(s)]"); break;
499         case 0x1344: DM("2021344 <rfcn> ","Pan [stop(s)] + Tilt [stop(f)]"); break;
500         case 0x1348: DM("2021348 <rfcn> ","Pan [left(s)]"); break;
501         case 0x1349: DM("2021349 <rfcn> ","Pan [left(s)] + Tilt [up(f)]"); break;
502         case 0x134A: DM("202134A <rfcn> ","Tilt [up(f)]"); break;
503         case 0x134B: DM("202134B <rfcn> ","Pan [right(s)] + Tilt [up(f)]"); break;
504         case 0x134C: DM("202134C <rfcn> ","Pan [right(s)]"); break;
505         case 0x134D: DM("202134D <rfcn> ","Pan [right(s)] + Tilt [down(f)]"); break;
506         case 0x134E: DM("202134E <rfcn> ","Tilt [down(f)]"); break;
507         case 0x134F: DM("202134F <rfcn> ","Pan [left(s)] + Tilt [down(f)]"); break;
508         case 0x1354: DM("2021354 <rfcn> ","Pan [stop(s)] + Tilt [stop(s)]"); break;
```

```
509          case 0x1358: DM("2021358 <rfcn> ","Pan [left(s)]"); break;
510          case 0x1359: DM("2021359 <rfcn> ","Pan [left(s)] + Tilt [up(s)]"); break;
511          case 0x135A: DM("202135A <rfcn> ","Tilt [up(s)]"); break;
512          case 0x135B: DM("202135B <rfcn> ","Pan [right(s)] + Tilt [up(s)]"); break;
513          case 0x135C: DM("202135C <rfcn> ","Pan [right(s)]"); break;
514          case 0x135D: DM("202135D <rfcn> ","Pan [right(s)] + Tilt [down(s)]"); break;
515          case 0x135E: DM("202135E <rfcn> ","Tilt [down(s)]"); break;
516          case 0x135F: DM("202135F <rfcn> ","Pan [left(s)] + Tilt [down(s)]"); break;
517          case 0x1364: DM("2021364 <rfcn> ","Pan [stop(v)] + Tilt [stop(v)]"); break;
518          case 0x1368: DM("2021368 <rfcn> ","Pan [left(v)]"); break;
519          case 0x1369: DM("2021369 <rfcn> ","Pan [left(v)] + Tilt [up(v)]"); break;
520          case 0x136A: DM("202136A <rfcn> ","Tilt [up(v)]"); break;
521          case 0x136B: DM("202136B <rfcn> ","Pan [right(v)] + Tilt [up(v)]"); break;
522          case 0x136C: DM("202136C <rfcn> ","Pan [right(v)]"); break;
523          case 0x136D: DM("202136D <rfcn> ","Pan [right(v)] + Tilt [down(v)]"); break;
524          case 0x136E: DM("202136E <rfcn> ","Tilt [down(v)]"); break;
525          case 0x136F: DM("202136F <rfcn> ","Pan [left(v)] + Tilt [down(v)]"); break;
526          case 0x1390: DM("2021390 <rfcn> ","Auto Pan Key:Auto Pan"); break;
527          case 0x1391: DM("2021391 <rfcn> ","Auto Pan Key:Seq"); break;
528          case 0x1392: DM("2021392 <rfcn> ","Auto Pan Key:Sort"); break;
529          case 0x13A0: DM("20213A0 <rfcn> ","Propo Pan/Tilt On"); break;
530          case 0x13A1: DM("20213A1 <rfcn> ","Propo Pan/Tilt Off"); break;
531          case 0x13C0: DM("20213C0 <rfcn> ","Pan Limit On"); break;
532          case 0x13C1: DM("20213C1 <rfcn> ","Pan Limit Off"); break;
533          case 0x13D1: DM("20213D1 <rfcn> ","180 Deg Turn"); break;
534          case 0x1400: DM("2021400 <rfcn> ","Preset Position Call"); break;
535          case 0x1410: DM("2021410 <rfcn> ","Home Position Call"); break;
536          case 0x1431: DM("2021431 <rfcn> ","Preset Seq [off] OR <rfco> Preset Mode [off]");
 break;
537          case 0x1437: DM("2021437 <rfcn> ","Preset Mode [seq on]"); break;
538          case 0x1438: DM("2021438 <rfcn> ","Preset Mode [sort on]"); break;
539          case 0x1490: DM("2021490 <rfcn> ","Patrol Play"); break;
540          case 0x1491: DM("2021491 <rfcn> ","Patrol Stop"); break;
541          case 0x1494: DM("2021494 <rfcn> ","Patrol Learn"); break;
542          case 0x1540: DM("2021540 <rfcn> ","Direct Preposition Set Start OR <rfco> Preset
 Position Mode [start]"); break;
543          case 0x1542: DM("2021542 <rfcn> ","Direct Preposition Set OR <rfco> Preset Position
 Mode [set]"); break;
544          case 0x1543: DM("2021543 <rfcn> ","Direct Preposition Set End OR <rfco> Preset Position
 Mode [end]"); break;
545          case 0x1547: DM("2021547 <rfcn> ","Preset Position ID all clear OR <rfco> Preset
 Position Mode [ID all clear]"); break;
546          case 0x1600: DM("2021600 <rfcn> ","Alarm In 1 Off"); break;
547          case 0x1601: DM("2021601 <rfcn> ","Alarm In 1 Posi"); break;
548          case 0x1610: DM("2021610 <rfcn> ","Alarm In 2 Off"); break;
549          case 0x1611: DM("2021611 <rfcn> ","Alarm In 2 Posi"); break;
550          case 0x1620: DM("2021620 <rfcn> ","Alarm In 3 Off"); break;
551          case 0x1621: DM("2021621 <rfcn> ","Alarm In 3 Posi"); break;
552          case 0x1630: DM("2021630 <rfcn> ","Alarm In 4 Off"); break;
553          case 0x1631: DM("2021631 <rfcn> ","Alarm In 4 Posi"); break;
554          case 0x1632: DM("2021632 <rfcn> ","Alarm In 4 BW"); break;
555          case 0x1700: DM("2021700 <rfcn> ","Cont 1 Off"); break;
```

