SYSTEM LIBRARY LISTINGS

THE CORVUS CONCEPT

CORVUS SYSTEMS

CORVUS SYSTEMS

The Corvus Concept
System Library Listings

PART NO. : 7100-03293
DOCUMENT NO. : CCC/30-33/1.1
RELEASE DATE : February, 1983

TABLE OF CONTENTS

```
    1. { CCDEFN.TEXT ------------------------------------------------------------}
    2. {
    3. {       CCDEFN -- Corvus CONCEPT Definition Unit
    4. {
    5. {       (c) Copyright 1983 Corvus Systems, Inc.
    6. {                          San Jose, California
    7. {
    8. {       All Rights Reserved
    9. {
   10. {       v 1.0  11-01-81  LEF  Original unit
   11. {       v 1.1  01-17-82  PHB  a few mods...
   12. {       v 1.2  03-24-82  LEF  Add SndRcvStr definition
   13. {       v 1.3  04-05-82  LEF  Add window record definition
   14. {                             Add I/O result equates
   15. {       v 1.4  10-18-82  LEF  Window record definition moved to CCwndIO
   16. {       v 1.5  01-07-83  LEF  Added more I/O result codes
   17. {
   18. {-----------------------------------------------------------------------}
   19.
   20. UNIT CCdefn;
   21.
   22. INTERFACE
   23.
   24. CONST
   25.     MAXWINDOW    = 20;
   26.     SysComPLoc   = $0180;
   27.     LongStrMax   = 1030;
   28.     MaxBytes     = 10000;
   29.
   30.     {                                                                  }
   31.     { Corvus CONCEPT I/O Result Codes                                  }
   32.     {                                                                  }
   33.
   34.     IOok      = 00; { Good result, no error                            }
   35.     IOEinvdev= 02; { Invalid unit number/invalid device                }
   36.     IOEioreq = 03; { Invalid I/O request                               }
   37.
   38.     IOEnotrn = 21; { Transporter not ready                             }
   39.     IOEtimot = 22; { Timed out waiting for Omninet event               }
   40.     IOEnobuf = 23; { Read without a valid write buffer                 }
   41.
   42.     IOEwndfn = 32; { Invalid window function                           }
   43.     IOEwndbe = 33; { Window create boundary                            }
   44.     IOEwndcs = 34; { Invalid character set                             }
   45.     IOEwnddc = 35; { Delete current window                             }
   46.     IOEwndds = 36; { Delete system window                              }
   47.     IOEwndiw = 37; { Inactive window                                   }
   48.     IOEwndwr = 38; { Invalid window record                             }
   49.     IOEwndwn = 39; { Invalid system window number                      }
   50.
   51.     IOEnodsp = 40; { Display driver not available                      }
   52.     IOEnokyb = 41; { Keyboard driver not available                     }
   53.     IOEnotim = 42; { Timer driver not available                        }
   54.     IOEnoomn = 43; { OMNINET driver not available                      }
```

```
55.       IOEnoprt = 44; { Printer driver not available          }
56.       IOEnfdrv = 45; { No floppy drive at slot               }
57.       IOEnodtc = 46; { DataComm driver not available         }
58.
59.       IOEtblid = 50; { Invalid table entry ID                }
60.       IOEtblfl = 51; { Table full                            }
61.       IOEtbliu = 52; { Table entry in use                    }
62.       IOEkybte = 53; { Keyboard transmission error           }
63.       IOEuiopm = 54; { Invalid unit I/O parameter            }
64.       IOEprmln = 55; { Invalid parameter block length        }
65.       IOEfnccd = 56; { Invalid function code                 }
66.       IOEclkmf = 57; { Clock (hardware) malfunction          }
67.
68.       IOEirdsbl= 60; { Input to read buffer disabled         }
69.       IOEordsbl= 61; { Output to read buffer disabled        }
70.       IOEiwdsbl= 62; { Input to write buffer disabled        }
71.       IOEowdsbl= 63; { Output to write buffer disabled       }
72.       IOEbszerr= 64; { Buffer size error                     }
73.       IOEwszerr= 65; { Write size error                      }
74.       IOErszerr= 66; { Read size error                       }
75.       IOEuarter= 67; { UART hardware error                   }
76.       IOEpaderr= 68; { Proportional spacing error            }
77.
78.  TYPE
79.       Byte          = -128..127;
80.       pByte         = ^Byte;
81.       String32      = STRING[32];
82.       pString32     = ^String32;
83.       String64      = STRING[64];
84.       pString64     = ^String64;
85.       String80      = STRING[80];
86.       pString80     = ^String80;
87.       Bytes         = ARRAY [0..9999] OF Byte;
88.       Words         = ARRAY [0..9999] OF INTEGER;
89.       pBytes        = ^Bytes;
90.       pWords        = ^Words;
91.
92.       SlotType      = (NoDisk, LocalDisk, OmninetDisk,
93.                        FlpyC8Disk, FlpyC5Disk, FlpyA5Disk);
94.
95.  IMPLEMENTATION
96.
97.  END.
98.
```

SYSCOMPLOC 26
WORDS 88 90

```
 1. { CCHEXOUT.TEXT ------------------------------------------------------}
 2. {
 3. {       CCHEXOUT -- Output Hex Characters Unit
 4. {
 5. {       (c) Copyright 1982 Corvus Systems, Inc.
 6. {                          San Jose, California
 7. {
 8. {       All Rights Reserved
 9. {
10. {       v 1.0  01-16-82  PHB  Original unit
11. {
12. {--------------------------------------------------------------------}
13. {$R-}
14.
15. UNIT CChexOut;
16.
17. INTERFACE
18.
19. USES {$U CCLIB} CCdefn;
20.
21. PROCEDURE CChexInit;
22. PROCEDURE puthexbyte (b: byte);
23. PROCEDURE puthexword (w: integer);
24. PROCEDURE puthexlong (l: longint);
25. PROCEDURE dumphex     (p: pBytes; len: integer);
26.
27. IMPLEMENTATION
28.
29. {$P}
```

```
30. TYPE
31.     NIBBLE = 0..15;
32.     HBYTE = packed array [0..1] of NIBBLE;
33.     HWORD = packed array [0..1] of HBYTE;
34.     HLONG = packed array [0..3] of HBYTE;
35.
36. VAR
37.     hexstr: array [0..15] of CHAR;
38.
39.
40. PROCEDURE CChexInit;
41.     var i: integer; ts: STRING32;
42.     begin
43.     ts := '0123456789ABCDEF';
44.     for i := 0 to 15 do hexstr[i] := ts[i+1];
45.     end;
46.
47.
48. PROCEDURE puthexbyte {(b: byte)};
49.     var trix: packed record case integer of
50.                     1: (h: HBYTE);
51.                     2: (num: byte);
52.                     end;
53.     begin
54.     with trix do begin
55.         num := b;
56.         write (hexstr[h[1]], hexstr[h[0]]);
57.         end;
58.     end;
59.
60.
61. PROCEDURE puthexword {(w: integer)};
62.     var i: integer;
63.         trix: packed record case integer of
64.                     1: (h: HWORD);
65.                     2: (num: integer);
66.                     end;
67.     begin
68.     with trix do begin
69.         num := w;
70.         for i := 0 to 1 do write (hexstr[h[i][1]], hexstr[h[i][0]]);
71.         end;
72.     end;
73.
74. {$P}
```

```
 75. PROCEDURE puthexlong {(l: longint)};
 76.     var i: integer;
 77.         trix: packed record case integer of
 78.                   1: (h: HLONG);
 79.                   2: (num: longint);
 80.                   end;
 81.     begin
 82.     with trix do begin
 83.         num := l;
 84.         for i := 0 to 3 do write (hexstr[h[i][1]], hexstr[h[i][0]]);
 85.         end;
 86.     end;
 87.
 88.
 89. PROCEDURE dumphex {(p: pBytes; len: integer)};
 90.     var i: integer;
 91.         trix: packed record case integer of
 92.                   1: (h: HBYTE);
 93.                   2: (num: byte);
 94.                   end;
 95.     begin
 96.     if len > MaxBytes then len := MaxBytes;
 97.     for i := 0 to len - 1 do begin
 98.         with trix do begin
 99.             num := p^[i];
100.             write (hexstr[h[1]], hexstr[h[0]], ' ');
101.             end;
102.         if i MOD 4 = 3 then begin
103.             write (' ');
104.             if i MOD 16 = 15 then writeln;
105.             if i MOD 128 = 127 then writeln;
106.             end;
107.         end;
108.     end;
109.
110. end.    {end of UNIT hexout}
111.
```

```
    1  { CCLNGINT.TEXT ------------------------------------------------------}
    2  {
    3  {       CCLNGINT -- Corvus CONCEPT Long Integer Unit
    4  {
    5  {       (c) Copyright 1982 Corvus Systems, Inc.
    6  {                          San Jose, California
    7  {
    8  {       All Rights Reserved
    9  {
   10  {       v 1.0  05-21-82  DP   Original unit
   11  {
   12  {--------------------------------------------------------------------}
   13  {$R-}
   14
   15  UNIT CCLngInt;
   16
   17  INTERFACE
   18
   19  USES {$U CCLIB} CCdefn;
   20
   21  FUNCTION  LIntByte (Which: integer; Num: longint): byte;
   22  PROCEDURE ByteLInt (VAR Num: longint; byte0,byte1,byte2,byte3: byte);
   23  FUNCTION  Int2Byte (Which,Num: INTEGER): byte;
   24  PROCEDURE Byte2Int (VAR Num: INTEGER; byte0,byte1: byte);
   25
   26
   27  IMPLEMENTATION
   28
   29  TYPE
   30
   31      Longaddr = RECORD CASE INTEGER OF
   32                   0: (Longword: LONGINT);
   33                   1: (Longbyte: PACKED ARRAY [0..3] OF BYTE);
   34                   END;
   35
   36      Intaddr  = RECORD CASE INTEGER OF
   37                   0: (int: INTEGER);
   38                   1: (Byt: PACKED ARRAY [0..1] OF BYTE);
   39                   END;
   40
   41
   42  {$P}
```

```
43. {-------------------------------------------------------------------------}
44. { Procedure:    LINTBYTE
45. {
46. { Description: This procedure returns the byte indicated by 'WHICH'
47. {              from the long integer 'NUM'.  The least significant byte
48. {              of the long integer is byte zero.
49. {
50. {-------------------------------------------------------------------------}
51.
52. FUNCTION LIntByte {(Which: integer; Num: longint): byte};
53.     VAR ByteNum: LongAddr;
54.     BEGIN
55.     ByteNum.LongWord:= Num;
56.     LIntByte := ByteNum.LongByte[Which];
57.     END; { LIntByte }
58.
59.
60. {-------------------------------------------------------------------------}
61. { Procedure:    BYTELINT
62. {
63. { Description: This procedure converts four byte quantities into a long
64. {              integer value.  Byte0 is the most  significant byte of
65. {              the long integer;  Byte3 is the least significant byte
66. {              Replacement is used instead of the arithmetic
67. {              solution for speed and compactness of code.
68. {
69. {-------------------------------------------------------------------------}
70.
71. PROCEDURE ByteLInt {(VAR Num: longint; byte0,byte1,byte2,byte3: byte)};
72.     VAR ByteNum: LongAddr;
73.     BEGIN
74.     ByteNum.LongByte[0] := Byte0;
75.     ByteNum.LongByte[1] := Byte1;
76.     ByteNum.LongByte[2] := Byte2;
77.     ByteNum.LongByte[3] := Byte3;
78.     Num := ByteNum.LongWord;
79.     END; { ByteLInt }
80.
81.
82. {$P}
```

```
 83. {-------------------------------------------------------------------------}
 84. { Procedure:    INT2BYTE
 85. {
 86. { Description:
 87. {
 88. {-------------------------------------------------------------------------}
 89.
 90. FUNCTION Int2Byte {(Which,Num: INTEGER): byte};
 91.     VAR ByteNum: IntAddr;
 92.     BEGIN
 93.     ByteNum.Int := Num;
 94.     Int2Byte := ByteNum.Byt[Which];
 95.     END; { Byte2Int }
 96.
 97.
 98. {-------------------------------------------------------------------------}
 99. { Procedure:    BYTE2INT
100. {
101. { Description:
102. {
103. {-------------------------------------------------------------------------}
104.
105. PROCEDURE Byte2Int {(VAR Num: INTEGER; byte0,byte1: byte)};
106.     VAR ByteNum: IntAddr;
107.     BEGIN
108.     ByteNum.Byt[0] := byte0;
109.     ByteNum.Byt[1] := byte1;
110.     Num := ByteNum.Int;
111.     END; { Byte2Int }
112.
113. END.
114.
```

```
{ CCCLKIO.TEXT ------------------------------------------------------------}
{
{       CCCLKIO -- Corvus CONCEPT Clock Processing Unit
{
{       (c) Copyright 1982 Corvus Systems, Inc.
{                          San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  04-10-82  LEF  Original unit
{       v 1.1  09-07-82  LEF  Rework of INTERFACE section
{
{-------------------------------------------------------------------------}
{$R-}

UNIT CCclkIO;

INTERFACE

TYPE
    ClkStr40 = string[40];
    ClkPB    = record
                 DayofWeek,Month,Day:              integer; { set by driver!
                 Hour,Mins,Secs,Tenths,LeapYear:   integer; { set by driver!
                 Year:                             integer; { set by unit  !
                 end;

    pClkDateRcd = ^ClkDateRcd;
    ClkDateRcd  = packed record
                    year:  0..100;
                    day:   0..31;
                    month: 0..12;
                    end;

PROCEDURE CCclkIOinit;
PROCEDURE ClkRead     (var CPB: ClkPB);
PROCEDURE ClkWrite    (CPB: ClkPB);
PROCEDURE ClkWeekDay  (var DateStr: ClkStr40); {day of week}
PROCEDURE ClkDate1    (var DateStr: ClkStr40); {"dy-mon-yr" format}
PROCEDURE ClkDate2    (var DateStr: ClkStr40); {"month dy, year" format}
PROCEDURE ClkDate3    (var DateStr: ClkStr40); {"dy month year" format}
PROCEDURE ClkTime1    (var DateStr: ClkStr40); {"hr:mi:sc" format}
PROCEDURE ClkTime2    (var DateStr: ClkStr40); {"hr:mi am" format}
PROCEDURE CvDateStr   (DateStr: ClkStr40; var drcd: ClkDateRcd);


IMPLEMENTATION

{$P}
```

```
50. CONST wrlen = $10;
51.       rdlen = $0E;
52.
53. TYPE  ClkStr2  = string[2];
54.       ClkStr10 = string[10];
55.
56. VAR   sysdate:  ClkDateRcd; { system date }
57.       ClkWD:    ClkStr10;   { day of week }
58.       ClkYr:    ClkStr10;   { year        }
59.       ClkMo:    ClkStr10;   { month       }
60.       ClkDy:    ClkStr2;    { day         }
61.       ClkHr:    ClkStr2;    { hour        }
62.       ClkMi:    ClkStr2;    { minute      }
63.       ClkSc:    ClkStr2;    { second      }
64.       ClkInfo:  ClkPB;      { clock parameter block }
65.
66. FUNCTION OStimDv: integer; external;
67. FUNCTION pOSdate: pClkDateRcd; external;
68.
69.
70. { CvtInt ------------------------------------------------------------}
71. { Convert integer to ClkStr2 string
72. {------------------------------------------------------------------}
73.