```
556         case 0x1701: DM("2021701 <rfcn> ","Cont 1 AUX 1"); break;
557         case 0x1702: DM("2021702 <rfcn> ","Cont 1 VMD"); break;
558         case 0x1710: DM("2021710 <rfcn> ","Cont 2 Off"); break;
559         case 0x1711: DM("2021711 <rfcn> ","Cont 2 AUX 2"); break;
560         case 0x1713: DM("2021713 <rfcn> ","Cont 2 BW"); break;
561
562         case 0x1000: DM("2021000 <rsro> ","Wiper On"); break;
563         case 0x1001: DM("2021001 <rsro> ","Wiper Off"); break;
564         case 0x1020: DM("2021020 <rsro> ","Def On"); break;
565         case 0x1021: DM("2021021 <rsro> ","Def Off"); break;
566         case 0x1100: DM("2021100 <rsro> ","AUX 1/2 AUX1:On/AUX2:On"); break;
567         case 0x1102: DM("2021102 <rsro> ","AUX 1/2 AUX1:Off/AUX2:On"); break;
568         case 0x1108: DM("2021108 <rsro> ","AUX 1/2 AUX1:On/AUX2:Off"); break;
569         case 0x110A: DM("202110A <rsro> ","AUX 1/2 AUX1:Off/AUX2:Off"); break;
570       //case 0x1180: DM("2021180 <rsro> ","AUX 1/2 Mode AUX1:Mometary/AUX2:Mometary"); break;
571       //case 0x1181: DM("2021181 <rsro> ","AUX 1/2 Mode AUX1:Latch/AUX2:Mometary"); break;
572       //case 0x1182: DM("2021182 <rsro> ","AUX 1/2 Mode AUX1:Mometary/AUX2:Latch"); break;
573       //case 0x1183: DM("2021183 <rsro> ","AUX 1/2 Mode AUX1:Latch/AUX2:Latch"); break;
574         case 0x1200: DM("2021200 <rsro> ","Camera Power On"); break;
575         case 0x1201: DM("2021201 <rsro> ","Camera Power Off"); break;
576       //case 0x1300: DM("2021300 <rsro> ","Random Pan On"); break;
577       //case 0x1301: DM("2021301 <rsro> ","Random Pan Off"); break;
578         case 0x1320: DM("2021320 <rsro> ","Auto Pan On"); break;
579         case 0x1321: DM("2021321 <rsro> ","Auto Pan Off"); break;
580         case 0x1360: DM("2021360 <rsro> ","Pan/Tilt Variable Available"); break;
581         case 0x1361: DM("2021361 <rsro> ","Pan/Tilt Variable Not Available"); break;
582         case 0x1380: DM("2021380 <rsro> ","Auto Pan Endless On"); break;
583         case 0x1381: DM("2021381 <rsro> ","Auto Pan Endless Off"); break;
584         case 0x1461: DM("2021461 <rsro> ","Preset Mode Off"); break;
585         case 0x1467: DM("2021467 <rsro> ","Preset Mode Seq"); break;
586         case 0x1468: DM("2021468 <rsro> ","Preset Mode Sort"); break;
587
588         default:
589                 strcat(decodedcmnd,"202");
590                 strcat(decodedcmnd,yytext);
591                 DM(" <rsro><rfco> "," Unknown command");
592                 break;
593         }
594     donecmnd();
595 }
596
597
598 // This is used to decode a four hex digit command.
599 void decode902(void)
600 {
601     DM("newnpptc 902","");
602 #ifdef DEBUGRUN
603     strcat(decodedcmnd,yytext);
604     strcat(decodedcmnd," ");
605 #endif // #ifdef DEBUGRUN
606
607     // Get zoom control
```

```
608     from[2] = yytext[0];
609     from[3] = '\0';
610     type = strtol(from,&end_P,16);
611     switch (type) {
612         case 0x0: strcat(decodedcmnd,"Tele zoom speed 1"); break;
613         case 0x1: strcat(decodedcmnd,"Tele zoom speed 2"); break;
614         case 0x2: strcat(decodedcmnd,"Tele zoom speed 3"); break;
615         case 0x3: strcat(decodedcmnd,"Tele zoom speed 0"); break;
616         case 0x4: strcat(decodedcmnd,"Wide zoom speed 1"); break;
617         case 0x5: strcat(decodedcmnd,"Wide zoom speed 2"); break;
618         case 0x6: strcat(decodedcmnd,"Wide zoom speed 3"); break;
619         case 0x7: strcat(decodedcmnd,"Wide zoom speed 0"); break;
620         case 0x8: strcat(decodedcmnd,"Zoom stop"); break;
621         case 0x9: strcat(decodedcmnd,"Zoom stop 1"); break;
622         case 0xA: strcat(decodedcmnd,"Zoom stop 2"); break;
623         case 0xB: strcat(decodedcmnd,"Zoom stop 3"); break;
624         case 0xC: strcat(decodedcmnd,"Zoom stop 4"); break;
625         case 0xD: strcat(decodedcmnd,"Zoom stop 5"); break;
626         case 0xE: strcat(decodedcmnd,"Zoom stop 6"); break;
627         case 0xF: strcat(decodedcmnd,"Zoom stop 7"); break;
628     }
629     strcat(decodedcmnd,", ");
630
631
632     // Now do direction
633     from[2] = yytext[1];
634     from[3] = '\0';
635     type = strtol(from,&end_P,16);
636     switch (type) {
637         case 0x0: strcat(decodedcmnd,"Null"); break;
638         case 0x1: strcat(decodedcmnd,"Motion stop"); break;
639         case 0x2: strcat(decodedcmnd,"Illegal motion"); break;
640         case 0x3: strcat(decodedcmnd,"Illegal motion"); break;
641         case 0x4: strcat(decodedcmnd,"Stop"); break;
642         case 0x5: strcat(decodedcmnd,"Illegal motion"); break;
643         case 0x6: strcat(decodedcmnd,"Illegal motion"); break;
644         case 0x7: strcat(decodedcmnd,"Illegal motion"); break;
645         case 0x8: strcat(decodedcmnd,"Left"); break;
646         case 0x9: strcat(decodedcmnd,"Left + Up"); break;
647         case 0xA: strcat(decodedcmnd,"Up"); break;
648         case 0xB: strcat(decodedcmnd,"Right + Up"); break;
649         case 0xC: strcat(decodedcmnd,"Right"); break;
650         case 0xD: strcat(decodedcmnd,"Right + Down"); break;
651         case 0xE: strcat(decodedcmnd,"Down"); break;
652         case 0xF: strcat(decodedcmnd,"Left + Down"); break;
653     }
654     strcat(decodedcmnd,", ");
655
656     // And now for the speed, first pan
657     strcat(decodedcmnd,"Pan = ");
658     from[2] = yytext[2];
659     from[3] = '\0';
```

```
660     type = strtol(from,&end_P,16);
661     switch (type) {
662         case 0x0: strcat(decodedcmnd,"0.10"); break;
663         case 0x1: strcat(decodedcmnd,"0.21"); break;
664         case 0x2: strcat(decodedcmnd,"0.47"); break;
665         case 0x3: strcat(decodedcmnd,"1.00"); break;
666         case 0x4: strcat(decodedcmnd,"3.19"); break;
667         case 0x5: strcat(decodedcmnd,"7.06"); break;
668         case 0x6: strcat(decodedcmnd,"12.01"); break;
669         case 0x7: strcat(decodedcmnd,"18.27"); break;
670         case 0x8: strcat(decodedcmnd,"26.54"); break;
671         case 0x9: strcat(decodedcmnd,"35.55"); break;
672         case 0xA: strcat(decodedcmnd,"44.97"); break;
673         case 0xB: strcat(decodedcmnd,"57.52"); break;
674         case 0xC: strcat(decodedcmnd,"71.63"); break;
675         case 0xD: strcat(decodedcmnd,"86.02"); break;
676         case 0xE: strcat(decodedcmnd,"103.10"); break;
677         case 0xF: strcat(decodedcmnd,"120.24"); break;
678     }
679     strcat(decodedcmnd," deg/sec, ");
680
681     // And now for the speed, then tilt
682     strcat(decodedcmnd,"Tilt = ");
683     from[2] = yytext[3];
684     from[3] = '\0';
685     type = strtol(from,&end_P,16);
686     switch (type) {
687         case 0x0: strcat(decodedcmnd,"0.10"); break;
688         case 0x1: strcat(decodedcmnd,"0.21"); break;
689         case 0x2: strcat(decodedcmnd,"0.47"); break;
690         case 0x3: strcat(decodedcmnd,"1.00"); break;
691         case 0x4: strcat(decodedcmnd,"3.19"); break;
692         case 0x5: strcat(decodedcmnd,"7.06"); break;
693         case 0x6: strcat(decodedcmnd,"12.01"); break;
694         case 0x7: strcat(decodedcmnd,"18.27"); break;
695         case 0x8: strcat(decodedcmnd,"26.54"); break;
696         case 0x9: strcat(decodedcmnd,"35.55"); break;
697         case 0xA: strcat(decodedcmnd,"44.97"); break;
698         case 0xB: strcat(decodedcmnd,"57.52"); break;
699         case 0xC: strcat(decodedcmnd,"71.63"); break;
700         case 0xD: strcat(decodedcmnd,"86.02"); break;
701         case 0xE: strcat(decodedcmnd,"103.10"); break;
702         case 0xF: strcat(decodedcmnd,"120.24"); break;
703     }
704     strcat(decodedcmnd," deg/sec");
705     donecmnd();
706 }
707
708 // This is used to decode a four hex digit command.
709 void decode903(void)
710 {
711 char stemp[10];
```

```
712
713     from[2] = yytext[0];
714     from[3] = yytext[1];
715     from[4] = yytext[2];
716     from[5] = yytext[3];
717     from[6] = '\0';
718
719     type = strtol(from,&end_P,16);
720
721 #ifdef DEBUGRUN
722     sprintf(stemp,"'%2d' ",((type % 256)+1));
723     strcat(decodedcmnd,stemp);
724 #endif // #ifdef DEBUGRUN
725
726     switch (type) {
727         case 0x0000: DM("9030000 <nc> ","Preset 1 Call"); break;
728         case 0x0001: DM("9030001 <nc> ","Preset 2 Call"); break;
729         case 0x0002: DM("9030002 <nc> ","Preset 3 Call"); break;
730         case 0x0003: DM("9030003 <nc> ","Preset 4 Call"); break;
731         case 0x0004: DM("9030004 <nc> ","Preset 5 Call"); break;
732         case 0x0005: DM("9030005 <nc> ","Preset 6 Call"); break;
733         case 0x0006: DM("9030006 <nc> ","Preset 7 Call"); break;
734         case 0x0007: DM("9030007 <nc> ","Preset 8 Call"); break;
735         case 0x0008: DM("9030008 <nc> ","Preset 9 Call"); break;
736         case 0x0009: DM("9030009 <nc> ","Preset 10 Call"); break;
737         case 0x000A: DM("903000A <nc> ","Preset 11 Call"); break;
738         case 0x000B: DM("903000B <nc> ","Preset 12 Call"); break;
739         case 0x000C: DM("903000C <nc> ","Preset 13 Call"); break;
740         case 0x000D: DM("903000D <nc> ","Preset 14 Call"); break;
741         case 0x000E: DM("903000E <nc> ","Preset 15 Call"); break;
742         case 0x000F: DM("903000F <nc> ","Preset 16 Call"); break;
743         case 0x0010: DM("9030010 <nc> ","Preset 17 Call"); break;
744         case 0x0011: DM("9030011 <nc> ","Preset 18 Call"); break;
745         case 0x0012: DM("9030012 <nc> ","Preset 19 Call"); break;
746         case 0x0013: DM("9030013 <nc> ","Preset 20 Call"); break;
747         case 0x0014: DM("9030014 <nc> ","Preset 21 Call"); break;
748         case 0x0015: DM("9030015 <nc> ","Preset 22 Call"); break;
749         case 0x0016: DM("9030016 <nc> ","Preset 23 Call"); break;
750         case 0x0017: DM("9030017 <nc> ","Preset 24 Call"); break;
751         case 0x0018: DM("9030018 <nc> ","Preset 25 Call"); break;
752         case 0x0019: DM("9030019 <nc> ","Preset 26 Call"); break;
753         case 0x001A: DM("903001A <nc> ","Preset 27 Call"); break;
754         case 0x001B: DM("903001B <nc> ","Preset 28 Call"); break;
755         case 0x001C: DM("903001C <nc> ","Preset 29 Call"); break;
756         case 0x001D: DM("903001D <nc> ","Preset 30 Call"); break;
757         case 0x001E: DM("903001E <nc> ","Preset 31 Call"); break;
758         case 0x001F: DM("903001F <nc> ","Preset 32 Call"); break;
759         case 0x0020: DM("9030020 <nc> ","Preset 33 Call"); break;
760         case 0x0021: DM("9030021 <nc> ","Preset 34 Call"); break;
761         case 0x0022: DM("9030022 <nc> ","Preset 35 Call"); break;
762         case 0x0023: DM("9030023 <nc> ","Preset 36 Call"); break;
763         case 0x0024: DM("9030024 <nc> ","Preset 37 Call"); break;
```