74. PROCEDURE cvtint (i: integer; var st: ClkStr2);
75.       begin
76.       st := '--'; i := i mod 100;
77.       st[1] := chr((i div 10)+ord('0'));
78.       st[2] := chr((i mod 10)+ord('0'));
79.       end;
80.
81.
82. { WeekDay ------------------------------------------------------------}
83. { Compute day of week (1..7 = Sunday to Saturday)
84. {------------------------------------------------------------------}
85.
86. FUNCTION WeekDay (d,m,y: integer): integer;
87.     begin
88.     if m <= 2 then begin m := m + 12; y := y - 1; end;
89.     WeekDay := ((y{*365} + (y div 4) + m*28 +
90.                ((13*m - 12) div 5) + d - 30) mod 7) + 1;
91.     end; {WeekDay}
92.
93.
94. {$P}
```

```
{ ClkFormat ------------------------------------------------------------}
{----------------------------------------------------------------------}

PROCEDURE ClkFormat (CPB: ClkPB);
        var yr: ClkStr2;
        begin
        with CPB do begin
            ClkWD := ('');
            case DayofWeek of
                  1: ClkWD := ('Sunday');
                  2: ClkWD := ('Monday');
                  3: ClkWD := ('Tuesday');
                  4: ClkWD := ('Wednesday');
                  5: ClkWD := ('Thursday');
                  6: ClkWD := ('Friday');
                  7: ClkWD := ('Saturday');
                end; {case}
            cvtint (Year,yr);
            ClkYr := concat ('19',yr);
            ClkMo := ('');
            case Month of
                  1: ClkMo := ('January');
                  2: ClkMo := ('February');
                  3: ClkMo := ('March');
                  4: ClkMo := ('April');
                  5: ClkMo := ('May');
                  6: ClkMo := ('June');
                  7: ClkMo := ('July');
                  8: ClkMo := ('August');
                  9: ClkMo := ('September');
                 10: ClkMo := ('October');
                 11: ClkMo := ('November');
                 12: ClkMo := ('December');
                end; {case}
            cvtint (day, ClkDy);
            cvtint (hour,ClkHr);
            cvtint (mins,ClkMi);
            cvtint (secs,ClkSc);
            end;
        end;


{$P}
```

```
138. { ClkWrite ------------------------------------------------------------}
139. { Write system clock
140. {---------------------------------------------------------------------}
141.
142. PROCEDURE ClkWrite; {(CPB: ClkPB)}
143.       var timer: integer;
144.       begin
145.       with CPB do begin
146.           DayofWeek := WeekDay (Day,Month,sysdate.year);
147.           LeapYear := Year mod 4;
148.           end;
149.       timer := OStimDv;
150.       unitwrite (timer,CPB,wrlen);
151.       timer := ioresult;
152.       if timer <> 0 then writeln ('Clock write error: ',timer:1);
153.       end;
154.
155.
156. { ClkRead -------------------------------------------------------------}
157. { Read system clock
158. {---------------------------------------------------------------------}
159.
160. PROCEDURE ClkRead; {(var CPB: ClkPB)}
161.       var timer: integer; psysdate: pClkDateRcd;
162.       begin
163.       timer := OStimDv;
164.       unitread (timer,CPB,rdlen);
165.       timer := ioresult;
166.       if timer <> 0 then writeln ('Clock read error: ',timer:1);
167.       psysdate := pOSdate; sysdate := psysdate^;
168.       with CPB do begin
169.           year := sysdate.year;
170.           LeapYear := Year mod 4;
171.           end;
172.       end;
173.
174.
175. { ClkWeekDay ----------------------------------------------------------}
176. { Return day of week string
177. {---------------------------------------------------------------------}
178. PROCEDURE ClkWeekDay {(var DateStr: ClkStr40)};
179.       begin
180.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
181.       DateStr := ClkWD;
182.       end;
183.
184.
185. {$P}
```

```
186. { ClkDate1 ------------------------------------------------------------}
187. { Return date string ("dy-mon-yr" format)
188. {------------------------------------------------------------------------}
189. PROCEDURE ClkDate1 {(var DateStr: ClkStr40)}; {"dy-mon-yr" format}
190.       begin
191.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
192.       DateStr := concat (ClkDy, '-', copy(ClkMo,1,3), '-', copy(ClkYr,3,2));
193.       end;
194.
195.
196. { ClkDate2 ------------------------------------------------------------}
197. { Return date string ("month dy, year" format)
198. {------------------------------------------------------------------------}
199. PROCEDURE ClkDate2 {(var DateStr: ClkStr40)}; {"month dy, year" format}
200.       var dy: ClkStr2;
201.       begin
202.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
203.       dy := ClkDy; if dy[1] = '0' then delete (dy,1,1);
204.       DateStr := concat (ClkMo,' ',dy,', ',ClkYr);
205.       end;
206.
207.
208. { ClkDate3 ------------------------------------------------------------}
209. { Return date string ("dy month year" format)
210. {------------------------------------------------------------------------}
211. PROCEDURE ClkDate3 {(var DateStr: ClkStr40)}; {"dy month year" format}
212.       var dy: ClkStr2;
213.       begin
214.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
215.       dy := ClkDy; if dy[1] = '0' then delete (dy,1,1);
216.       DateStr := concat (dy,' ',ClkMo,' ',ClkYr);
217.       end;
218.
219.
220. {$P}
```

```
221. { ClkTime1 ------------------------------------------------------------------}
222. { Return time string ("hr:mi:sc" format)
223. {-----------------------------------------------------------------------------}
224. PROCEDURE ClkTime1 ((var DateStr: ClkStr40)); {"hr:mi:sc" format}
225.       begin
226.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
227.       DateStr := concat (ClkHr,':',ClkMi,':',ClkSc);
228.       end;
229.
230.
231. { ClkTime2 ------------------------------------------------------------------}
232. { Return time string ("hr:mi am" format)
233. {-----------------------------------------------------------------------------}
234. PROCEDURE ClkTime2 ((var DateStr: ClkStr40)); {"hr:mi am" format}
235.       var hr,ampm: ClkStr2;
236.       begin
237.       ClkRead (ClkInfo); ClkFormat (ClkInfo);
238.       with ClkInfo do begin
239.           if Hour in [0..11] then ampm := 'am' else ampm := 'pm';
240.           if Hour = 0 then Hour := 12;
241.           if Hour > 12
242.               then cvtint (Hour-12,hr)
243.               else cvtint (Hour,hr);
244.           if hr[1] = '0' then delete (hr,1,1);
245.           end;
246.       DateStr := concat (hr,':',ClkMi,' ',ampm);
247.       end;
248.
249.
250. {$P}
```

```
  251. { CvDateStr ------------------------------------------------------------}
  252. { Convert ClkStr40 string to binary date
  253. {----------------------------------------------------------------------}
  254.
  255. PROCEDURE CvDateStr ((DateStr: ClkStr40; var drcd: ClkDateRcd));
  256.       var i,ix: integer; s: ClkStr40; ch: char; ok: boolean;
  257.
  258.       FUNCTION nextch: char;
  259.           var ch: char;
  260.           begin
  261.           if ix <= length(s)
  262.               then begin
  263.                   ch := s[ix]; ix := ix + 1;
  264.                   if ch >= 'a' then ch := chr(ord(ch) - 32);
  265.                   end
  266.               else ch := '~';
  267.           nextch := ch;
  268.           end; {nextch}
  269.
  270.       FUNCTION GetMonth (var fmonth: integer): Boolean;
  271.           var n: integer; m: packed array [1..3] of char; result: boole
  272.           begin
  273.           result := FALSE;
  274.           while not (ch in ['A'..'Z','~']) do ch := nextch;
  275.           n := 0;
  276.           while (ch >= 'A') and (ch <= 'Z') do begin
  277.                 n := n + 1;
  278.                 if n <= 3 then m[n] := ch;
  279.                 ch := nextch;
  280.                 end;
  281.           if n >= 3 then begin
  282.                n := 0;
  283.                if m = 'JAN' then n := 1;
  284.                if m = 'FEB' then n := 2;
  285.                if m = 'MAR' then n := 3;
  286.                if m = 'APR' then n := 4;
  287.                if m = 'MAY' then n := 5;
  288.                if m = 'JUN' then n := 6;
  289.                if m = 'JUL' then n := 7;
  290.                if m = 'AUG' then n := 8;
  291.                if m = 'SEP' then n := 9;
  292.                if m = 'OCT' then n := 10;
  293.                if m = 'NOV' then n := 11;
  294.                if m = 'DEC' then n := 12;
  295.                if n > 0 then begin result := TRUE; fmonth := n; end;
  296.               end;
  297.           GetMonth := result;
  298.           if ok then ok := result;
  299.           end; {GetMonth}
  300.
  301. {$P}
```

```
302         FUNCTION GetNum (var fnum: integer; flo,fhi: integer): Boolean;
303             var val: integer; Answer,result: Boolean;
304             begin
305             while not (ch in ['0'..'9','~']) do ch := nextch;
306             val := 0; Answer := FALSE;
307             while (ch >= '0') and (ch <= '9') do begin
308                   Answer := TRUE;
309                   val := val*10 + ord(ch) - ord('0');
310                   ch := nextch;
311                   end;
312             fnum := val;
313             result := Answer and ((val >= flo) and (val <= fhi));
314             GetNum := result;
315             if ok then ok := result;
316             end;
317
318         begin {CvDateStr}
319             s := DateStr; ix := 1; ch := nextch; ok := TRUE;
320         while not (ch in ['A'..'Z','0'..'9','~']) do ch := nextch;
321         with dtcd do begin
322             if ch in ['0'..'9']
323             then begin
324                 if GetNum (i,1,31) then begin
325                     day := i;
326                     if GetMonth (i) then begin
327                         month := i;
328                         if GetNum (i,0,2000) then year := i mod 100;
329                         end;
330                     end;
331                 end
332             else begin
333                 if GetMonth (i) then begin
334                     month := i;
335                     if GetNum (i,1,31) then begin
336                         day := i;
337                         if GetNum (i,0,2000) then year := i mod 100;
338                         end;
339                     end;
340                 end;
341             if not ok then begin
342                 year := 0; month := 0; day := 0; end;
343             end;
344         end;
345
346
347 {$P}
```

```
348. { CCclkIOinit ------------------------------------------------------------}
349. { CCclkIO unit initialization
350. {-------------------------------------------------------------------------}
351.
352. PROCEDURE CCclkIOinit;
353.         begin
354.         ClkRead (ClkInfo);
355.         if not (ClkInfo.month in [1..12])
356.         or not (ClkInfo.day in [1..31]) then with ClkInfo do begin
357.             DayofWeek  := WeekDay (sysdate.day,sysdate.month,sysdate.year);
358.             Month      := sysdate.month;
359.             Day        := sysdate.day;
360.             Hour       := 0;
361.             Mins       := 0;
362.             Secs       := 0;
363.             Tenths     := 0;
364.             LeapYear   := sysdate.year mod 4;
365.             ClkWrite (ClkInfo);
366.             end;
367.         end;
368.
369. end.   {unit CCclkIO}
370.
```

```
{ CCCRTIO.TEXT ------------------------------------------------------------}
{
{       CCCRTIO -- Corvus CONCEPT CRT Control Unit
{
{       (c) Copyright 1982 Corvus Systems, Inc.
{                          San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  10-23-81  LEF  Original unit
{       v 1.1  01-16-82  PHB  Modifications for LONGINTs
{       v 1.2  04-29-82  LEF  Add display CRT commands
{       v 1.3  06-19-82  LEF  Fix GetByte, GetLongNum procedures
{       v 1.4  08-23-82  LEF  Fix GetByte to get input from INPUT file
{
{-------------------------------------------------------------------------}
{$R-}

UNIT CCcrtIO;

INTERFACE

USES {$U CCLIB} CCdefn;

CONST
    CCcrtIOversion = '1.4';

TYPE
    CrtRdx      = (BinRdx,OctRdx,DecRdx,HexRdx);
    CrtStatus   = (Normal, Escape, Error);

    CrtCommand = (ErasEOS,        {clear to end of screen}
                  ErasEOL,        {clear to end of line}
                  EraseALL,       {clear screen and home}
                  CursorHome,     {move cursor home}
                  CursorUp,       {move cursor up}
                  CursorDown,     {move cursor down}
                  CursorRight,    {move cursor right}
                  CursorLeft,     {move cursor left}
                  CursorFtab,     {forward tab}
                  CursorBtab,     {back tab}
                  CursorOff,      {display cursor OFF}
                  CursorOn,       {display cursor ON}
                  CursorUndscr,   {set underline cursor}
                  CursorInvrse,   {set inverse cursor}
                  InsertLine,     {insert line at cursor}
                  DeleteLine,     {delete line at cursor}
                  InsertChar,     {insert character at cursor}
                  DeleteChar,     {delete character at cursor}
                  InsertOff,      {character insert mode OFF}
                  InsertOn,       {character insert mode ON}
                  ScrollOff,      {scroll mode OFF}
                  ScrollOn,       {scroll mode ON}
                  PagingOff,      {paging mode OFF}
```

```
                    PagingOn,       {paging mode ON}
                    WrapOff,        {line wrap OFF}
                    WrapOn,         {line wrap ON}
                    GrfMode,        {set graphics mode}
                    TxtMode,        {set text mode}
                    InvrtScreen,    {invert screen video}
                    VdoNor,         {set normal video}
                    VdoInv,         {set inverse video}
                    VdoNorUnd,      {set normal underline video}
                    VdoInvUnd,      {set inverse underline video}
                    EchoOn,         {echo user input ON}
                    EchoOff,        {echo user input OFF}
                    TypAhdOn,       {type ahead allowed ON}
                    TypAhdOff,      {type ahead allowed OFF}
                    UcaseOn,        {convert user input to uppercase ON}
                    UcaseOff,       {convert user input to uppercase OFF}
                    BsupOn,         {blank suppress user input ON}
                    BsupOff,        {blank suppress user input OFF}
                    DefStrOn,       {output default strings ON}
                    DefStrOff,      {output default strings OFF}
                    DefNumOn,       {output default numeric values ON}
                    DefNumOff,      {output default numeric values OFF}
                    StartBeat,      {}
                    HeartBeat,      {}
                    LeadIn);        {}

VAR
    Beep      : char;         {bell character}
    CrtTpgm   : string[16];   {program name string}
    CrtTvrs   : string[16];   {program version number string}
    CrtTcpy   : string[80];   {copyright notice string}
    CrtEcho   : boolean;      {echo input flag          default - TRUE }
    CrtNdef   : boolean;      {output default number    default - TRUE }
    CrtSdef   : boolean;      {output default string    default - FALSE}
    CrtShft   : boolean;      {convert to uppercase     default - TRUE }
    CrtBsup   : boolean;      {blank suppress           default - FALSE}
    CrtTahd   : boolean;      {type ahead allowed       default - TRUE }
    ExtCRT    : boolean;      {TRUE if using an external terminal}
    WndowLin  : integer;      {window size - lines}
    WndowCol  : integer;      {window size - columns}


{$P}
```

```
  98. PROCEDURE CCcrtIOinit;
  99. FUNCTION  UpperCase  (ch: char): char;
 100. FUNCTION  GetLongNum (var num: LongInt): CrtStatus;
 101. FUNCTION  GetNum     (var num: integer): CrtStatus;
 102. FUNCTION  GetString  (var buf: String80): CrtStatus;
 103. FUNCTION  GetByte:   char;
 104. FUNCTION  CvStrInt   (buf: String80; var num: integer): CrtStatus;
 105. FUNCTION  CvStrLInt  (buf: String80; var num: LongInt): CrtStatus;
 106. PROCEDURE CvIntStr   (num: integer; var buf: String80; rdx: CrtRdx);
 107. PROCEDURE CvLIntStr  (num: LongInt; var buf: String80; rdx: CrtRdx);
 108. PROCEDURE CrtAction  (cmd: CrtCommand);
 109. PROCEDURE CrtTitle   (txt: String80);
 110. PROCEDURE CrtPrompt  (txt,opt: String80);
 111. PROCEDURE CrtPause   (var ch: char);
 112. PROCEDURE GoToXY     (x,y: integer);
 113. FUNCTION  BellTone   (timbre: byte; duration,period: integer): integer;
 114.
 115. {PROCEDURES/FUNCTIONS for compatibity}
 116. PROCEDURE Crt (cmd: CrtCommand);
 117.
 118.
 119. IMPLEMENTATION
 120.
 121. {$P}
```

```
122. CONST
123.      bs     =  08; {backspace character}
124.      cr     =  13; {carriage return character}
125.      esc    =  27; {escape character}
126.      del    = $7F; {backspace character}
127.
128. VAR
129.      display:   integer;
130.      BeatCnt:   integer;
131.      CrtInfo:   packed array [CrtCommand] of char;
132.      Prefixed:  array [CrtCommand] of boolean;
133.      hexstr:    array [0..15] of char;
134.
135. FUNCTION OSextCRT:   boolean;                        EXTERNAL;
136. FUNCTION OStimDv:    integer;                        EXTERNAL;
137. FUNCTION OSdispDv:   integer;                        EXTERNAL;
138. FUNCTION pOScurWnd:  pBytes;                        EXTERNAL;
139. FUNCTION pOSsysWnd (wndnbr: integer): pBytes;   EXTERNAL;
140.
141.
142. { UpperCase ------------------------------------------------------------}
143. { Convert character to upper case
144. {------------------------------------------------------------------------}
145.
146. FUNCTION UpperCase ((ch: char): char);
147.      begin
148.      if ch IN ['a'..'z'] then uppercase := chr(ord(ch)-ord('a')+ord('A'))
149.                          else uppercase := ch;
150.      end;
151.
152.
153. { GoToXY ---------------------------------------------------------------}
154. { Position cursor
155. {------------------------------------------------------------------------}
156.
157. PROCEDURE GoToXY ((x,y: integer));
158.      begin
159.      if ExtCRT
160.           then write (chr(esc),'=',chr(y+32),chr(x+32))
161.           else write (chr(esc),'=',chr(x),chr(y));
162.      end;
163.
164.
165. {$P}
```

```
166. { CrtAction ------------------------------------------------------------}
167. {
168. { Perform CRT action
169. {
170. {-----------------------------------------------------------------------}
171.
172. PROCEDURE CrtAction {(cmd: CrtCommand)};
173.     var cmdlen: integer; buf: packed array [1..3] of char;
174.     begin
175.     cmdlen := 0;
176.     if Prefixed[cmd] then begin
177.         cmdlen := cmdlen+1;
178.         buf[cmdlen] := CrtInfo[LeadIn];
179.         end;
180.     case cmd of
181.           EchoOn: CrtEcho := TRUE;       EchoOff: CrtEcho := FALSE;
182.         TypAhdOn: CrtTahd := TRUE;     TypAhdOff: CrtTahd := FALSE;
183.          UcaseOn: CrtShft := TRUE;      UcaseOff: CrtShft := FALSE;
184.            BsupOn: CrtBsup := TRUE;       BsupOff: CrtBsup := FALSE;
185.         DefStrOn: CrtSdef := TRUE;     DefStrOff: CrtSdef := FALSE;
186.         DefNumOn: CrtNdef := TRUE;     DefNumOff: CrtNdef := FALSE;
187.        StartBeat: begin BeatCnt := 1; writeln; exit (CrtAction); end;
188.        HeartBeat: if BeatCnt > WndowCol-1
189.                       then begin
190.                           CrtAction (StartBeat); exit (CrtAction); end
191.                       else BeatCnt := BeatCnt+1;
192.           VdoNor,
193.           VdoInv,
194.        VdoNorUnd,
195.        VdoInvUnd: begin
196.                   cmdlen := cmdlen+1;
197.                   buf[cmdlen] := 'G';
198.                   end;
199.               end; {case}
200.     if CrtInfo[cmd] <> chr(00) then begin
201.         cmdlen := cmdlen+1;
202.         buf[cmdlen] := CrtInfo[cmd];
203.         if extcrt then UNITWRITE (1,buf,cmdlen,0,12)
204.                   else UNITWRITE (display,buf,cmdlen,0,12);
205.         end;
206.     end;
207.
208.
209. { Crt ----------------------------------------------------------------------}
210. { Calls CrtAction (for compatibility)
211. {-----------------------------------------------------------------------}
212.
213. PROCEDURE Crt {(cmd: CrtCommand)};  {same as CrtAction}
214.     begin CrtAction (cmd); end;
215.
216.
217. {$P}
```

```
{ CvLIntStr ------------------------------------------------------------}
{ Convert long integer value to Bin, Oct, Dec, or Hex string value
{----------------------------------------------------------------------}

PROCEDURE CvLIntStr ((num: longint; var buf: String80; rdx: CrtRdx));
    var x,idx: integer; sign,ch: char;
        numrcd: record case integer of
                    1: (li: longint);
                    2: (bt: packed array [0..31] of 0..1);
                end;

    PROCEDURE getbits (n: integer);
        var i,n1,n2: integer;
        begin
        n1 := idx-n+1; n2 := idx;
        if n1 < 0 then n1 := 0;
        x := 0;
        for i := n1 to n2 do
            x := x*2 + numrcd.bt[((i div 8)*8)+(7-(i mod 8))];
        idx := idx-n;
        end;

    begin
    buf := ''; sign := ' ';
    if num = 0
        then begin buf := '0'; exit (CvLIntStr); end;
    if rdx = DecRdx then begin
        if num < 0 then begin
            if num = $80000000 then begin
                buf := '-2147483648'; exit (CvLIntStr); end;
            sign := '-'; num := 0-num;
            end;
        while num <> 0 do begin
            x := num MOD 10; num := num DIV 10;
            ch := chr(ord('0')+x);
            buf := concat (' ',buf); buf[1] := ch;
            end; {while}
        if sign <> ' ' then begin
            buf := concat (' ',buf); buf[1] := sign; end;
        exit (CvLIntStr);
        end;
{$P}
```

```
260.      numrcd.li := num; idx := 31;
261.      repeat
262.          case rdx of
263.              BinRdx: getbits (1);
264.              OctRdx: getbits (3);
265.              HexRdx: getbits (4);
266.              end;
267.          if x > 9 then ch := chr(ord('A')+x-10)
268.                   else ch := chr(ord('0')+x);
269.          buf := concat (' ',buf); buf[1] := ch;
270.          until idx < 0;
271.      while buf[1] = '0' do delete (buf,1,1);
272.      end;
273.
274.
275.  { CvIntStr ------------------------------------------------------------}
276.  { Convert integer value to Bin, Oct, Dec, or Hex string value
277.  {---------------------------------------------------------------------}
278.
279.  PROCEDURE CvIntStr ((num: integer; var buf: String80; rdx: CrtRdx));
280.      var numrcd: record case integer of
281.                      1: (l: longint);
282.                      2: (w: array [0..1] of integer);
283.                  end;
284.      begin
285.      if rdx = DecRdx
286.          then numrcd.l := num
287.          else with numrcd do begin l := 0; w[1] := num; end;
288.      CvLIntStr (numrcd.l,buf,rdx);
289.      end;
290.
291.
292.  {$P}
```

```
{ CvStrLInt ------------------------------------------------------------}
{----------------------------------------------------------------------}

FUNCTION CvStrLInt {(buf: String80; var num: LongInt): CrtStatus};
    var base,i,inc,mult: integer;

    PROCEDURE cnverr;
        begin num := 0; CvStrLInt := Error; exit (CvStrLInt); end;

    begin
    while pos(' ',buf) <> 0 do delete (buf,pos(' ',buf),1);
    num := 0; mult := 1; base := 10;
    if not (buf[1] IN ['0'..'9']) then begin
        case buf[1] of
            '+','#': base := 10;
            '$','!': base := 16;
                '%': base := 8;
                '-': begin base := 10; mult := -1 end;
            end; {case}
        delete (buf,1,1);
        end;
    for i := 1 to length(buf) do begin
        if not (buf[i] IN ['0'..'9','A'..'F']) then cnverr;
        inc := ord(buf[i])-48;
        if inc > 9 then inc := inc-7;     { 65-48 = 17, 17-7 = 10 }
        if not (inc < base) then cnverr;
        num := num * base + inc;
        end;
    num := num * mult;
    CvStrLInt := Normal;
    end;


{ CvStrInt -------------------------------------------------------------}
{----------------------------------------------------------------------}

FUNCTION CvStrInt {(buf: String80; var num: integer): CrtStatus};
    var li: LongInt;
    begin
    CvStrInt := CvStrLInt (buf,li);
    num := ord(li);
    end;


{$P}
```

```
338. { ReadString ------------------------------------------------------------}
339. {------------------------------------------------------------------------}
340.
341. FUNCTION ReadString (var buf: String80;
342.                       BsupFg,ShftFg,PrmpFg,NumOnly: boolean): CrtStatus;
343.     var c,c1: char; i: integer; validnum: boolean;
344.     begin
345.     if ShftFg then
346.         for i := 1 to length(buf) do buf[i] := uppercase (buf[i]);
347.     if BsupFg then
348.         while pos(' ',buf) <> 0 do delete (buf,pos(' ',buf),1);
349.     if PrmpFg then begin
350.         write (buf);
351.         for i := 1 to length(buf) do write (chr(bs));
352.         end;
353.     ReadString := Normal;
354.     if not CrtTahd then unitclear (1);
355.     read (c);
356.     if EOLN then exit (ReadString);
357.     i := 0; buf := ''; CrtAction (EraseEOL);
358.     repeat
359.         if not CrtEcho then
360.             if not (ord(c) in [del,bs]) then
361.                 write (chr(bs),' ',chr(bs));
362.         case ord(c) of
363.          del,bs: begin {bs}
364.                  if i > 0 then begin
365.                      delete (buf,i,1); i := i-1;
366.                      if CrtEcho then write (chr(bs),' ',chr(bs));
367.                      end;
368.                  c := chr(0);
369.                  end;
370.             esc: begin {esc}
371.                  ReadString := ESCAPE; exit (ReadString);
372.                  end;
373.             end; {case}
374.         if NumOnly and (c <> chr (0)) then begin
375.             validnum := FALSE;
376.             c := uppercase (c);
377.             if i = 0
378.                 then begin
379.                     if c in ['0'..'9','#','$','!','%','+','-'] then beg
380.                         validnum := TRUE;
381.                         c1 := c;
382.                         if c1 in ['0'..'9'] then c1 := '+';
383.                         end;
384.                     end
385. {$P}
```

```
386.                    else begin
387.                        case c1 of
388.                            '%': if c in ['0'..'7'] then validnum := TR!
389.                        '+','-','#': if c in ['0'..'9'] then validnum := TR!
390.                            '$','!': if c in ['0'..'9','A'..'F']
391.                                                    then validnum := TR!
392.                                end; {case}
393.                        end;
394.                if not validnum then begin
395.                    write (chr(bs),' ',chr(bs),beep);
396.                    c := chr(0);
397.                    end;
398.                end;
399.            if i = 80
400.                then begin
401.                    write (beep);
402.                    if CrtEcho then write (chr(bs),' ',chr(bs));
403.                    end
404.                else if c <> chr(0) then begin
405.                    buf := concat (buf,' ');
406.                    i := i+1; buf[i] := c;
407.                    end;
408.            read (c);
409.            until EOLN;
410.        if ShftFg then
411.            for i := 1 to length(buf) do buf[i] := uppercase (buf[i]);
412.        if BsupFg then
413.            while pos(' ',buf) <> 0 do delete (buf,pos(' ',buf),1);
414.        end;
415.
416.
417.    {$P}
```

```
418. { GetLongNum ------------------------------------------------------------}
419. {------------------------------------------------------------------------}
420.
421. FUNCTION GetLongNum {(var num: LongInt): CrtStatus};
422.     var snum: String80;
423.     begin
424.     if not CrtNdef then num := 0;
425.     CvLIntStr (num,snum,DecRdx);
426.     if ReadString (snum,TRUE,TRUE,CrtNdef,TRUE) = Escape then begin
427.         num := 0; GetLongNum := Escape; exit (GetLongNum); end;
428.     GetLongNum := CvStrLInt (snum,num);
429.     end;
430.
431.
432. { GetNum ---------------------------------------------------------------}
433. {------------------------------------------------------------------------}
434.
435. FUNCTION GetNum {(var num: integer): CrtStatus};
436.     var li: LongInt;
437.     begin
438.     li := num;
439.     GetNum := GetLongNum (li);
440.     num := ord(li);
441.     end;
442.
443.
444. { GetByte ---------------------------------------------------------------}
445. {------------------------------------------------------------------------}
446.
447. FUNCTION GetByte {: char};
448.     var ch: char;
449.     begin
450.     if not CrtTahd then unitclear (1);
451.     read (ch);
452.     if EOLN then ch := ' ';
453.     if EOF  then ch := '!';
454.     if ch = chr(esc) then ch := '!';
455.     if not CrtEcho then write (chr(bs),' ',chr(bs));
456.     GetByte := uppercase (ch);
457.     end;
458.
459.
460. {$P}
```

```
{ GetString ------------------------------------------------------------------}
{----------------------------------------------------------------------------}

FUNCTION GetString ((var buf: String80): CrtStatus);
    begin
    if not CrtSdef then buf := '';
    GetString := ReadString (buf,CrtBsup,CrtShft,CrtSdef,FALSE);
    end;


{ CrtTitle -------------------------------------------------------------------}
{----------------------------------------------------------------------------}

PROCEDURE CrtTitle ((txt: String80));
    begin
    GoToXY (0,0); CrtAction (EraseALL);
    CrtAction (VdoInv);
    GoToXY (0,0); CrtAction (ErasEOL);
    write (' ',CrtTpgm,' [',CrtTvrs,']: ',txt);
    GoToXY (0,1); CrtAction (ErasEOL);
    write (' ',CrtTcpy);
    CrtAction (VdoNor);
    GoToXY (0,2); CrtAction (ErasEOL);
    GoToXY (0,3);
    end;


{ CrtPrompt ------------------------------------------------------------------}
{----------------------------------------------------------------------------}

PROCEDURE CrtPrompt ((txt,opt: String80));
    begin
    GoToXY (0,WndowLin-1);
    if length(txt) <> 0 then write (txt)
                        else write ('Enter option');
    if length(opt) <> 0 then write (' [',opt,']');
    write (': '); CrtAction (ErasEOL);
    end;


{$P}
```

```
{ CrtPause ------------------------------------------------------------}
{---------------------------------------------------------------------}

PROCEDURE CrtPause {(ch: char)};
    var wptr1,wptr2: pBytes; line: integer;
    begin
    if extcrt
        then begin
            line := WndowLin;
            GoToXY (WndowCol-27,line)
            end
        else begin
            line := 1;
            wptr1 := pOScurWnd;
            wptr2 := pOSsysWnd (2);
            UnitStatus (display,wptr2^,3);
            GoToXY (0,line);
        end;
    write ('Press <space> to continue '); CrtAction (ErasEOL);
    CrtEcho := FALSE; ch := GetByte; CrtEcho := TRUE;
    GoToXY (0,line); CrtAction (ErasEOL);
    if not extcrt then UnitStatus (display,wptr1^,3);
    end;


{ BellTone ------------------------------------------------------------}
{  input (timbre:    byte;          {on and off of the speaker
{  input  duration:  integer;       {nmbr of 50 ms ticks to leave speaker !