```
764         case 0x0025: DM("9030025 <nc> ","Preset 38 Call"); break;
765         case 0x0026: DM("9030026 <nc> ","Preset 39 Call"); break;
766         case 0x0027: DM("9030027 <nc> ","Preset 40 Call"); break;
767         case 0x0028: DM("9030028 <nc> ","Preset 41 Call"); break;
768         case 0x0029: DM("9030029 <nc> ","Preset 42 Call"); break;
769         case 0x002A: DM("903002A <nc> ","Preset 43 Call"); break;
770         case 0x002B: DM("903002B <nc> ","Preset 44 Call"); break;
771         case 0x002C: DM("903002C <nc> ","Preset 45 Call"); break;
772         case 0x002D: DM("903002D <nc> ","Preset 46 Call"); break;
773         case 0x002E: DM("903002E <nc> ","Preset 47 Call"); break;
774         case 0x002F: DM("903002F <nc> ","Preset 48 Call"); break;
775         case 0x0030: DM("9030030 <nc> ","Preset 49 Call"); break;
776         case 0x0031: DM("9030031 <nc> ","Preset 50 Call"); break;
777         case 0x0032: DM("9030032 <nc> ","Preset 51 Call"); break;
778         case 0x0033: DM("9030033 <nc> ","Preset 52 Call"); break;
779         case 0x0034: DM("9030034 <nc> ","Preset 53 Call"); break;
780         case 0x0035: DM("9030035 <nc> ","Preset 54 Call"); break;
781         case 0x0036: DM("9030036 <nc> ","Preset 55 Call"); break;
782         case 0x0037: DM("9030037 <nc> ","Preset 56 Call"); break;
783         case 0x0038: DM("9030038 <nc> ","Preset 57 Call"); break;
784         case 0x0039: DM("9030039 <nc> ","Preset 58 Call"); break;
785         case 0x003A: DM("903003A <nc> ","Preset 59 Call"); break;
786         case 0x003B: DM("903003B <nc> ","Preset 60 Call"); break;
787         case 0x003C: DM("903003C <nc> ","Preset 61 Call"); break;
788         case 0x003D: DM("903003D <nc> ","Preset 62 Call"); break;
789         case 0x003E: DM("903003E <nc> ","Preset 63 Call"); break;
790         case 0x003F: DM("903003F <nc> ","Preset 64 Call"); break;
791         case 0x0040: DM("9030040 <nc> ","Auto - Pan On"); break;
792         case 0x0041: DM("9030041 <nc> ","Auto - Pan Off"); break;
793         case 0x0042: DM("9030042 <nc> ","Auto - Pan Speed Inc"); break;
794         case 0x0043: DM("9030043 <nc> ","Auto - Pan Speed Dec"); break;
795         case 0x0044: DM("9030044 <nc> ","Auto - Pan Start Point Set"); break;
796         case 0x0045: DM("9030045 <nc> ","Auto - Pan End Point Set"); break;
797         case 0x0046: DM("9030046 <nc> ","Auto - Mode Off"); break;
798         case 0x0047: DM("9030047 <nc> ","Auto - Seq On"); break;
799         case 0x0048: DM("9030048 <nc> ","Auto - Sort On"); break;
800         case 0x0049: DM("9030049 <nc> ","Auto - Pan Sweep Area Invert"); break;
801         case 0x004A: DM("903004A <nc> ","Unknown #3"); break;
802         case 0x004B: DM("903004B <nc> ","Endless On"); break;
803         case 0x004C: DM("903004C <nc> ","Endless Off"); break;
804         case 0x004D: DM("903004D <nc> ","Digital Flip On"); break;
805         case 0x004E: DM("903004E <nc> ","Digital Flip Off"); break;
806         case 0x004F: DM("903004F <nc> ","Proportional P/T On"); break;
807         case 0x0050: DM("9030050 <nc> ","Proportional P/T Off"); break;
808         case 0x0051: DM("9030051 <nc> ","Unknown #1"); break;
809         case 0x0052: DM("9030052 <nc> ","Unknown #2"); break;
810         case 0x0053: DM("9030053 <nc> ","Super-DII On"); break;
811         case 0x0054: DM("9030054 <nc> ","Super-DII Off"); break;
812         case 0x0055: DM("9030055 <nc> ","Stop AF On - Menu Item -"); break;
813         case 0x0056: DM("9030056 <nc> ","Stop AF Off - Menu Item -"); break;
814         case 0x0057: DM("9030057 <nc> ","1 Shot AF On (start)"); break;
815         case 0x0058: DM("9030058 <nc> ","Home Position Move"); break;
```

```
816         case 0x0059: DM("9030059 <nc> ","Home Position On - Menu Item -"); break;
817         case 0x005A: DM("903005A <nc> ","Home Position Off - Menu Item -"); break;
818         case 0x005B: DM("903005B <nc> ","Home Position Auto - Menu Item -"); break;
819         case 0x005C: DM("903005C <nc> ","Camera ID On"); break;
820         case 0x005D: DM("903005D <nc> ","Camera ID Off"); break;
821         case 0x005E: DM("903005E <nc> ","Area Title Nesw On"); break;
822         case 0x005F: DM("903005F <nc> ","Area Title User On"); break;
823         case 0x0060: DM("9030060 <nc> ","Area Title Off"); break;
824         case 0x0061: DM("9030061 <nc> ","EL-Zoom On"); break;
825         case 0x0062: DM("9030062 <nc> ","EL-Zoom Off"); break;
826         case 0x0063: DM("9030063 <nc> ","Refresh"); break;
827         case 0x0064: DM("9030064 <nc> ","Preset 1 Set"); break;
828         case 0x0065: DM("9030065 <nc> ","Preset 2 Set"); break;
829         case 0x0066: DM("9030066 <nc> ","Preset 3 Set"); break;
830         case 0x0067: DM("9030067 <nc> ","Preset 4 Set"); break;
831         case 0x0068: DM("9030068 <nc> ","Preset 5 Set"); break;
832         case 0x0069: DM("9030069 <nc> ","Preset 6 Set"); break;
833         case 0x006A: DM("903006A <nc> ","Preset 7 Set"); break;
834         case 0x006B: DM("903006B <nc> ","Preset 8 Set"); break;
835         case 0x006C: DM("903006C <nc> ","Preset 9 Set"); break;
836         case 0x006D: DM("903006D <nc> ","Preset 10 Set"); break;
837         case 0x006E: DM("903006E <nc> ","Preset 11 Set"); break;
838         case 0x006F: DM("903006F <nc> ","Preset 12 Set"); break;
839         case 0x0070: DM("9030070 <nc> ","Preset 13 Set"); break;
840         case 0x0071: DM("9030071 <nc> ","Preset 14 Set"); break;
841         case 0x0072: DM("9030072 <nc> ","Preset 15 Set"); break;
842         case 0x0073: DM("9030073 <nc> ","Preset 16 Set"); break;
843         case 0x0074: DM("9030074 <nc> ","Preset 17 Set"); break;
844         case 0x0075: DM("9030075 <nc> ","Preset 18 Set"); break;
845         case 0x0076: DM("9030076 <nc> ","Preset 19 Set"); break;
846         case 0x0077: DM("9030077 <nc> ","Preset 20 Set"); break;
847         case 0x0078: DM("9030078 <nc> ","Preset 21 Set"); break;
848         case 0x0079: DM("9030079 <nc> ","Preset 22 Set"); break;
849         case 0x007A: DM("903007A <nc> ","Preset 23 Set"); break;
850         case 0x007B: DM("903007B <nc> ","Preset 24 Set"); break;
851         case 0x007C: DM("903007C <nc> ","Preset 25 Set"); break;
852         case 0x007D: DM("903007D <nc> ","Preset 26 Set"); break;
853         case 0x007E: DM("903007E <nc> ","Preset 27 Set"); break;
854         case 0x007F: DM("903007F <nc> ","Preset 28 Set"); break;
855         case 0x0080: DM("9030080 <nc> ","Preset 29 Set"); break;
856         case 0x0081: DM("9030081 <nc> ","Preset 30 Set"); break;
857         case 0x0082: DM("9030082 <nc> ","Preset 31 Set"); break;
858         case 0x0083: DM("9030083 <nc> ","Preset 32 Set"); break;
859         case 0x0084: DM("9030084 <nc> ","Preset 33 Set"); break;
860         case 0x0085: DM("9030085 <nc> ","Preset 34 Set"); break;
861         case 0x0086: DM("9030086 <nc> ","Preset 35 Set"); break;
862         case 0x0087: DM("9030087 <nc> ","Preset 36 Set"); break;
863         case 0x0088: DM("9030088 <nc> ","Preset 37 Set"); break;
864         case 0x0089: DM("9030089 <nc> ","Preset 38 Set"); break;
865         case 0x008A: DM("903008A <nc> ","Preset 39 Set"); break;
866         case 0x008B: DM("903008B <nc> ","Preset 40 Set"); break;
867         case 0x008C: DM("903008C <nc> ","Preset 41 Set"); break;
```