{  input  period:    integer);      {time between speaker tones
{ result             integer);      {IORESULT}
{---------------------------------------------------------------------}

FUNCTION BellTone;
    var bellPB: record
                per: integer; tmb: byte; fil: byte; dur: integer;
                end;
        TimerUnit: integer;
    begin
    TimerUnit := OStimDv;
    with bellPB do begin
        per := period; tmb := timbre; fil := 0; dur := duration; end;
    UnitStatus (TimerUnit,bellPB,0);
    BellTone := IORESULT;
    end;


{$P}
```

```
549.  { CCcrtIOinit ------------------------------------------------------}
550.  { Unit initialization
551.  {------------------------------------------------------------------}
552.
553.  PROCEDURE CCcrtIOinit;
554.      type WinStatBuff = record xhome,yhome,xlen,ylen: integer; end;
555.      var i: integer; ts: String32; ws: WinStatBuff;
556.      begin
557.      ts := '0123456789ABCDEF';
558.      for i:= 0 to 15 do hexstr[i] := ts[i+1];
559.      Beep    := chr(7);
560.      CrtEcho := TRUE;     {input echo flag}
561.      CrtTahd := TRUE;     {type ahead allowed flag}
562.      CrtShft := TRUE;     {convert to uppercase flag}
563.      CrtBsup := FALSE;    {suppress spaces flag}
564.      CrtSdef := FALSE;    {default string processing}
565.      CrtNdef := TRUE;     {default number processing}
566.      CrtTpgm := 'pgmid'; CrtTvrs := '0.0';
567.      CrtTcpy := '(c) Copyright 1983 Corvus Systems, Inc.
568.      ExtCRT := OSextCRT;
569.      display := 0; WndowLin := 23; WndowCol := 79;
570.      if not ExtCRT then begin
571.          display := OSdispDv;
572.          UnitStatus (display,ws,5);
573.          if ioresult = 0 then begin
574.              WndowLin := ws.ylen; WndowCol := ws.xlen; end;
575.          end;
576.
577.      CrtInfo[LeadIn]     := chr(esc);    Prefixed[LeadIn]     := FALSE;
578.      CrtInfo[HeartBeat]  := ' ';         Prefixed[HeartBeat]  := FALSE;
579.      CrtInfo[StartBeat]  := ' ';         Prefixed[StartBeat]  := FALSE;
580.      CrtInfo[EchoOn]     := chr(00);     Prefixed[EchoOn]     := FALSE;
581.      CrtInfo[EchoOff]    := chr(00);     Prefixed[EchoOff]    := FALSE;
582.      CrtInfo[TypAhdOn]   := chr(00);     Prefixed[TypAhdOn]   := FALSE;
583.      CrtInfo[TypAhdOff]  := chr(00);     Prefixed[TypAhdOff]  := FALSE;
584.      CrtInfo[UcaseOn]    := chr(00);     Prefixed[UcaseOn]    := FALSE;
585.      CrtInfo[UcaseOff]   := chr(00);     Prefixed[UcaseOff]   := FALSE;
586.      CrtInfo[BsupOn]     := chr(00);     Prefixed[BsupOn]     := FALSE;
587.      CrtInfo[BsupOff]    := chr(00);     Prefixed[BsupOff]    := FALSE;
588.      CrtInfo[DefStrOn]   := chr(00);     Prefixed[DefStrOn]   := FALSE;
589.      CrtInfo[DefStrOff]  := chr(00);     Prefixed[DefStrOff]  := FALSE;
590.      CrtInfo[DefNumOn]   := chr(00);     Prefixed[DefNumOn]   := FALSE;
591.      CrtInfo[DefNumOff]  := chr(00);     Prefixed[DefNumOff]  := FALSE;
592.
593.      if ExtCRT
594.          then begin
595.
596.              CrtInfo[EraseALL]    := '+';      Prefixed[EraseALL]    := T
597.              CrtInfo[ErasEOS]     := 'Y';      Prefixed[ErasEOS]     := T
598.              CrtInfo[ErasEOL]     := 'T';      Prefixed[ErasEOL]     := T
599.
600.              CrtInfo[CursorHome]  := chr(30); Prefixed[CursorHome]  := F
601.              CrtInfo[CursorUp]    := chr(11); Prefixed[CursorUp]    := F
602.              CrtInfo[CursorDown]  := chr(10); Prefixed[CursorDown]  := F
```

```
603.                CrtInfo[CursorRight] := chr(12);  PrefixedCursorRight] := F!
604.                CrtInfo[CursorLeft]  := chr(08);  PrefixedCursorLeft]  := F!
605.                CrtInfo[CursorFtab]  := chr(09);  PrefixedCursorFtab]  := F!
606.                CrtInfo[CursorBtab]  := 'I';      PrefixedCursorBtab]  := T!
607.
608.                CrtInfo[InsertLine]  := 'E';      PrefixedInsertLine]  := T!
609.                CrtInfo[DeleteLine]  := 'R';      PrefixedDeleteLine]  := T!
610.                CrtInfo[InsertChar]  := 'Q';      PrefixedInsertChar]  := T!
611.                CrtInfo[DeleteChar]  := 'W';      PrefixedDeleteChar]  := T!
612.
613.                CrtInfo[CursorUndscr]:= chr(00);  PrefixedCursorUndscr]:= F!
614.                CrtInfo[CursorInvrse]:= chr(00);  PrefixedCursorInvrse]:= F!
615.                CrtInfo[CursorOff]   := chr(00);  PrefixedCursorOff]   := F!
616.                CrtInfo[CursorOn]    := chr(00);  PrefixedCursorOn]    := F!
617.                CrtInfo[ScrollOff]   := chr(00);  PrefixedScrollOff]   := F!
618.                CrtInfo[ScrollOn]    := chr(00);  PrefixedScrollOn]    := F!
619.                CrtInfo[PagingOff]   := chr(00);  PrefixedPagingOff]   := F!
620.                CrtInfo[PagingOn]    := chr(00);  PrefixedPagingOn]    := F!
621.                CrtInfo[WrapOff]     := chr(00);  PrefixedWrapOff]     := F!
622.                CrtInfo[WrapOn]      := chr(00);  PrefixedWrapOn]      := F!
623.                CrtInfo[InsertOff]   := chr(00);  PrefixedInsertOff]   := F!
624.                CrtInfo[InsertOn]    := chr(00);  PrefixedInsertOn]    := F!
625.
626.                CrtInfo[GrfMode]     := chr(00);  PrefixedGrfMode]     := F!
627.                CrtInfo[TxtMode]     := chr(00);  PrefixedTxtMode]     := F!
628.                CrtInfo[InvrtScreen] := chr(00);  PrefixedInvrtScreen] := F!
629.                CrtInfo[VdoNor]      := '0';      PrefixedVdoNor]      := T!
630.                CrtInfo[VdoInv]      := '4';      PrefixedVdoInv]      := T!
631.                CrtInfo[VdoNorUnd]   := '6';      PrefixedVdoNorUnd]   := T!
632.                CrtInfo[VdoInvUnd]   := '<';      PrefixedVdoInvUnd]   := T!
633.
634.                end
635.            else begin
636.
637.                CrtInfo[EraseALL]    := 'J';      PrefixedEraseALL]    := T!
638.                CrtInfo[ErasEOS]     := 'Y';      PrefixedErasEOS]     := T!
639.                CrtInfo[ErasEOL]     := 'K';      PrefixedErasEOL]     := T!
640.
641.                CrtInfo[CursorHome]  := 'H';      PrefixedCursorHome]  := T!
642.                CrtInfo[CursorUp]    := 'A';      PrefixedCursorUp]    := T!
643.                CrtInfo[CursorDown]  := 'B';      PrefixedCursorDown]  := T!
644.                CrtInfo[CursorRight] := 'C';      PrefixedCursorRight] := T!
645.                CrtInfo[CursorLeft]  := 'D';      PrefixedCursorLeft]  := T!
646.                CrtInfo[CursorFtab]  := chr(09);  PrefixedCursorFtab]  := F!
647.                CrtInfo[CursorBtab]  := 'i';      PrefixedCursorBtab]  := T!
648.
649.                CrtInfo[InsertLine]  := 'E';      PrefixedInsertLine]  := T!
650.                CrtInfo[DeleteLine]  := 'R';      PrefixedDeleteLine]  := T!
651.                CrtInfo[InsertChar]  := 'Q';      PrefixedInsertChar]  := T!
652.                CrtInfo[DeleteChar]  := 'W';      PrefixedDeleteChar]  := T!
653.
654.                CrtInfo[CursorUndscr]:= 'u';      PrefixedCursorUndscr]:= T!
655.                CrtInfo[CursorInvrse]:= 'v';      PrefixedCursorInvrse]:= T!
656.                CrtInfo[CursorOff]   := 'b';      PrefixedCursorOff]   := T!
```

```
657.                CrtInfo[CursorOn]    := 'c';     Prefixed[CursorOn]    := T!
658                 CrtInfo[ScrollOff]   := 'n';     Prefixed[ScrollOff]   := T!
659.                CrtInfo[ScrollOn]    := 's';     Prefixed[ScrollOn]    := T!
660.                CrtInfo[PagingOff]   := 'y';     Prefixed[PagingOff]   := T!
661.                CrtInfo[PagingOn]    := 'a';     Prefixed[PagingOn]    := T!
662                 CrtInfo[WrapOff]     := 'x';     Prefixed[WrapOff]     := T!
663.                CrtInfo[WrapOn]      := 'w';     Prefixed[WrapOn]      := T!
664                 CrtInfo[InsertOff]   := 'r';     Prefixed[InsertOff]   := T!
665.                CrtInfo[InsertOn]    := 'q';     Prefixed[InsertOn]    := T!
666.
667                 CrtInfo[GrfMode]     := 'g';     Prefixed[GrfMode]     := T!
668                 CrtInfo[TxtMode]     := 't';     Prefixed[TxtMode]     := T!
669                 CrtInfo[InvrtScreen] := 'z';     Prefixed[InvrtScreen] := T!
670.                CrtInfo[VdoNor]      := '0';     Prefixed[VdoNor]      := T!
671.                CrtInfo[VdoInv]      := '4';     Prefixed[VdoInv]      := T!
672.                CrtInfo[VdoNorUnd]   := '8';     Prefixed[VdoNorUnd]   := T!
673.                CrtInfo[VdoInvUnd]   := '<';     Prefixed[VdoInvUnd]   := T!
674.
675.                end;
676.        end;
677.
678. end.
679
```

WRAPON	57	622	663							
WS	555	572	574							
X	112	160	161	223	234	236	251	252	267	268
XHOME	554									
XLEN	554	574								
Y	112	160	161							
YHOME	554									
YLEN	554	574								

```
1.  { CCDCPIO.TEXT ----------------------------------------------------------}
2.  {
3.  {       CCDCPIO -- Corvus CONCEPT DataCom and Printer I/O Unit
4.  {
5.  {       Copyright 1983 Corvus Systems, Inc.
6.  {                      San Jose, California
7.  {
8.  {       All Rights Reserved
9.  {
10. {       v 1.0  04-08-82  MB   Original unit (was CCprtIO)
11. {       v 2.0  12-10-82  KB   Updated to new functions and datacom added
12. {
13. {-----------------------------------------------------------------------}
14. {$R-}
15.
16. UNIT CCdcpIO;
17.
18. INTERFACE
19.
20. USES {$U /CCUTIL/CCLIB} CCdefn;
21.
22. CONST  { UnitStatus function codes }
23.        { not used by this unit }
24.
25.        {Printer driver}
26.        FCMODECHG     = $80; {toggle transparent/translate mode}
27.        FCINSTALT     = $81; {install alt char translate table}
28.        FCATTCHPR     = $82; {attach printer to unit}
29.        FCSLCTPITCH   = $83; {select pitch - 10 or 12}
30.        FCSLCTINCH    = $84; {select lines per inch - 6 or 8}
31.        FCINSTACT     = $85; {install printer action table}
32.        FCCLPISTAT    = $86; {return state of CPI and LPI}
33.
34.        {DataCom driver}
35.        FCRDSTATUS    = $07; {read buffer status}
36.        FCWRSTATUS    = $08; {write buffer status}
37.        FCSETHIWATER  = $09; {set hi water mark for read buffer}
38.        FCSETLOWATER  = $0A; {set low water mark for read buffer}
39.        FCRDOUTDSBL   = $0B; {toggle read buffer output disable - BUFFER !
40.        FCRDINDSBL    = $0C; {toggle read buffer input disable - PORT TO !
41.        FCWROUTDSBL   = $0D; {toggle write buffer output disable - BUFFER!
42.        FCWRINDSBL    = $0E; {toggle write buffer input disable - USER TO!
43.        FCWRBUFCHRS   = $0F; {get the number of characters in the write b!
44.        FCRDBUFCHRS   = $10; {get the number of characters in the read bu!
45.        FCAUTOLF      = $11; {toggle the forced auto line feed flag}
46.        FCBTWNENQ     = $12; {set the number of chars between ENQ's or ET!
47.        FCRDALTBUF    = $13; {set an alternate read buffer}
48.        FCWRALTBUF    = $14; {set an alternate write buffer}
49. {$P}
```

```
50.         { baud rate codes }
51.         BAUD300  = 0;
52.         BAUD600  = 1;
53.         BAUD1200 = 2;
54.         BAUD2400 = 3;
55.         BAUD4800 = 4;              { default }
56.         BAUD9600 = 5;
57.         BAUD19200 = 6;
58.
59.         { parity codes }
60.         PARDISABLED = 0;           { default }
61.         PARODD      = 1;
62.         PAREVEN     = 2;
63.         PARMARKXNR  = 3;
64.         PARSPACEXNR = 4;
65.
66.         { printer port select codes }
67.         PORT1       = 0;
68.         PORT2       = 1;           { default }
69.
70.         { word size (charsize) codes }
71.         CHARS78     = 0;           { default }
72.         CHARS77     = 1;
73.
74.         { handshake codes }
75.         LINECTSINVERTED = 0;
76.         LINECTSNORMAL   = 1;
77.         LINEDSRINVERTED = 2;
78.         LINEDSRNORMAL   = 3;       { default }
79.         LINEDCDINVERTED = 4;
80.         LINEDCDNORMAL   = 5;
81.         XONXOFF         = 6;
82.         ENQACK          = 7;
83.         ETXACK          = 8;       {new protocol}
84.         NOPROTOCOL      = 9;       {new protocol}
85.
86.         { unit number codes }
87.         PRINTERUNIT     = 0;
88.         DTACOM1UNIT     = 1;
89.         DTACOM2UNIT     = 2;
90.         DCPINVUNITNO    = -1;
91.
92.   {$P}
```

```
 93  TYPE
 94        WrBufStatus = RECORD
 95                        BufferSize : INTEGER;
 96                        FreeSpace  : INTEGER;
 97                        ChrBtwnENQ : INTEGER;
 98                        InputDisbld: BOOLEAN;
 99                        OutputDsbld: BOOLEAN;
100                        AutoLinFeed: BOOLEAN;
101                        AltBufAvail: BOOLEAN;
102                        AltBufAddr : pByte;
103                        AltBufSize : INTEGER;
104                        END;
105
106        RdBufStatus = RECORD
107                        BufferSize : INTEGER;
108                        FreeSpace  : INTEGER;
109                        HiWater    : INTEGER;
110                        LowWater   : INTEGER;
111                        InputDisbld: BOOLEAN;
112                        OutputDsbld: BOOLEAN;
113                        LostData   : BOOLEAN;
114                        AltBufAvail: BOOLEAN;
115                        AltBufAddr : pByte;
116                        AltBufSize : INTEGER;
117                        END;
118
119        PrtStatusBlk = RECORD
120                         CPI : INTEGER;
121                         LPI : INTEGER;
122                         END;
123
124  VAR   PrtAvail: boolean;   { printer available (assigned) }
125        DC1Avail: boolean;   { datacom 1 available (assigned) }
126        DC2Avail: boolean;   { datacom 2 available (assigned) }
127        PRT: integer;        { unit number of /Printer }
128        DC1: integer;        { unit number of /Dtacom1 }
129        DC2: integer;        { unit number of /Dtacom2 }
130
131  FUNCTION  DCPStatus     (var br,par,dc,chsz,hs: integer): integer;
132  FUNCTION  DCPPwrFree    (var freebytes: integer):         integer;
133  FUNCTION  DCPPrdFree    (var freebytes: integer):         integer;
134  FUNCTION  DCPBaudRate   (baudrate: integer):              integer;
135  FUNCTION  DCPParity     (parity: integer):                integer;
136  FUNCTION  DCPCharSize   (charsize: integer):              integer;
137  FUNCTION  DCPHandShake  (protocol: integer):              integer;
138  FUNCTION  DCPGetUnitNo:                                   integer;
139  FUNCTION  DCPSetUnitNo (unitno: integer):                 integer;
140  FUNCTION  PrtDataCom    (port: integer):                  integer;
141  FUNCTION  DCPRdStatus   (var RDst: RdBufStatus):          integer;
142  FUNCTION  DCPWrStatus   (var WRst: WrBufstatus):          integer;
143  FUNCTION  DCPAutoLF:                                      integer;
144  FUNCTION  PrtTblStatus (var ChrInch,LinesInch: integer): integer;
145
146  PROCEDURE CCdcpIOinit;
```

```
147.
148.  IMPLEMENTATION
149.
150.  CONST
151.                { UnitStatus function codes }
152.       FWRFREE    = 0;       {new - write buffer free space}
153.       FBAUDRATE  = 1;
154.       FPARITY    = 2;
155.       FDATACOM   = 3;       {new - printer only}
156.       FCHARSIZE  = 4;
157.       FHANDSHAKE= 5;
158.       FSTATUS    = 6;
159.
160.       FRDFREE    = 3;       {new - read buffer free space, datacoms only}
161.
162.  VAR    DCPunitno: integer; { current unit number }
163.  {$P}
```

```
164  FUNCTION pOSdevNam (untnbr: integer): pString64;          external;
165  FUNCTION OSprtrDv: integer;                               external;
166  FUNCTION OSdcm1Dv: integer;                               external;
167  FUNCTION OSdcm2Dv: integer;                               external;
168
169  FUNCTION GetError: integer;
170      begin
171      if      DCPunitno = PRT then GetError := IOEnoprt
172      else if DCPunitno = DC1 then GetError := IOEnodtc
173      else if DCPunitno = DC2 then GetError := IOEnodtc
174      else                         GetError := IOEinvdev;
175      end;
176
177  FUNCTION GotDevice(var ior : integer): boolean;
178      var devavail  boolean;
179      begin
180      ior := 0;
181      if      DCPunitno = PRT then devavail := PrtAvail
182      else if DCPunitno = DC1 then devavail := DC1Avail
183      else if DCPunitno = DC2 then devavail := DC2Avail
184      else                         devavail := FALSE;
185      if Not devavail then ior := GetError;
186      GotDevice := devavail;
187      end;
188
189  FUNCTION DCPStatus; {(var br,par,dc,chsz,hs: integer);}
190      type statusblock = record
191           baudrate,parity,port,charsize,handshake: integer; end;
192      var stb: statusblock;
193          ior: integer;
194      begin
195      if GotDevice(ior) then begin
196          UnitStatus (DCPunitno,stb,FSTATUS);
197          ior := IORESULT;
198          if ior = 0 then with stb do begin
199              br := baudrate; par := parity; dc := port;
200              chsz := charsize; hs := handshake; end;
201          end;
202      DCPStatus := ior;
203      end;
204
205  FUNCTION DCPwrFree; {(var freebytes: integer): integer;}
206      var ior: integer;
207      begin
208      if GotDevice(ior) then begin
209          UnitStatus (DCPunitno,freebytes,FWRFREE);
210          ior := IORESULT;
211          end;
212      DCPwrFree := ior;
213      end;
214  {$P}
```

```
215. FUNCTION DCPrdFree; {(var freebytes: integer): integer;}
216.     var ior: integer;
217.     begin
218.     if DCPunitno = PRT then ior := IOEinvdev
219.     else if GotDevice(ior) then begin
220.             UnitStatus (DCPunitno,freebytes,FRDFREE);
221.             ior := IORESULT;
222.             end;
223.     DCPrdFree := ior;
224.     end;
225.
226. FUNCTION DCPBaudRate; {(baudrate: integer): integer;}
227.     var ior: integer;
228.     begin
229.     if GotDevice(ior) then begin
230.         UnitStatus(DCPunitno,baudrate,FBAUDRATE);
231.         ior := IORESULT;
232.         end;
233.     DCPBaudRate := ior;
234.     end;
235.
236. FUNCTION DCPParity; {(parity: integer): integer;}
237.     var ior: integer;
238.     begin
239.     if GotDevice(ior) then begin
240.         UnitStatus(DCPunitno,parity,FPARITY);
241.         ior := IORESULT;
242.         end;
243.     DCPParity := ior;
244.     end;
245.
246. FUNCTION PrtDataCom; {(port: integer): integer;}
247.     begin
248.     if PrtAvail then begin
249.         UnitStatus (PRT,port,FDATACOM);
250.         PrtDataCom := IORESULT;
251.         end
252.     else PrtDataCom := IUEnoprt;
253.     end;
254.
255. FUNCTION DCPCharSize; {(charsize integer): integer;}
256.     var ior: integer;
257.     begin
258.     if GotDevice(ior) then begin
259.         UnitStatus(DCPunitno,charsize,FCHARSIZE);
260.         ior := IORESULT;
261.         end;
262.     DCPCharSize := ior;
263.     end;
264.
265. FUNCTION DCPHandShake; {(protocol: integer): integer;}
266.     var ior: integer;
267.     begin
268.     if GotDevice(ior) then begin
```

```
269             UnitStatus(DCPunitno,protocol,FHANDSHAKE);
270             ior := IORESULT;
271             end;
272         DCPHandShake := ior;
273         end;
274     {$P}
```

```
275.  FUNCTION PrtTblStatus;{(var ChrInch,LinesInch: integer):integer;}
276.      var ior: integer;
277.          pb: PrtStatusBlk;
278.      begin
279.      if DCPunitno <> PRT then ior := DCPINVUNITNO
280.      else if GotDevice(ior) then begin
281.             UnitStatus(DCPunitno,pb,FCCLPISTAT);
282.             ior := IORESULT;
283.             if ior = 0 then
284.                begin
285.                ChrInch := pb.CPI;
286.                LinesInch := pb.LPI;
287.                end;
288.             end;
289.      PrtTblStatus := ior;
290.      end;
291.
292.  FUNCTION DCPRdStatus;{(var RDst: RdBufStatus):integer;}
293.      var ior: integer;
294.      begin
295.      if DCPunitno = PRT then ior := DCPINVUNITNO
296.      else if GotDevice(ior) then begin
297.             UnitStatus(DCPunitno,RDst,FCRDSTATUS);
298.             ior := IORESULT;
299.             end;
300.      DCPRdStatus := ior;
301.      end;
302.
303.  FUNCTION DCPWrStatus;{(var WRst: WrBufStatus):integer;}
304.      var ior: integer;
305.      begin
306.      if GotDevice(ior) then begin
307.          UnitStatus(DCPunitno,WRst,FCWRSTATUS);
308.          ior := IORESULT;
309.          end;
310.      DCPWrStatus := ior;
311.      end;
312.
313.  FUNCTION DCPAutoLF;{:integer;}
314.      var ior: integer;
315.      begin
316.      if GotDevice(ior) then begin
317.          UnitStatus(DCPunitno,ior,FCAUTOLF);
318.          ior := IORESULT;
319.          end;
320.      DCPAutoLf := ior;
321.      end;
322.
323.  FUNCTION  DCPGetUnitNo; {: integer;}
324.      begin
325.      if      DCPunitno = PRT then DCPGetUnitNo := PRINTERUNIT
326.      else if DCPunitno = DC1 then DCPGetUnitNo := DTACOM1UNIT
327.      else if DCPunitno = DC2 then DCPGetUnitNo := DTACOM2UNIT
328.      else                         DCPGetUnitNo := DCPINVUNITNO;
```

```
329.      end;
330.
331.  FUNCTION DCPSetUnitNo; {(unitno: integer): integer;}
332.      var ior,SVunitno: integer;
333.          bad: boolean;
334.      begin
335.      bad := false;
336.      SVunitno := DCPunitno;
337.      case unitno of
338.          PRINTERUNIT: DCPunitno := PRT;
339.          DTACOM1UNIT: DCPunitno := DC1;
340.          DTACOM2UNIT: DCPunitno := DC2;
341.          otherwise: bad := true;
342.          end;
343.      if bad then ior := IOEinvdev
344.              else if NOT GotDevice(ior) then DCPunitno := SVunitno;
345.      DCPSetUnitNo := ior;
346.      end;
347.
348.  PROCEDURE CCdcpIOinit;
349.      var pIDptr: pString64; i: integer;
350.      begin
351.      PRT := OSprtrDv;    { unit number of /Printer }
352.      DC1 := OSdcm1Dv;    { unit number of /Dtacom1 }
353.      DC2 := OSdcm2Dv;    { unit number of /Dtacom2 }
354.      DCPunitno := PRT;   { default unit is printer }
355.      pIDptr := pOSdevNam (PRT); PrtAvail := (pIDptr^ = 'PRINTER');
356.      pIDptr := pOSdevNam (DC1); DC1Avail := (pIDptr^ = 'DTACOM1');
357.      pIDptr := pOSdevNam (DC2); DC2Avail := (pIDptr^ = 'DTACOM2');
358.      if DC1Avail then begin
359.          UnitStatus (DC1,i,FWRFREE);
360.          if IORESULT <> 0 then begin
361.              PrtAvail := FALSE;
362.              DC1Avail := FALSE;
363.              DC2Avail := FALSE;
364.              end;
365.          end;
366.      end;
367.
368.  END.    {CCdcpIO}
369.
370.
```

```
{ CCDIRIO.TEXT ------------------------------------------------------------}
{
{       CCDIRIO -- Corvus CONCEPT Volume Directory Unit
{
{       (c) Copyright 1982 Corvus Systems, Inc.
{                          San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  10-06-82  LEF  Original unit
{
{-------------------------------------------------------------------------}
{$R-}

UNIT CCdirIO;

INTERFACE

CONST
        BlockSize = 512;
        VIDlength = 7;
        TIDlength = 15;
        MaxDirEnt = 77;

TYPE
        dirrange = 0..MaxDirEnt;
        vid      = string[VIDlength];
        tid      = string[TIDlength];
        filekind = (UNTYPEDFILE, XDSKFILE, CODEFILE, TEXTFILE, INFOFILE,
                    DATAFILE, GRAFFILE, FOTOFILE, SECURDIR);

        daterec  = packed record
                     year:  0..100;  { 100 = temp file flag }
                     day:   0..31;
                     month: 0..12;   { 0 = date not meaningful }
                     end;
{$P}
```

```
38.       direntry = packed record
39.                    firstblock: integer;
40.                    nextblock: integer;
41.                    MarkBit: Boolean;
42.                    filler: 0..2047;
43.                    case fkind: filekind of
44.                      SECURDIR,
45.                      UNTYPEDFILE:
46.                         (dvid: vid;              { Disk volume name }
47.                          deovblock: integer;     { Last block of volume }
48.                          dnumfiles: integer;     { Number of files }
49.                          dloadtime: integer;     { Time of last access }
50.                          dlastboot: daterec;     { Most recent date setting }
51.                          MemFlipped: Boolean;    { TRUE if flipped in memory }
52.                          DskFlipped: Boolean);   { TRUE if flipped on disk }
53.                      XDSKFILE,CODEFILE,TEXTFILE,INFOFILE,
54.                      DATAFILE,GRAFFILE,FOTOFILE:
55.                         (dtid: tid;              { Title of file }
56.                          dlastbyte: 1..BlockSize; { Bytes in last block }
57.                          daccess: daterec);      { Last modification date }
58.                    end;
59.
60.       directory = array [dirrange] of direntry;
61.
62.  PROCEDURE CCdirIOinit;
63.
64.  PROCEDURE GetVolDir (     fvid: vid;
65.                       var fdir: directory;
66.                       var DevBlocked: Boolean;
67.                       var fdevno: integer;
68.                       var DevValid: Boolean);
69.
70.  PROCEDURE PutVolDir (var fdir: directory;
71.                           fdevno: integer);
72.
73.  IMPLEMENTATION
74.
75.  {$P}
```

```
   76. PROCEDURE xgetdir (fvid: vid;
   77.                    var fdir: directory;
   78.                    var DevBlocked: Boolean;
   79.                    var fdevno: integer;
   80.                    var DevValid: Boolean);                    externa!
   81.
   82. PROCEDURE xputdir (var fdir: directory; fdevno: integer);     externa!
   83.
   84. PROCEDURE GetVolDir ((    fvid: vid;
   85.                         var fdir: directory;
   86.                         var DevBlocked: Boolean;
   87.                         var fdevno: integer;
   88.                         var DevValid: Boolean));
   89.       begin
   90.       xgetdir (fvid,fdir,DevBlocked,fdevno,DevValid);
   91.       end;
   92.
   93. PROCEDURE PutVolDir ((var fdir: directory;
   94.                            fdevno: integer));
   95.       begin
   96.       xputdir (fdir,fdevno);
   97.       end;
   98.
   99. PROCEDURE CCdirIOinit;
  100.       begin end;
  101.
  102. end.
  103.
```

VID	27	46	64	76
VIDLENGTH	21	27		
XDSKFILE	29	53		
XGETDIR	76	90		
XPUTDIR	82	96		
YEAR	33			

```
 1. { CCGRFIO.TEXT ------------------------------------------------------}
 2. {
 3. {       CCGRFIO -- Corvus CONCEPT Graphics Support Unit
 4. {
 5. {       (c) Copyright 1982 Corvus Systems, Inc.
 6. {                          San Jose, California
 7. {
 8. {       All Rights Reserved
 9. {
10. {       v 1.0  04-10-82  MB   Original unit
11. {       v 1.1  05-13-82  MB   WriteBytes, ReadBytes now UnitStatus calls
12. {
13. {------------------------------------------------------------------------}
14  {$R-}
15.
16. UNIT CCgrfIO;
17.