```
868         case 0x008D: DM("903008D <nc> ","Preset 42 Set"); break;
869         case 0x008E: DM("903008E <nc> ","Preset 43 Set"); break;
870         case 0x008F: DM("903008F <nc> ","Preset 44 Set"); break;
871         case 0x0090: DM("9030090 <nc> ","Preset 45 Set"); break;
872         case 0x0091: DM("9030091 <nc> ","Preset 46 Set"); break;
873         case 0x0092: DM("9030092 <nc> ","Preset 47 Set"); break;
874         case 0x0093: DM("9030093 <nc> ","Preset 48 Set"); break;
875         case 0x0094: DM("9030094 <nc> ","Preset 49 Set"); break;
876         case 0x0095: DM("9030095 <nc> ","Preset 50 Set"); break;
877         case 0x0096: DM("9030096 <nc> ","Preset 51 Set"); break;
878         case 0x0097: DM("9030097 <nc> ","Preset 52 Set"); break;
879         case 0x0098: DM("9030098 <nc> ","Preset 53 Set"); break;
880         case 0x0099: DM("9030099 <nc> ","Preset 54 Set"); break;
881         case 0x009A: DM("903009A <nc> ","Preset 55 Set"); break;
882         case 0x009B: DM("903009B <nc> ","Preset 56 Set"); break;
883         case 0x009C: DM("903009C <nc> ","Preset 57 Set"); break;
884         case 0x009D: DM("903009D <nc> ","Preset 58 Set"); break;
885         case 0x009E: DM("903009E <nc> ","Preset 59 Set"); break;
886         case 0x009F: DM("903009F <nc> ","Preset 60 Set"); break;
887         case 0x00A0: DM("90300A0 <nc> ","Preset 61 Set"); break;
888         case 0x00A1: DM("90300A1 <nc> ","Preset 62 Set"); break;
889         case 0x00A2: DM("90300A2 <nc> ","Preset 63 Set"); break;
890         case 0x00A3: DM("90300A3 <nc> ","Preset 64 Set"); break;
891         case 0x00A4: DM("90300A4 <nc> ","Patrol - Play"); break;
892         case 0x00A5: DM("90300A5 <nc> ","Patrol - Stop"); break;
893         case 0x00A6: DM("90300A6 <nc> ","Patrol - Learn"); break;
894         case 0x00A8: DM("90300A8 <nc> ","Iris Open"); break;
895         case 0x00A9: DM("90300A9 <nc> ","Iris Close"); break;
896         case 0x00AA: DM("90300AA <nc> ","Shutter On"); break;
897         case 0x00AB: DM("90300AB <nc> ","Shutter Off"); break;
898         case 0x00AC: DM("90300AC <nc> ","Shutter Inc"); break;
899         case 0x00AD: DM("90300AD <nc> ","Shutter Dec"); break;
900         case 0x00AE: DM("90300AE <nc> ","AGC On"); break;
901         case 0x00AF: DM("90300AF <nc> ","AGC Off"); break;
902         case 0x00B0: DM("90300B0 <nc> ","Sens Up Fix - On"); break;
903         case 0x00B1: DM("90300B1 <nc> ","Sens Up Fix - Off"); break;
904         case 0x00B2: DM("90300B2 <nc> ","Sens Up Fix - Inc"); break;
905         case 0x00B3: DM("90300B3 <nc> ","Sens Up Fix - Dec"); break;
906         case 0x00B4: DM("90300B4 <nc> ","Sens Up Auto On"); break;
907         case 0x00B5: DM("90300B5 <nc> ","Sens Up Auto Off"); break;
908         case 0x00B6: DM("90300B6 <nc> ","Sens Up Auto Inc"); break;
909         case 0x00B7: DM("90300B7 <nc> ","Sens Up Auto Dec"); break;
910         case 0x00B8: DM("90300B8 <nc> ","LL Phase Inc"); break;
911         case 0x00B9: DM("90300B9 <nc> ","LL Phase Dec"); break;
912         case 0x00BA: DM("90300BA <nc> ","180 Deg Pan Turn"); break;
913         case 0x00BB: DM("90300BB <nc> ","Cleaning On (menu)"); break;
914         case 0x00BC: DM("90300BC <nc> ","Cleaning Off (menu)"); break;
915
916         case 0x4000: DM("9034000 <newnpfc> ","Focus Control: Near speed 0"); break;
917         case 0x4001: DM("9034001 <newnpfc> ","Focus Control: Near speed 1"); break;
918         case 0x4002: DM("9034002 <newnpfc> ","Focus Control: Near speed 2"); break;
919         case 0x4003: DM("9034003 <newnpfc> ","Focus Control: Near speed 3"); break;
```

```
920         case 0x4004: DM("9034004 <newnpfc> ","Focus Control: Far speed 0"); break;
921         case 0x4005: DM("9034005 <newnpfc> ","Focus Control: Far speed 1"); break;
922         case 0x4006: DM("9034006 <newnpfc> ","Focus Control: Far speed 2"); break;
923         case 0x4007: DM("9034007 <newnpfc> ","Focus Control: Far speed 3"); break;
924         case 0x4008: DM("9034008 <newnpfc> ","Focus Control: Stop"); break;
925         case 0x4009: DM("9034009 <newnpfc> ","Focus Control: Stop"); break;
926         case 0x400A: DM("903400A <newnpfc> ","Focus Control: Stop"); break;
927         case 0x400B: DM("903400B <newnpfc> ","Focus Control: Stop"); break;
928         case 0x400C: DM("903400C <newnpfc> ","Focus Control: Stop"); break;
929         case 0x400D: DM("903400D <newnpfc> ","Focus Control: Stop"); break;
930         case 0x400E: DM("903400E <newnpfc> ","Focus Control: Stop"); break;
931         case 0x400F: DM("903400F <newnpfc> ","Focus Control: Stop"); break;
932
933         default:
934                 strcat(decodedcmnd,"903");
935                 strcat(decodedcmnd,yytext);
936                 DM(" <nc> "," Unknown command");
937                 break;
938         }
939     donecmnd();
940 }
941
942
943 // Used to complete the decoding of a command by printing it and
944 // resetting pointers
945 void donecmnd(void)
946 {
947     fprintf(yyout,"%s\n",decodedcmnd);
948     decodedcmnd[0] = '\0';
949     linecount++;
950 //    if ((linecount % 10) == 0)
951 //        fprintf(yyout,"\n");
952 }
953
954
955 // yywrap is for End Of File processing, 1 = done
956 int yywrap(void)
957 {
958         return 1;
959 }
960
961 main(int argc, char *argv[])
962 {
963 char infilename[100];
964 char outfilename[100];
965
966 time_t thistime;
967 struct tm *t;
968
969         fprintf(stderr,"$Id: GcProc.l,v 1.7 2004-11-30 15:07:14-08 Hamilton Exp Hamilton $\n");
970
971         strcpy(infilename,"stdin");
```