18. INTERFACE
19.
20. USES {$U CCL1B} CCdefn;
21.
22. CONST
23.     GrfMwhite =  1; { mode values }
24.     GrfMblack =  0;
25      GrfMflip  = -1;
26.
27.     GrfQgrRel =  1; { qual values }
28.     GrfQgrAbs =  2;
29.     GrfQchRel =  3;
30.     GrfQchAbs =  4;
31.
32. PROCEDURE CCgrfIOinit;
33. PROCEDURE SetOrigin  (x,y,qual:              integer);
34. PROCEDURE PlotPoint  (x,y,mode:              integer);
35. PROCEDURE DrawLine   (x1,y1,x2,y2,mode:      integer);
36. PROCEDURE FillBox    (x1,y1,wd,ht,density:  integer);
37. PROCEDURE CopyBox    (x1,y1,wd,ht,x2,y2:    integer);
38. PROCEDURE WriteBytes (count: integer; pBuff: pBytes);
39. PROCEDURE ReadBytes  (count: integer; pBuff: pBytes);
40.
41. IMPLEMENTATION
42.
43. {$P}
```

```
44  CONST ESC = $1B;
45        WRBYTES = 6; RDBYTES = 7;   { UnitStatus functions }
46
47  TYPE
48        graphbuffer = record case integer of
49                        0: (pi: array [1..10] of integer);
50                        1: (pb: array [1..20] of byte);
51                        end;
52        wrbuffer  =     record
53                        bytecount: integer;
54                        buffptr:   pBytes;
55                        end;
56
57  VAR   DisplayDrv: integer;
58        buf:        graphbuffer;
59        wbuf:       wrbuffer;
60        b:          byte;
61
62  FUNCTION OSdispDv  integer; external;
63
64  PROCEDURE SetOrigin; { (x,y,qual: integer) }
65        begin
66        buf.pb[1] := ESC;   buf.pb[2] := ord('o');
67        buf.pi[2] := x;     buf.pi[3] := y;
68        buf.pb[7] := qual mod 128;
69        unitwrite (DisplayDrv,buf,7);
70        end;
71
72  PROCEDURE PlotPoint; { (x,y,mode: integer) }
73        begin
74        buf.pb[1] := ESC;   buf.pb[2] := ord('p');
75        buf.pi[2] := x;     buf.pi[3] := y;
76        buf.pb[7] := mode mod 256;
77        unitwrite (DisplayDrv,buf,7);
78        end;
79
80  PROCEDURE DrawLine; { (x1,y1,x2,y2,mode: integer) }
81        begin
82        buf.pb[1] := ESC;   buf.pb[2] := ord('l');
83        buf.pi[2] := x1;    buf.pi[3] := y1;
84        buf.pi[4] := x2;    buf.pi[5] := y2;
85        buf.pb[11] := mode mod 256;
86        unitwrite (DisplayDrv,buf,11);
87        end;
88
89  {$P}
```

```
 90. PROCEDURE FillBox; { (x1,y1,wd,ht,density: integer) }
 91.       begin
 92.       buf.pb[1] := ESC;   buf.pb[2] := ord('f');
 93.       buf.pi[2] := x1;    buf.pi[3] := y1;
 94.       buf.pi[4] := ht;    buf.pi[5] := wd;
 95.       buf.pb[11] := density mod 256;
 96.       unitwrite (DisplayDrv,buf,11);
 97.       end;
 98.
 99. PROCEDURE CopyBox; { (x1,y1,wd,ht,x2,y2: integer) }
100.       begin
101.       buf.pb[1] := ESC;   buf.pb[2] := ord('m');
102.       buf.pi[2] := x1;    buf.pi[3] := y1;
103.       buf.pi[4] := ht;    buf.pi[5] := wd;
104.       buf.pi[6] := x2;    buf.pi[7] := y2;
105.       unitwrite (DisplayDrv,buf,14);
106.       end;
107.
108. PROCEDURE WriteBytes; { (count: integer; pBuff: pBytes) }
109.       begin
110.       wbuf.bytecount := count;
111.       wbuf.buffptr   := pBuff;
112.       unitstatus (DisplayDrv,wbuf,WRBYTES);
113.       end;
114.
115. PROCEDURE ReadBytes; { (count: integer; pBuff: pBytes) }
116.       begin
117.       wbuf.bytecount := count;
118.       wbuf.buffptr   := pBuff;
119.       unitstatus (DisplayDrv,wbuf,RDBYTES);
120.       end;
121.
122. PROCEDURE CCgrfIOinit;
123.       begin DisplayDrv := OSdispDv; end;
124.
125. END.  { Unit CCgrfIO }
```

WRBUFFER	52	59				
WRBYTES	45	112				
WRITEBYTES	38	108				
X	33	34	67	75		
X1	35	36	37	83	93	102
X2	35	37	84	104		
Y	33	34	67	75		
Y1	35	36	37	83	93	102
Y2	35	37	84	104		

```
    1. { CCLBLIO.TEXT ------------------------------------------------------}
    2. {
    3. {       CCLBLIO -- Corvus CONCEPT Label Processing Unit
    4. {
    5. {       (c) Copyright 1982 by Corvus Systems, Inc.
    6. {                             San Jose, California
    7. {
    8. {       All Rights Reserved
    9. {
   10. {       v 1.0  04-01-82  KB   Original unit
   11. {       v 1.1  07-09-82  LEF  Function labels expanded to 8 characters
   12. {       v 1.2  01-11-83  LEF  Add conditionals for p-System
   13. {
   14. {!CC}{ Corvus CONCEPT version
   15. {------------------------------------------------------------------------}
   16. {$R-}
   17.
   18. UNIT CClblIO;
   19.
   20. INTERFACE
   21.
   22. TYPE
   23.       LblKeyStr = string[8];
   24.       LblRtnStr = string[16];
   25.
   26. PROCEDURE CClblIOinit;
   27. PROCEDURE CClblIOterm;
   28. PROCEDURE LblsInit;
   29. PROCEDURE LblsOn;
   30. PROCEDURE LblsOff;
   31. FUNCTION  LblSet (KN: integer; LblStr: LblKeyStr;
   32.                                 RetStr: LblRtnStr): integer;
   33.
   34. IMPLEMENTATION
   35.
   36. {$P}
```

```
37.  CONST
38.  {!CC} Init    = $FF; { initialize labels function code      }
39.  {!CC} SetKey  = $FE; { set label table entry function code  }
40.  {!CC} TurnOff = $FD; { turn off labels function code        }
41.  {!CC} TurnOn  = $FC; { turn on labels function code         }
42.
43.  TYPE  lblchs     = packed array [1..8] of char;
44.        LblPblock = record
45.                      KeyNumber: integer;
46.                      Lblch:     lblchs;
47.                      ReturnStr: LblRtnStr;
48.                      end;
49.
50.  VAR   Systerm: integer; { unit number of label manager systerm }
51.
52.  FUNCTION OSstrmDv: integer; external;
53.
54.  PROCEDURE LblsInit;
55.        var SKParmBlock: LblPblock; i: integer;
56.        begin
57.        UnitStatus (Systerm,i,TurnOff); {function uses NO ParameterBlock}
58.        with SKParmBlock do begin       {initialize all labels to blanks}
59.            for i := 1 to 8 do Lblch[i] := ' '; ReturnStr := '';
60.            for i := 0 to 39 do begin
61.                KeyNumber := i;
62.                UnitStatus (Systerm,SKParmBlock,SetKey);
63.                end; {for}
64.            end; {with}
65.        end;
66.
67.  PROCEDURE LblsOn;
68.        var i : integer;
69.        begin UnitStatus (Systerm,i,TurnOn); end;
70.
71.  PROCEDURE LblsOff;
72.        var i : integer;
73.        begin UnitStatus (Systerm,i,TurnOff); end;
74.
75.  {$P}
```

```
 76. FUNCTION LblSet {(KN: integer; LblStr: LblKeyStr;
 77.                                 RetStr: LblRtnStr): integer};
 78.       { returns IORESULT }
 79.       var SKParmBlock: LblPblock; i: integer;
 80.       begin
 81.       UnitStatus (Systerm,i,TurnOff); {function uses NO ParameterBlock}
 82.       with SKParmBlock do begin
 83.           KeyNumber := KN;
 84.           for i := 1 to 8 do
 85.               if i > length(LblStr) then Lblch[i] := ' '
 86.                                     else Lblch[i] := LblStr[i];
 87.           Returnstr := RetStr;
 88.           end;
 89.       UnitStatus (Systerm,SKParmBlock,SetKey);
 90.       LblSet := IORESULT;
 91.       end;
 92.
 93. PROCEDURE CClblIOinit;
 94.       begin Systerm := OSstrmDv; LblsInit; end;
 95.
 96. PROCEDURE CClblIOterm;
 97.       begin
 98.       LblsInit;
 99.       end;
100.
101. END.  { Unit CClblIO }
102.
```

```
{ CCOMNIO.TEXT ------------------------------------------------------------}
{
        CCOMNIO -- OMNINET Commands Unit for Corvus CONCEPT
{
{       (c) Copyright 1982 Corvus Systems, Inc.
                           San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  01-09-82  PHB  Original unit
{       v 1.1  05-15-82  LEF  CComnIO unit modifications
{       v 1.2  10-27-82  LEF  OCsndMesg and OCsetRecv call modifications
{
{ Purpose:      This UNIT contains procedures which construct
{       Omninet commands and send them to the Transporter.
{       It also defines constants and data structures which are
{       useful when programming an Omninet application.
{       Hopefully, a Pascal programmer can use this UNIT without
{       knowing the details of the Transporter interface ...
{
{---------------------------------------------------------------------------}
{$R-}

UNIT CComnIO;

INTERFACE

USES {$U CCLIB} CCdefn;

CONST
    { Transporter Return Codes }
    Waiting = $FF;
    CmdAcpt = $FE;    { command accepted                                    }
    Echoed  = $C0;    { echo command was successful                         }

    GaveUp  = $80;    { aborted a send command after MaxRetries tries       }
    TooLong = $81;    { last message sent was too long for the receiver      }
    NoSockt = $82;    { sent to an uninitialized socket                     }
    HdrErr  = $83;    { sender's header length did not match receiver's     }
    BadSock = $84;    { invalid socket number                               }
    Inuse   = $85;    { tried to set up a receive on an active socket       }
    BadDest = $86;    { sent to an invalid host number                      }

    OkCode  = 0;      { success!!!                                          }

    NoTrans = -1;     { indicates that we are unable to communicate         }
                      { with Transporter - strobit failed                   }

{$P}
```

```
50.      { Transporter Opcodes }
51.      RecvOp   = $F0;    { SETUPRECV opcode }
52.      SendOp   = $40;    { SENDMSG opcode   }
53.      InitOp   = $20;    { INIT opcode      }
54.      EndOp    = $10;    { ENDRECV opcode   }
55.      DebOp    = $08;    { PEEK/POKE opcode }
56.      EchoOp   = $02;    { ECHOCMD opcode   }
57.      WhoOp    = $01;    { WHOAMI opcode    }
58.
59.  TYPE
60.      pOCrsltRcd = ^OCrsltRcd;
61.      OCrsltRcd  = RECORD
62.                     Rcode: byte;
63.                     Sourc: byte;
64.                     Len:   integer;
65.                     UCdta: array [0..255] of byte;
66.                     END;
67.
68.  VAR
69.      OCresult: integer;     { similar to IORESULT in UCSD Pascal,          '
70.                             { may be checked after each Transporter comma'
71.      OCrslt:   OCrsltRcd;   { result record which is used for all command'
72.                             { except OCsndMesg and OCsetRecv ...           '
73.      OCcurBP:  pBytes;      { current buffer pointer                       '
74.      OCcurRP:  pOCrsltRcd;  { current result pointer                       '
75.
76.  PROCEDURE COmnIOinit;
77.  PROCEDURE OCsndMesg    (bp: pBytes; rp: pOCrsltRcd; sn,dln,hln,dst: inte'
78.  PROCEDURE OCsetRecv    (bp: pBytes; rp: pOCrsltRcd; sn,dln,hln: integer);
79.  PROCEDURE OCendRecv    (sn: integer);
80.  PROCEDURE OCinitTrans;
81.  PROCEDURE OCechoTrans (dest: integer);
82.  FUNCTION  OCpeekTrans (adr: integer): byte;
83.  PROCEDURE OCpokeTrans (adr: integer; val: byte);
84.  PROCEDURE OCwhoAmI;
85.
86.  IMPLEMENTATION
87.
88.  {$P}
```

```
 89  CONST
 90      RdyAdr = $30F7F;   { address of VIA register A, used for OMNINET rea
 91      StrAdr = $30FA1;   { address of Transporter register }
 92
 93      PeekOp = $00;
 94      PokeOp = $FF;
 95
 96      { offsets into command record for byte fields }
 97      Op      = 1;    { opcode }
 98      Sock    = 5;    { socket number }
 99      HLen    = 11;   { header length }
100      Dest    = 12;   { destination for sends }
101      EDst    = 5;    { destination for echo commands }
102      PePo    = 7;    { peek/poke designator for Deb commands }
103      PoVal   = 8;    { Poke value }
104      PAdr    = 5;    { Transporter Address to peek or poke }
105
106  TYPE
107      pOmniCmd = ^OmniCmd;
108
109      OmniCmd = RECORD
110                  CASE integer OF
111                      1: (P: RECORD
112                               RP: pOCrsltRcd;
113                               DP: pBytes;
114                               LN: integer;
115                               HL: integer;
116                               end);
117                      2: (A: array [1..12] of byte);
118                  END;
119
120      TrixRcd = RECORD
121                  CASE integer OF
122                      1: (LNG: longint);
123                      2: (PTR: ^byte);
124                      3: (CPT: pOmniCmd);
125                      4: (RPT: pOCrsltRcd);
126                      5: (ARR: array [0..3] of byte);
127                  END;
128
129  VAR
130      ocmd:       OmniCmd;     { the command record used for all commands }
131      trslt:      OCrsltRcd;
132      strobeadr:  TrixRcd;
133      readyadr:   TrixRcd;
134      cmdadr:     TrixRcd;
135      transrp:    pOCrsltRcd;  { result pointer used for short commands }
136
137  {$P}
```

```
{-------------------------------------------------------------------------}
{ ready -                                                                 }
{-------------------------------------------------------------------------}

FUNCTION ready: boolean;
    var i: byte; j: integer;
    begin
    j := 10000;
    repeat
        i := readyadr.PTR^;
        j := j-1;
        until (j = 0) or (ODD (i));
    ready := ODD (i);
    end;

{-------------------------------------------------------------------------}
{ unsigned - convert byte to unsigned integer                             }
{-------------------------------------------------------------------------}

FUNCTION unsigned (b: byte): integer;
    begin
    if b < 0 then unsigned := b + 256
             else unsigned := b;
    end;

{-------------------------------------------------------------------------}
{ strobit - strobe command address to Transporter                         }
{-------------------------------------------------------------------------}

FUNCTION strobit: boolean;
    var i: integer; isready: boolean;
    begin
    i := 1;
    repeat
        isready := ready;
        if isready then
          strobeadr.PTR^ := cmdadr.ARR[i];
        i := i + 1
        until (i > 3) or (NOT isready);
    strobit := isready;
    end;

{$P}
```

```
181. {-------------------------------------------------------------------------}
182. { doit - "strobes in" the command and waits for the result              }
183. { to change ..... this is used for the simple commands                 }
184. {-------------------------------------------------------------------------}
185.
186. PROCEDURE doit (cmd: byte);
187.     var j: integer;
188.     begin
189.     OCrslt.Rcode := ORD (Waiting);
190.     ocmd.P.RP := @trslt; { must load this pointer BEFORE opcode byte }
191.     ocmd.A[Op] := cmd;
192.     trslt.Rcode := -1;
193.     if strobit
194.         then begin
195.              j := 10000;
196.              repeat
197.                  j := j - 1
198.                  until (trslt.Rcode <> -1) or (j = 0);
199.              OCrslt := trslt;
200.              OCresult := unsigned (OCrslt.Rcode);
201.              end
202.         else OCresult := NoTrans;
203.     end;
204.
205. {-------------------------------------------------------------------------}
206. { cnvsock - convert socket number to Transporter socket number           }
207. {-------------------------------------------------------------------------}
208.
209. FUNCTION cnvsock (sn: integer): byte;
210.     begin
211.     case sn of
212.         1, $80: cnvsock := ORD ($80);
213.         2, $90: cnvsock := ORD ($90);
214.         3, $A0: cnvsock := ORD ($A0);
215.         4, $B0: cnvsock := ORD ($B0);
216.     {otherwise: cnvsock := ORD ($FF);}
217.             end; {case}
218.     end;
219.
220. {$P}
```

```
221.  {-------------------------------------------------------------------------}
222.  {                                                                         }
223.  { The following procedures construct Omninet commands and send           }
224.  { them to the Transporter.                                               }
225.  {                                                                         }
226.  {-------------------------------------------------------------------------}
227.
228.
229.  {--- -----------------------------------------------------------------------}
230.  { OCinitTrans - initialize Transporter and determine our host number  }
231.  {-------------------------------------------------------------------------}
232.
233.  PROCEDURE OCinitTrans;
234.      begin doit (InitOp); end;
235.
236.  {-------------------------------------------------------------------------}
237.  { OCwhoAmI - find out what this host number is                          }
238.  {-------------------------------------------------------------------------}
239.
240.  PROCEDURE OCwhoAmI;
241.      begin doit (WhoOp); end;
242.
243.  {-------------------------------------------------------------------------}
244.  { OCechoTrans - echo to specified Transporter                           }
245.  {-------------------------------------------------------------------------}
246.
247.  PROCEDURE OCechoTrans ((dest: integer));
248.      begin ocmd.ALEDst] := dest; doit (EchoOp); end;
249.
250.  {$P}
```

```
{-------------------------------------------------------------------------}
{ OCsndMesg - send a message to another host...                            }
{                                                                         }
{ ASSUMPTIONS:                                                            }
{     - the body of the message is at the memory location                 }
{       specified by bp.                                                  }
{     - the user header (if any) is at memory location                    }
{       rp+4.  (The user header is always immediately                     }
{       following the result vector, which is 4 bytes long.)              }
{     - the result vector to be modified is at rp                         }
{-------------------------------------------------------------------------}

PROCEDURE OCsndMesg ((bp: pBytes; rp: pOCrsltRcd;
                      sn,dln,hln,dst: integer));
    begin
    OCcurBP := bp; OCcurRP := rp;
    with ocmd do begin
        P.RP := OCcurRP; { must load pointers BEFORE Op and Sock fields!
        P.DP := OCcurBP;
        A[Op] := SendOp;
        A[Sock] := cnvsock (sn);
        A[HLen] := hln;
        P.LN := dln;
        A[Dest] := dst;
        end;
    OCcurRP^.Rcode := -1;  {ORD (Waiting);}
    if strobit
        then begin
            repeat until OCcurRP^.Rcode <> -1; {ORD (Waiting);}
            OCresult := unsigned (OCcurRP^.Rcode);
            end
        else OCresult := NoTrans;
    end;

{$P}
```

```
286  {-------------------------------------------------------------------------}
287  { OCsetRecv - assembles a receive command and sends it to the           }
288  {             transporter                                               }
289  {             will not return until the command has been accepted...    }
290  {-------------------------------------------------------------------------}
291
292  PROCEDURE OCsetRecv ((bp: pBytes; rp: pOCrsltRcd;
293                        sn,dln,hln: integer));
294      begin
295      OCcurBP := bp; OCcurRP := rp;
296      with ocmd do begin
297          P_DP := OCcurBP;
298          P_RP := OCcurRP;
299          P_LN := dln;
300          AEOpId := ORD (RecvOp);
301          AESock1 := cnvsock (sn);
302          AEMLen1 := hln;
303          end;
304      OCcurRP^.Rcode := -1; { ORD (Waiting); }
305      if strobit
306          then begin
307              repeat until OCcurRP^.Rcode <> -1; {ORD (Waiting);}
308              OCresult := unsigned (OCcurRP^.Rcode);
309              end
310          else OCresult := NoTrans;
311      end;
312
313
314  {-------------------------------------------------------------------------}
315  { OCendRecv - reset a setup receive                                       }
316  {-------------------------------------------------------------------------}
317
318  PROCEDURE OCendRecv ((sn: integer));
319      begin
320      ocmd.AESock1 := cnvsock (sn);
321      doit (EndOp);
322      end;
323
324  {$P}
```

```
{-------------------------------------------------------------------------}
{ OCpeekTrans - read from the RAM inside the Transporter                  }
{               if OCresult < 0 then the value returned is undefined     }
{-------------------------------------------------------------------------}

FUNCTION OCpeekTrans ((adr: integer): byte);
    var x: integer;
    begin
    with ocmd do begin
        P.RP := @trslt;
        A[Op] := DebOp;
        A[PePo] := PeekOp; { peek }
        A[PAdr] := adr DIV 256;
        A[PAdr+1] := adr MOD 256;
        end;
    trslt.Rcode := -1; { ORD (Waiting); }
    if strobit
        then begin
            x := 0;
            repeat x := x + 1 until (trslt.Rcode <> -1) or (x >= 200);
            { the peek value could be equal to Waiting !!! }
            OCrslt := trslt;
            OCresult := unsigned (OCrslt.Rcode);
            OCpeekTrans := ORD (OCresult);
            end
        else OCresult := NoTrans;
    end;

{-------------------------------------------------------------------------}
{ OCpokeTrans - write into the Transporter's RAM                          }
{-------------------------------------------------------------------------}

PROCEDURE OCpokeTrans ((adr: integer; val: byte));
    begin
    with ocmd do begin
        A[PAdr]   := adr DIV 256;
        A[PAdr+1] := adr MOD 256;
        A[PePo]   := ORD (PokeOp);
        A[PoVal]  := val;
        end;
    doit (DebOp);
    end;

{$P}
```

```
369.  {-------------------------------------------------------------------------}
370.  { OComnIOinit - initialize CComnIO unit                                   }
371.  {-------------------------------------------------------------------------}
372.
373.  PROCEDURE CComnIOinit;
374.      begin
375.      OCcurDF := NIL;
376.      readyadr.LNG := RdyAdr;
377.      strobeadr.LNG := StrAdr;
378.      cmdadr.CPT := @ocmd;
379.      transrp := @OCrslt; { is this pointer necessary?       }
380.                          { result pointer points at OCrslt }
381.      { this procedure could also initialize the Transporter and Poke }
382.      { the proper values for the Transporter parameters which have   }
383.      { values other than the default.....                            }
384.      end;
385.
386.  END.
387.
```

```
1.  { CCWNDIO.TEXT ------------------------------------------------------}
2.  {
3.  {       CCWNDIO -- Corvus CONCEPT Window Processing Unit
4.  {
5.  {       (c) Copyright 1982 by Corvus Systems, Inc.
6.  {                          San Jose, California
7.  {
8.  {       All Rights Reserved
9.  {
10. {       v 1.0  04-01-82  MB   Original unit
11. {       v 1.1  10-17-82  LEF  Minor revision
12. {       v 1.2  12-17-82  LEF  Expand window record to 48 bytes
13. {
14. {-------------------------------------------------------------------}
15. {$R-}
16.
17. UNIT CCwndIO;
18.
19. INTERFACE
20.
21. USES {$U CCL1B} CCdefn;
22.
23. CONST
24.       { attr2 flag values - add together }
25.       WfgGraf   = 2;   { graphics mode  }
26.       WfgCursOn = 4;   { cursor on      }
27.       WfgInvCur = 8;   { inverse cursor }
28.       WfgWrap   = 16;  { line wrap      }
29.       WfgScrOff = 32;  { scroll off     }
30.       WfgClrPg  = 64;  { clear page     }
31.
32.       { values of wn for WinSystem }
33.       WsysCurr  = 1;   { current process window }
34.       WsysCmd   = 2;   { cmd/msg window         }
35.       WsysRoot  = 3;   { root user window       }
36.
37. TYPE
38.       pCharSet  = ^CharSet;
39.       CharSet   =  record
40. {length offset}
41. {     4      0 }   tblloc: pBytes;  {character set data pointer}
42. {     2      4 }   lpch:   integer; {scanlines per character (assume wid!
43. {     2      6 }   bpch:   integer; {bits per character (vertical height!
44. {     2      8 }   frstch: integer; {first character code - ascii}
45. {     2     10 }   lastch: integer; {last character code - ascii}
46. {     4     12 }   mask:   longint; {mask used in positioning cells}
47. {     1     16 }   attr1:  byte;    {attributes}
48.                                     {  bit 0 = 1 - vertical orientation}
49. {     1     17 }   attr2:  byte;    {currently unused}
50. { total   18 }   end;
51.
52. {$P}
```

```
53        pWndRcd    = ^WndRcd;
54        WndRcd     =  record
55  {length offset}
56  {     4      0 }   charpt:  pCharSet;{character set record pointer}
57  {     4      4 }   homept:  pBytes;   {home (upper left) pointer}
58  {     4      8 }   curadr:  pBytes;   {current location pointer}
59  {     2     12 }   homeof:  integer;  {bit offset of home location}
60  {     2     14 }   basex:   integer;  {home x value, rel to root window}
61  {     2     16 }   basey:   integer;  {home y value, rel to root window}
62  {     2     18 }   lngthx:  integer;  {maximum x value, bits rel to window}
63  {     2     20 }   lngthy:  integer;  {maximum y value, bits rel to window}
64  {     2     22 }   cursx:   integer;  {current x value, bits rel to window}
65  {     2     24 }   cursy:   integer;  {current y value, bits rel to window}
66  {     2     26 }   bitofs:  integer;  {bit offset of current address}
67  {     2     28 }   grorgx:  integer;  {graphics - origin x, bits rel to ho!
68  {     2     30 }   grorgy:  integer;  {graphics - origin y, bits rel to ho!
69  {     1     32 }   attr1:   byte;     {inverse, underscore, insert}
70  {     1     33 }   attr2:   byte;     {v/h, graphics/char, cursor on/off,
71                                          cursor inv/underline}
72  {     1     34 }   state:   byte;     {used for decoding escape sequences}
73  {     1     35 }   rcdlen:  byte;     {window description record length}
74  {     1     36 }   attr3:   byte;     {enhanced character set attributes}
75  {     1     37 }   fill1:   byte;     {currently unused}
76  {     1     38 }   fill2:   byte;     {currently unused}
77  {     1     39 }   fill3:   byte;     {currently unused}
78  {     4     40 }   fill4:   longint;  {currently unused}
79  {     4     44 }   wwsptr:  pBytes;   {window working storage pointer}
80  { total     48 }   end;
81
82
83  PROCEDURE CCwndIOinit;
84  FUNCTION  WinSystem (     wn: integer):                                integer;
85  FUNCTION  WinSelect (var WR: WndRcd):                                  integer;
86  FUNCTION  WinDelete (var WR: WndRcd):                                  integer;
87  FUNCTION  WinCreate (var WR: WndRcd; homex,homey,
88                               width,lngth,flags: integer):              integer;
89  FUNCTION  WinClear  (var WR: WndRcd):                                  integer;
90  FUNCTION  WinStatus (var homex,homey,width,lngth,
91                               curx,cury: integer):                      integer;
92  FUNCTION  WinLoadCh (     name: string80):                             integer;
93
94  {$P}
```

```
  95. IMPLEMENTATION
  96.
  97. const
  98.         SENSE  = 0;
  99.         CREATE = 1;
 100.         DELETE = 2;
 101.         SELECT = 3;
 102.         CLEAR  = 4;
 103.         STATUS = 5;
 104.
 105. VAR    display: integer;
 106.
 107. FUNCTION OSdispDv: integer;                                          extern;
 108. FUNCTION pOScurWnd: pWndRcd;                                        extern;
 109. FUNCTION pOSsysWnd (wndnbr: integer): pWndRcd;                      extern;
 110.
 111.
 112. { WinSystem ------------------------------------------------------------}
 113. { Select system window
 114. { 0 = root, 1 = current process window, 2 = msg/cmd
 115. {------------------------------------------------------------------------}
 116.
 117. FUNCTION WinSystem; { (wn: integer) }
 118.         var iost: integer; nilptr,wptr: pWndRcd;
 119.         begin
 120.         nilptr := nil;
 121.         if wn = 0 then wn := 3;
 122.         if wn = 1
 123.             then begin
 124.                 UnitStatus (display,nilptr^,SELECT); iost := IORESULT; end
 125.             else if wn in [2..MAXWINDOW] then begin
 126.                 wptr := pOSsysWnd (wn);
 127.                 if wptr = nil
 128.                     then iost := IOEwndwn
 129.                     else begin
 130.                         UnitStatus (display,wptr^,SELECT);
 131.                         iost := IORESULT; end;
 132.                 end
 133.             else iost := IOEwndwn;
 134.         WinSystem := iost;
 135.         end;
 136.
 137.
 138. { WinSelect ----------------------------------------------------------}
 139. {----------------------------------------------------------------------}
 140.
 141. FUNCTION WinSelect; {(var WR: WndRcd)}
 142.         begin UnitStatus (display,WR,SELECT); WinSelect := IORESULT; end;
 143.
 144. {$P}
```

```
145. { WinDelete ------------------------------------------------------------}
146. {----------------------------------------------------------------------}
147.
148. FUNCTION WinDelete; {(var WR: WndRcd)}
149.       begin UnitStatus (display,WR,DELETE); WinDelete := IORESULT; end;
150.
151.
152. { WinCreate ------------------------------------------------------------}
153. {----------------------------------------------------------------------}
154.
155. FUNCTION WinCreate; {(var WR: WndRcd; homex,homey,
156.                         width,lngth,flags: integer)   integer; }
157.       var CWptr: pWndRcd;
158.       begin
159.       CWptr := pOS(urWnd);
160.       WR.basex  := homex; WR.basey  := homey;
161.       WR.lngthx := width; WR.lngthy := lngth;
162.       WR.attr1  := CWptr^.attr1 mod 2;
163.     { WR.attr2  := flags mod 128; }
164.       WR.attr2  := (flags AND $7E)+(CWptr^.attr2 AND $01);
165.       WR.charpt := CWptr^.charpt;
166.       UnitStatus (display,WR,CREATE);
167.       WinCreate := IORESULT;
168.       end;
169.
170.
171. { WinStatus ----------------------------------------------------------}
172. {----------------------------------------------------------------------}
173.