```
 972         strcpy(outfilename,"stdout");
 973
 974         // Do we have to deal with stdin/stdout?
 975         // 3 = one for the program name and one each for input and output names
 976         if (argc == 3) {
 977             // Nope
 978             // Get over the program, etc., name
 979             argc--;
 980             argv++;
 981             yyin = fopen(argv[0],"r");
 982             if (yyin == NULL) {
 983                 fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
 984                                 argv[0]);
 985                 exit(EXIT_FAILURE);
 986             }
 987             strcpy(outfilename,argv[0]);
 988
 989             // Now get over the input file name and get to the output file name
 990             argc--;
 991             argv++;
 992             yyout = fopen(argv[0],"w");
 993             if (yyout == NULL) {
 994                 fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
 995                                 argv[0]);
 996                 exit(EXIT_FAILURE);
 997             }
 998             strcpy(infilename,argv[0]);
 999         }
1000         else {
1001             // Woops we are using std... type io
1002             yyin  = stdin;
1003             yyout = stdout;
1004         }
1005
1006         fprintf(yyout,"%%\n%% %s   %5d $Id: GcProc.l,v 1.7 2004-11-30 15:07:14-08 Hamilton Exp
 Hamilton $\n",
1007                         __FILE__,__LINE__);
1008         t = time(NULL);
1009         thistime = localtime(&t);
1010         fprintf(yyout,"%% %s   %5d %s",
1011                         __FILE__,__LINE__,asctime(thistime));
1012         fprintf(yyout,"%% %s   %5d Reading from \"%s\"\n",
1013                         __FILE__,__LINE__,infilename);
1014         fprintf(yyout,"%% %s   %5d Writing into \"%s\"\n",
1015                         __FILE__,__LINE__,outfilename);
1016         fprintf(yyout,"%% %s   %5d  %\n",__FILE__,__LINE__);
1017         fprintf(yyout,"%% %s   %5d  %s\n",
1018                         __FILE__,__LINE__,"' --->' = DTE, from keyboard");
1019         fprintf(yyout,"%% %s   %5d  %s\n",
1020                         __FILE__,__LINE__,"'<--- ' = DCE, from camera");
1021         fprintf(yyout,"%% %s   %5d  %\n",__FILE__,__LINE__);
1022
```

```
1023          from[0] = '0';         // For converting hex values
1024          from[1] = 'x';         // . . from ASCII
1025          decodedcmnd[0] = '\0'; // We do a lot of concatenating here
1026
1027          yylex();
1028          exit(EXIT_SUCCESS);
1029  }
1030
1031  // $Log: GcProc.l,v $
1032  // Revision 1.7  2004-11-30 15:07:14-08  Hamilton
1033  // Normal end of day data saving
1034  //
1035  // Revision 1.6  2004-11-30 13:06:11-08  Hamilton
1036  // Normal end of day data saving
1037  //
1038  // Revision 1.5  2004-11-18 13:23:54-08  Hamilton
1039  // Normal end of day data saving
1040  //
1041  // Revision 1.4  2004-11-17 15:54:55-08  Hamilton
1042  // Normal end of day data saving
1043  //
1044  // Revision 1.3  2004-11-15 15:31:41-08  Hamilton
1045  // Normal end of day data saving
1046  //
1047  // Revision 1.2  2004-11-12 14:04:47-08  Hamilton
1048  // Normal end of day data saving
1049  //
1050  // Revision 1.1  2004-11-12 07:10:04-08  Hamilton
1051  // Initial revision
1052  //
```

APPENDIX A

A 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

[2] `$Header: d:/txb-p/Captures/RCS/BreakSav.inc,v 1.1 2004-11-04 12:10:27-08 Hamilton Exp Hamilton $`

4.1.1.1. Select FILE (Alt-F):

4.1.1.2. While in FILE select SAVE AS

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 analyzation.

APPENDIX B

B DB

B.1 Description

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

B.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 like this:

```
 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
```

B.3 Input Data Format

The file format of a breakout file is detailed in the BREAKOUT documentation. It is in binary and a partial 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
```

B.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,  163: $ Id: db.c,v 1.17 2002-07-12 08:13:19-07 Hamilton Exp Hamilton $
%
%       db.c,  166: Dumping "breakout\19nov04a.cap" a 88888 byte file.
%
%       db.c,  176: The .CAP file's header is:
%       db.c,  178: "Saved Fri Nov 19 12:56:03 2004
%
%       db.c,  187: Format of this output is as follows:
%       db.c,  188: There are two directions of data flow.
%       db.c,  189: Data direction is indicated with either
%       db.c,  190: a "1" or "0" followed by a "-".
%       db.c,  191: It is intended that "1" will
%       db.c,  192: be used to indicate data that is coming in,
%       db.c,  193: and that "0" is used for data
%       db.c,  194: going out. If this is not true, then use
%       db.c,  195: the /R option to reverse the logic
%       db.c,  196: of these indicators
%       db.c,  197: Following the "-" is the value of the
%       db.c,  198: data byte in hexidecimal.
%       db.c,  199: Whenever a direction change is detected
%       db.c,  200: a new line is output.
%       db.c,  202: Data source directions are reversed.
%       db.c,  205: Each direction-data item is preceeded
%       db.c,  206: 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,  224: There were 22213 data bytes converted
%       db.c,  226: There were 1412 lines output
%
```

B.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
```

B.6 Listing

```
 1  // $Header: d:/txb-p/Captures/RCS/db.c,v 1.17 2002-07-12 08:13:19-07 Hamilton Exp Hamilton $
 2  // Copyright (c) by Pelco, 1998, 1999, 2001
 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  #include <ctype.h>
13  #include <stdio.h>
14  #include <string.h>
15  #include <time.h>
16
17  #define FROM         0           // First data direction
18  #define TO           1           // Other data direction
19
20  #define NO           0           // No
21  #define YES          1           // Yes
22
23  void    DumpFrame(void);         // Interpertes a Breakout data frame
24  long    Flength(char *fname);    // Returns the length of a file
25  int     GetDirection(void);      // What is the current data direction
26  void    PrintHelp(void);         // Prints out a help file for bad argruments
27
28  // Defines for use with breakout's .CAP file
29  #define SIGNALS     0            // State of the RS-232 control lines
30  #define DTR         0x01         // Data Terminal Ready bit
31  #define RTS         0x02         // Request To Send bit
32  #define SB          0x04         // Set Break bit
33  #define BI          0x08         // Break Interrupt bit
34  #define CTS         0x10         // Clear To Send bit
35  #define DSR         0x20         // Data Set Ready bit
36  #define RI          0x40         // Ring Indicator bit
37  #define DCD         0x80         // Data Carrier Detect bit
38
39  #define SOURCE      1            // Direction/status of the data
40  #define DTE         0x01         // DTE data event bit
41  #define OER         0x02         // Overflow error bit
42  #define PER         0x04         // Parity error bit
43  #define FER         0x08         // Frame error bit
44  #define TSL         0x10         // Long Time Stamp Enabled bit
45  #define TSE         0x20         // Time Stamp Enabled bit
46  #define VE          0x40         // Valid Event bit
47  #define DP          0x80         // Data Present bit
48  #define DATA_OK     (VE|DP)      // Indicates that the data byte is valid
49  #define TIME_STAMP  (VE|TSE|TSL) // Might indicate a valid time stamp field
50
51  #define TS          2            // Time stamp
```

```
52  #define DATA        3           // The data for this frame
53
54  // For long time stamp records the following holds:
55  //
56  // SIGNALS Same as for data records
57  // SOURCE  Top 4 bits identify a time record, bottom 4 bits are the
58  //         4 MSBs of the time tag
59  // TS      Same as for data records, i.e. all 8 bits are the 8 LSBs of the
60  //         time tag
61  // DATA    Middle 8 bits of the time tag
62  //
63  // Thus we have:
64  //
65  // TimeTag = SOURCE[3,2,1,0] + DATA[7,6,5,4,3,2,1,0] + TS[7,6,5,4,3,2,1,0]
66  //           [bits involved]
67
68  long          ByteCounter = 0;          // Number of data bytes dumped
69  unsigned int  DataFrame[4];             // Contents of a Breakout data frame
70  long          FileLength;               // Input file length, in bytes
71  long          InputByte = 0;            // Current data frame being worked on
72  char         *InputPointer = NULL;      // Input file name from the cmnd line
73  FILE         *InFile;                   // Pointer to the input file
74  long          LineCounter = 23;         // How many lines have been output
75                                          // 23 = 18 lines of header and
76                                          //       5 lines of trailer
77  int           MsgTime;                  // This message's time
78  int           OkToPrint = YES;          // YES or NO
79  int           PrintASCII = NO;          // Print ASCII data equivelents
80  int           ReverseDirection = NO;    // Swap input flags with output flags
81  int           TwoLineFeeds = NO;        // 1 or 2 LFs on direction change
82
83
84  int main(int argc, char *argv[])
85  {
86  char   *ptr2 = NULL;
87  char    temp[100];   // Dont expect an argument to be larger than 100 chars
88  long    i;
89  int     data;
90
91      fprintf(stderr,"\n$Header: d:/txb-p/Captures/RCS/db.c,v 1.17 2002-07-12 08:13:19-07
Hamilton Exp Hamilton $\n");
92
93      // Process the args
94      while (--argc) {
95          strcpy(temp, *(++argv));
96          strupr(temp);
97
98          switch (temp[0]) {
99              // For Unix comment out the following line "case '/':"
100             // to enable root directory access, then use '-' for all switches
101             case '/':
102             case '-':
```

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

```
155                         __FILE__,__LINE__,InputPointer);
156         fprintf(stderr,"\a%12s, %4d: Unable to dump \"%s\"\n",
157                         __FILE__,__LINE__,InputPointer);
158         fclose(stdout);
159     }
160
161     fprintf(stderr,"%12s, %4d: Dumping \"%s\" a %ld byte file.\n",
162                     __FILE__,__LINE__,InputPointer, FileLength);
163     fprintf(stdout,"%%%12s, %4d: $Id: db.c,v 1.17 2002-07-12 08:13:19-07 Hamilton Exp Hamilton
 $\n",__FILE__,__LINE__);
164     fprintf(stdout,"%%\n");
165     fprintf(stdout,"%%%12s, %4d: Dumping \"%s\" a %ld byte file.\n",
166                     __FILE__,__LINE__,InputPointer, FileLength);
167     InFile = fopen(InputPointer,"rb");
168
169     // First check for Breakout's signature
170     if ((InFile != NULL) &&
171         (fgetc(InFile) == 0x0F) &&
172         (fgetc(InFile) == 0xA5) &&
173         (fgetc(InFile) == 0x66) &&
174         (fgetc(InFile) == 0x20)) {
175             fprintf(stdout,"%%\n");
176             fprintf(stdout,"%%%12s, %4d: The .CAP file's header is:\n",__FILE__,__LINE__);
177             // Then printout the breakout header
178             fprintf(stdout,"%%%12s, %4d: \"",__FILE__,__LINE__);
179             for (i = 4; i < 36; i++) {
180                 data = fgetc(InFile);
181                 if (data == 0x13) // If there is a CR then start a new line
182                     fprintf(stdout,"%%%12s, %4d: ",__FILE__,__LINE__);
183                 fprintf(stdout,"%c",data);
184             }
185             fprintf(stdout,"\"\n");
186             fprintf(stdout,"%%\n");
187         fprintf(stdout,"%%%12s, %4d: Format of this output is as
 follows:\n",__FILE__,__LINE__);
188         fprintf(stdout,"%%%12s, %4d: There are two directions of data
 flow.\n",__FILE__,__LINE__);
189         fprintf(stdout,"%%%12s, %4d: Data direction is indicated with
 either\n",__FILE__,__LINE__);
190         fprintf(stdout,"%%%12s, %4d: a \"1\" or \"0\" followed by a
 \"-\".\n",__FILE__,__LINE__);
191         fprintf(stdout,"%%%12s, %4d: It is intended that \"1\" will\n",__FILE__,__LINE__);
192         fprintf(stdout,"%%%12s, %4d: be used to indicate data that is coming
 in,\n",__FILE__,__LINE__);
193         fprintf(stdout,"%%%12s, %4d: and that \"0\" is used for data\n",__FILE__,__LINE__);
194         fprintf(stdout,"%%%12s, %4d: going out. If this is not true, then
 use\n",__FILE__,__LINE__);
195         fprintf(stdout,"%%%12s, %4d: the /R option to reverse the logic\n",__FILE__,__LINE__);
196         fprintf(stdout,"%%%12s, %4d: of these indicators\n",__FILE__,__LINE__);
197         fprintf(stdout,"%%%12s, %4d: Following the \"-\" is the value of
 the\n",__FILE__,__LINE__);
198         fprintf(stdout,"%%%12s, %4d: data byte in hexidecimal.\n",__FILE__,__LINE__);
```

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

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

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

APPENDIX C

C Fix0.l

C.1 Description

DB breaks sequences of commands whenever the direction of data flow changes. Sometimes additional breaks are desirable and FIX0 makes them.

C.2 Calling Sequence

```
fix0 infile outfile
```

C.3 Input Data Format

FIX0 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
```

C.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.

C.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
```

C.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/Fix0.l,v 1.3 2004-11-30 15:07:13-08 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 only 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: Fix0.l,v 1.3 2004-11-30 15:07:13-08 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%% %s     %5d $Id: Fix0.l,v 1.3 2004-11-30 15:07:13-08 Hamilton Exp
Hamilton $\n",
71                          __FILE__,__LINE__);
72          t = time(NULL);
73          thistime = localtime(&t);
74          fprintf(yyout,"%% %s     %5d %s",
75                          __FILE__,__LINE__,asctime(thistime));
76          fprintf(yyout,"%% %s     %5d Reading from \"%s\"\n",
77                          __FILE__,__LINE__,infilename);
78          fprintf(yyout,"%% %s     %5d Writing into \"%s\"\n%%\n",
79                          __FILE__,__LINE__,outfilename);
80
81          yylex();
82          exit(EXIT_SUCCESS);
83  }
84
85  // $Log: Fix0.l,v $
86  // Revision 1.3  2004-11-30 15:07:13-08  Hamilton
87  // Normal end of day data saving
88  //
89  // Revision 1.2  2004-11-30 13:06:10-08  Hamilton
90  // Normal end of day data saving
91  //
92  // Revision 1.1  2004-11-12 07:10:11-08  Hamilton
93  // Initial revision
94  //
95
```

APPENDIX D

D FixB.l

D.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 FIX0 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.

D.2 Calling Sequence

```
fixb infile outfile
```

D.3 Input Data Format

This routine is designed to take a BREAKOUT capture file that has been processed by FIX0. The output of FIX0 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
```

D.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.

D.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
```

D.6 Listing

```
 1  %{
 2  // $Header: d:/txb-p/Captures/RCS/FixB.l,v 1.10 2004-11-30 15:07:14-08 Hamilton Exp Hamilton $
 3  // FIXB is used to reformat breakout formatted captured files into FTS format.
 4  #include <stdio.h>
 5  #include <stdlib.h>
 6  #include <ctype.h>
 7  #include <time.h>
 8
 9  int lmsgnbr = 0;
10  int msgnbr = 0;
11  int vlmsgnbr = 0;
12
13  char digit[10];
14
15  void messagenumber(void);
16
17  #undef yywrap
18  %}
19  %x DUMPCR
20  %%
21  "%".*\n              ECHO; // Dump comments starting with a % sign
22
23  " I-00"              ; // nulls are always illegal
24  "I-00"               ;
25  " O-00"              ;
26  "O-00"               ;
27
28  "I-"[0-9A-F]{2}  {fprintf(yyout,"DCE 11/22/3333 44:%02d:%02d.%03d PM %c%c\n",
29                                 vlmsgnbr,lmsgnbr,msgnbr,
30                                 tolower(yytext[2]),tolower(yytext[3]));
31                   digit[0] = yytext[2];
32                   digit[1] = yytext[3];
33                   messagenumber();
34                   BEGIN DUMPCR;
35                  }
36  " I-"[0-9A-F]{2} {fprintf(yyout,"DCE 11/22/3333 44:%02d:%02d.%03d PM %c%c\n",
37                                 vlmsgnbr,lmsgnbr,msgnbr,
38                                 tolower(yytext[3]),tolower(yytext[4]));
39                   digit[0] = yytext[3];
40                   digit[1] = yytext[4];
41                   messagenumber();
42                   BEGIN DUMPCR;
43                  }
44
45
46  "O-"[0-9A-F]{2}  {fprintf(yyout,"DTE 11/22/3333 44:%02d:%02d.%03d PM %c%c\n",
47                                 vlmsgnbr,lmsgnbr,msgnbr,
48                                 tolower(yytext[2]),tolower(yytext[3]));
49                   digit[0] = yytext[2];
50                   digit[1] = yytext[3];
51                   messagenumber();
```

```
52                    BEGIN DUMPCR;
53                   }
54  " 0-"[0-9A-F]{2} {fprintf(yyout,"DTE 11/22/3333 44:%02d:%02d.%03d PM %c%c\n",
55                                  vlmsgnbr,lmsgnbr,msgnbr,
56                                  tolower(yytext[3]),tolower(yytext[4]));
57                    digit[0] = yytext[3];
58                    digit[1] = yytext[4];
59                    messagenumber();
60                    BEGIN DUMPCR;
61                   }
62
63
64  <DUMPCR>"%".*\n  ;   // Dump comments starting with a % sign
65  <DUMPCR>\n       BEGIN 0;
66  <DUMPCR>.        BEGIN 0;
67
68  .|\n             ECHO;
69  %%
70  // Increment the message number
71  void messagenumber(void)
72  {
73      if ((digit[0] == '0') && (digit[1] == '3')) {
74           msgnbr++;
75           if (msgnbr > 999) {
76               msgnbr = 0;
77               lmsgnbr++;
78               if (lmsgnbr > 99) {
79                   lmsgnbr =0;
80                   if (lmsgnbr > 99) {
81                       lmsgnbr = 0;
82                       vlmsgnbr++;
83                   }
84               }
85          }
86      }
87  }
88
89
90  // yywrap is for End Of File processing, 1 = done
91  int yywrap(void)
92  {
93          return 1;
94  }
95
96  main(int argc, char *argv[])
97  {
98  char infilename[100];
99  char outfilename[100];
100
101 time_t thistime;
102 struct tm *t;
103
```

```
104         fprintf(stderr,"$Id: FixB.l,v 1.10 2004-11-30 15:07:14-08 Hamilton Exp Hamilton $\n");
105
106         strcpy(infilename,"stdin");
107         strcpy(outfilename,"stdout");
108
109         // Do we have to deal with stdin/stdout?
110         // 3 = one for the program name and one each for input and output names
111         if (argc == 3) {
112             // Nope
113             // Get over the program, etc., name
114             argc--;
115             argv++;
116             yyin = fopen(argv[0],"r");
117             if (yyin == NULL) {
118                 fprintf(stderr,"\aCould not open \"%s\" for reading, quitting",
119                                 argv[0]);
120                 exit(EXIT_FAILURE);
121             }
122             strcpy(infilename,argv[0]);
123
124             // Now get over the input file name and get to the output file name
125             argc--;
126             argv++;
127             yyout = fopen(argv[0],"w");
128             if (yyout == NULL) {
129                 fprintf(stderr,"\aCould not open \"%s\" for writing, quitting",
130                                 argv[0]);
131                 exit(EXIT_FAILURE);
132             }
133             strcpy(outfilename,argv[0]);
134         }
135         else {
136             // Woops we are using std... type io
137             yyin  = stdin;
138             yyout = stdout;
139         }
140
141         fprintf(yyout,"%% %s     %5d $Id: FixB.l,v 1.10 2004-11-30 15:07:14-08 Hamilton Exp
 Hamilton $\n",
142                         __FILE__,__LINE__);
143
144         t = time(NULL);
145         thistime = localtime(&t);
146         fprintf(yyout,"%% %s     %5d %s",
147                         __FILE__,__LINE__,asctime(thistime));
148         fprintf(yyout,"%% %s     %5d Reading from \"%s\"\n",
149                         __FILE__,__LINE__,infilename);
150         fprintf(yyout,"%% %s     %5d Writing into \"%s\"\n%%\n",
151                         __FILE__,__LINE__,outfilename);
152
153         fprintf(yyout,"FTS capture file, generated from a breakout file.\n");
154         fprintf(yyout,"Event <none>\n");
```

```
155          fprintf(yyout,"Event <none>\n");
156          fprintf(yyout,"\n");
157          fprintf(yyout,"Timestamp <none>\n");
158          fprintf(yyout,"Sd Timestamp <none>\n");
159          yylex();
160          exit(EXIT_SUCCESS);
161  }
162
163  // $Log: FixB.l,v $
164  // Revision 1.10  2004-11-30 15:07:14-08  Hamilton
165  // Normal end of day data saving
166  //
167  // Revision 1.9  2004-11-30 13:06:10-08  Hamilton
168  // Normal end of day data saving
169  //
170  // Revision 1.8  2004-11-15 15:31:40-08  Hamilton
171  // Normal end of day data saving
172  //
173  // Revision 1.7  2004-11-11 13:27:23-08  Hamilton
174  // Normal end of day data saving
175  //
176  // Revision 1.6  2004-11-11 08:45:51-08  Hamilton
177  // Normal end of day data saving
178  //
179  // Revision 1.5  2004-11-11 07:45:21-08  Hamilton
180  // Normal end of day data saving
181  //
182  // Revision 1.4  2004-11-11 07:14:23-08  Hamilton
183  // Normal end of day data saving
184  //
185  // Revision 1.3  2004-11-04 12:19:54-08  Hamilton
186  // Normal end of day data saving
187  //
188
```

APPENDIX E

Index