174. FUNCTION WinStatus; {(var homex,homey,width,lngth,curx,cury: integer)}
175.       var iost: integer;
176.           WS: record xhome,yhome,xlen,ylen: integer; end;
177           WC: array [0..1] of integer;
178.       begin
179.       UnitStatus (display,WS,STATUS);
180.       iost := IORESULT;
181.       if iost = 0 then begin
182.          homex := WS.xhome; homey := WS.yhome;
183.          width := WS.xlen;  lngth := WS.ylen;
184.          UnitStatus (display,WC,SENSE);
185.          iost := IORESULT;
186.          if iost = 0 then begin
187.             curx := WC[0]; cury := WC[1];
188.             end;
189.          end;
190.       WinStatus := iost;
191.       end;
192.
193. {$P}
```

```
194.  { WinClear ------------------------------------------------------------}
195.  {--------------------------------------------------------------------}
196.
197.  FUNCTION WinClear; {(var WR: WndRcd)}
198.        begin UnitStatus (display,WR,CLEAR); WinClear := IORESULT; end;
199.
200.
201.  { WinLoadCh -----------------------------------------------------------}
202.  {--------------------------------------------------------------------}
203.
204.  FUNCTION WinLoadCh; {(name: string80): integer;}
205.        type    str64 = string[64];
206.              pstr64 = ^str64;
207.              strtbl = array [1..100] of pstr64;
208.             pstrtbl = ^strtbl;
209.        var   result: integer;
210.              s1,s2: str64; p1,p2: pstr64; p: pstrtbl;
211.        begin
212.        p := @p1; p1 := @s1; p2 := @s2;
213.        s1 := 'CSDISP'; s2 := name;
214.        WinLoadCh := call (''CC.WNDMGR',input,output,p^,2);
215.        end;
216.
217.
218.  { CCwndIOinit ---------------------------------------------------------}
219.  { Unit initialization
220.  {--------------------------------------------------------------------}
221.
222.  PROCEDURE CCwndIOinit;
223.        begin display := OSdispDv; end;
224.
225.  END.   { Unit CCwndIO }
226.
227.
```

WSYSCURR	33	
WSYSROOT	35	
WWSPTR	79	
XHOME	176	182
XLEN	176	183
YHOME	176	182
YLEN	176	183

```
 1. { TURTLE.TEXT ------------------------------------------------------------}
 2. {
 3. {       TURTLE -- Corvus CONCEPT TurtleGraphics Unit
 4. {
 5. {       (c) Copyright 1982 Corvus Systems, Inc.
 6. {                          San Jose, California
 7. {
 8. {       All Rights Reserved
 9. {
10. {       v 1.0  09-17-82  LEF  Original unit
11. {
12. {-----------------------------------------------------------------------}
13. {$R-}
14.
15. UNIT TurtleGraphics;
16.
17. INTERFACE
18.
19. CONST
20.       TurtleVersion = '1.0';
21.
22. TYPE
23.       ScreenColor = ( none, white, black, reverse, radar, black1, green,
24.                       violet, white1, black2, orange, blue, white2 );
25.
26. PROCEDURE InitTurtle;
27. PROCEDURE GrafMode;
28. PROCEDURE TextMode;
29. PROCEDURE ViewPort   (left,right,bottom,top: integer);
30. PROCEDURE PenColor   (c: ScreenColor);
31. PROCEDURE FillScreen (c: ScreenColor);
32. PROCEDURE Turn       (degrees: integer);
33. PROCEDURE TurnTo     (degrees: integer);
34. PROCEDURE Move       (dist: integer);
35. PROCEDURE MoveTo     (nxtX,nxtY: integer);
36. FUNCTION  TurtleX:   integer;
37. FUNCTION  TurtleY:   integer;
38. FUNCTION  TurtleAng: integer;
39. FUNCTION  ScreenBit: boolean;
40.
41.
42. IMPLEMENTATION
43.
44. {$P}
```

```
45. CONST
46.       esc        = $1B;
47.
48.       GrfMwhite =  1; { mode values }
49.       GrfMblack =  0;
50.       GrfMflip  = -1;
51.
52.       GrfQgrRel = 1;   { qual values }
53.       GrfQgrAbs = 2;
54.       GrfQchRel = 3;
55.       GrfQchAbs = 4;
56.
57. TYPE
58.       graphbuffer = record case integer of
59.                         0: (pi: array [1..10] of integer);
60.                         1: (pb: array [1..20] of -128..127);
61.                         end;
62.
63. VAR
64.       curColor:  ScreenColor; { current pen color       }
65.       curX,curY: integer;     { current turtle x, y     }
66.       curAng:    integer;     { current turtle angle    }
67.       vpX1,vpY1: integer;     { viewport left, bottom   }
68.       vpX2,vpY2: integer;     { viewport right, top     }
69.       display:   integer;     { display unit number     }
70.       buf:       graphbuffer; { display command buffer }
71.
72.
73. { SetOrigin ------------------------------------------------------------}
74. { Set graphics origin
75. {--------------------------------------------------------------------------}
76.
77. PROCEDURE SetOrigin (x,y: integer);
78.       begin
79.       with buf do begin
80.           pb[1] := esc; pb[2] := ord('o');
81.           pi[2] := x;   pi[3] := y;
82.           pb[7] := 2;
83.           unitwrite (display,buf,7);          { set graphics origin   }
84.           end;
85.       end;
86.
87. {$P}
```

```
88.  { DrawLine ------------------------------------------------------------}
89.  {---------------------------------------------------------------------}
90.
91.  PROCEDURE DrawLine (x1,y1,x2,y2: integer);
92.        var mode: integer; exchange: boolean;
93.
94.        procedure clip (r,s: real; var nx,ny: integer);
95.            var rs: real;
96.            begin rs := r+s;
97.            nx := round ((r*x2 + s*x1) / rs);
98.            ny := round ((r*y2 + s*y1) / rs);
99.            end;
100.
101.       procedure flip;
102.           var t: integer;
103.           begin
104.           t := x1; x1 := x2; x2 := t;
105.           t := y1; y1 := y2; y2 := t;
106.           exchange := not exchange;
107.           end;
108.
109.       begin
110.       if curColor = none then exit (DrawLine);
111.       exchange := FALSE;
112.       if x2 < x1 then flip;
113.       if x2 < vpX1 then exit (DrawLine)
114.           else if x1 < vpX1 then clip (vpX1-x1,x2-vpX1,x1,y1);
115.       if x1 > vpX2 then exit (DrawLine)
116.           else if x2 > vpX2 then clip (vpX2-x1,x2-vpX2,x2,y2);
117.       if y2 < y1 then flip;
118.       if y2 < vpY1 then exit (DrawLine)
119.           else if y1 < vpY1 then clip (vpY1-y1,y2-vpY1,x1,y1);
120.       if y1 > vpY2 then exit (DrawLine)
121.           else if y2 > vpY2 then clip (vpY2-y1,y2-vpY2,x2,y2);
122.       if exchange then flip;
123.       case curColor of
124.           white, white1, white2      : mode := GrfMwhite;
125.           green, violet, orange, blue: mode := GrfMflip;
126.           black, black1, black2      : mode := GrfMblack;
127.           end; {case curColor of}
128.       with buf do begin
129.           pb[1] := esc;  pb[2] := ord('1');
130.           pi[2] := x1;   pi[3] := y1;
131.           pi[4] := x2;   pi[5] := y2;
132.           pb[11] := mode;
133.           unitwrite (display,buf,11);
134.           end;
135.       end;
136.
137. {$P}
```

```
138  { GrafMode ----------------------------------------------------------------}
139  { Switch to graphics mode
140  {-------------------------------------------------------------------------}
141
142  PROCEDURE GrafMode; begin end;
143
144
145  { TextMode ----------------------------------------------------------------}
146  { Switch to text mode
147  {-------------------------------------------------------------------------}
148
149  PROCEDURE TextMode; begin end;
150
151
152  { ViewPort ----------------------------------------------------------------}
153  {-------------------------------------------------------------------------}
154
155  PROCEDURE ViewPort ((left,right,bottom,top: integer));
156        begin
157        if (left < right) and (bottom < top) then begin
158             vpX1 := left;  vpY1 := bottom;
159             vpX2 := right; vpY2 := top;
160             end;
161        end;
162
163
164  { PenColor ----------------------------------------------------------------}
165  { Set pen color
166  {-------------------------------------------------------------------------}
167
168  PROCEDURE PenColor ((c: ScreenColor));
169        begin
170        case c of
171              reverse: case curColor of
172                                white: curColor := black;
173                                black: curColor := white;
174                               black1: curColor := white1;
175                                green: curColor := violet;
176                               violet: curColor := green;
177                               white1: curColor := black1;
178                               black2: curColor := white2;
179                               orange: curColor := blue;
180                                 blue: curColor := orange;
181                               white2: curColor := black2;
182                                  end; {case curColor of}
183                 radar: ;
184             otherwise: curColor := c;
185                   end; {case c of}
186        end;
187
188  {$P}
```

```
189. { FillScreen ------------------------------------------------------------}
190. { Fill entire viewport with specified color
191. {-----------------------------------------------------------------------}
192.
193. PROCEDURE FillScreen {(c: ScreenColor)};
194.       var density: integer;
195.       begin
196.       density := 0;
197.       if c = reverse
198.           then begin
199.               case curColor of
200.                   white, white1, white2 : density := 0;
201.                   green, violet         : density := 2;
202.                   orange, blue          : density := 3;
203.                   black, black1, black2 : density := 1;
204.                   end; {case curColor of}
205.               end
206.           else begin
207.               case c of
208.                   white, white1, white2 : density := 1;
209.                   green, violet         : density := 3;
210.                   orange, blue          : density := 2;
211.                   black, black1, black2 : density := 0;
212.                   end; {case c of}
213.               end;
214.       with buf do begin
215.           pb[1] := esc;          pb[2] := ord('f');
216.           pi[2] := vpX1;         pi[3] := vpY1;
217.           pi[4] := vpY2-vpY1+1; pi[5] := vpX2-vpX1+1;
218.           pb[11] := density;
219.           unitwrite (display,buf,11);
220.           end;
221.       end;
222.
223.
224. { Turn ----------------------------------------------------------------}
225. { Turn turtle specified degrees (relative to current angle)
226. {-----------------------------------------------------------------------}
227.
228. PROCEDURE Turn {(degrees: integer)};
229.       begin
230.       curAng := (curAng + degrees) mod 360;
231.       if curAng < 0 then curAng := curAng + 360;
232.       end;
233.
234.
235. { TurnTo --------------------------------------------------------------}
236. { Turn turtle to specified angle (absolute)
237. {-----------------------------------------------------------------------}
238.
239. PROCEDURE TurnTo {(degrees: integer)};
240.       begin curAng := 0; Turn (degrees); end;
241.
242. {$P}
```

```
243.  { Move ------------------------------------------------------------------}
244.  { Move turtle for specified distance
245.  {------------------------------------------------------------------------}
246.
247.  PROCEDURE Move ((dist: integer));
248.        var nxtX,nxtY: integer; curRad: real;
249.        begin
250.        curRad := curAng * 3.1415927 / 180.0;
251.        nxtX := curX + round (dist * cos (curRad));
252.        nxtY := curY + round (dist * sin (curRad));
253.        drawline (curX,curY,nxtX,nxtY);
254.        curX := nxtX; curY := nxtY;
255.        end;
256.
257.
258.  { MoveTo ----------------------------------------------------------------}
259.  { Move turtle to specified location (absolute)
260.  {------------------------------------------------------------------------}
261.
262.  PROCEDURE MoveTo ((nxtX,nxtY: integer));
263.        begin
264.        drawline (curX,curY,nxtX,nxtY);
265.        curX := nxtX; curY := nxtY;
266.        end;
267.
268.
269.  { TurtleX ---------------------------------------------------------------}
270.  { Return current turtle X coordinate
271.  {------------------------------------------------------------------------}
272.
273.  FUNCTION TurtleX (: integer);
274.        begin TurtleX := curX; end;
275.
276.
277.  { TurtleY ---------------------------------------------------------------}
278.  { Return current turtle Y coordinate
279.  {------------------------------------------------------------------------}
280.
281.  FUNCTION TurtleY (: integer);
282.        begin TurtleY := curY; end;
283.
284.
285.  { TurtleAng -------------------------------------------------------------}
286.  { Return current turtle angle
287.  {------------------------------------------------------------------------}
288.
289.  FUNCTION TurtleAng (: integer);
290.        begin TurtleAng := curAng; end;
291.
292.  {$P}
```

```
293. { ScreenBit ------------------------------------------------------------}
294. { Return status of screen bit
295. {----------------------------------------------------------------------}
296.
297. FUNCTION ScreenBit {: boolean};
298.       const RDBYTES = 7;
299.       type  bytes = array [0..1] of -128..127;
300.       var wbuf: record bcnt: integer; bptr: ^bytes; end;
301.           b: bytes;
302.       begin
303.       ScreenBit := FALSE;
304.       wbuf.bcnt := 1; wbuf.bptr := @b;
305.       with buf do begin
306.           SetOrigin (curX,curY);                { set graphics origin   }
307.           unitstatus (display,wbuf,RDBYTES); { get byte from screen  }
308.           if b[0] < 0 then ScreenBit := TRUE;
309.           SetOrigin (0,0);                      { set graphics origin   }
310.           end;
311.       end;
312.
313.
314. { InitTurtle ---------------------------------------------------------}
315. { TurtleGraphics unit initialization
316. {----------------------------------------------------------------------}
317.
318. PROCEDURE InitTurtle;
319.       var ws: record xhome,yhome,xlen,ylen: integer; end;
320.       begin
321.       display := 1;
322.       with buf do begin
323.           pb[1] := esc; pb[2] := ord('J');
324.           unitwrite (display,buf,2);          { clear screen          }
325.           pb[1] := esc; pb[2] := ord('g');
326.           unitwrite (display,buf,2);          { set graphics mode     }
327.           SetOrigin (0,0);
328.           UnitStatus (display,ws,5);          { get window size       }
329.           pb[1] := esc; pb[2] := ord('t');
330.           unitwrite (display,buf,2);          { set text mode         }
331.           end;
332.       curAng   := 0;                  { set initial angle             }
333.       curColor := none;               { set initial pen color         }
334.       vpX1 := 0; vpX2 := ws.xlen;     { set view port left, right     }
335.       vpY1 := 0; vpY2 := ws.ylen;     { set view port bottom, top     }
336.       curX := vpX2 div 2;             { set initial X                 }
337.       curY := vpY2 div 2;             { set initial Y                 }
338.       end;
339.
340. end.
```

```
GRFQCHREL      54
GRFQGRABS      53
GRFQGRREL      52
INITTURTLE     26   318
LEFT           29   157   158
MODE           92   124   125   126   132
MOVE           34   247
MOVETO         35   262
NONE           23   110   333
NX             94    97
NXTX           35   248   251   253   254   264   265
NXTY           35   248   252   253   254   264   265
NY             94    98
ORANGE         24   125   179   180   202   210
OTHERWISE     184
PB             60    80    82   129   132   215   218   323   325   329
PENCOLOR       30   168
PI             59    81   130   131   216   217
R              94    96    97    98
RADAR          23   183
RDBYTES       298   307
REVERSE        23   171   197
RIGHT          29   157   159
RS             95    96    97    98
S              94    96    97    98
SCREENBIT      39   297   303   308
SCREENCOLO     23    30    31    64
SETORIGIN      77   306   309   327
T             102   104   105
TEXTMODE       28   149
TOP            29   157   159
TURN           32   228   240
TURNTO         33   239
TURTLEANG      38   289   290
TURTLEGRAP     15
TURTLEVERS     20
TURTLEX        36   273   274
TURTLEY        37   281   282
VIEWPORT       29   155
VIOLET         24   125   175   176   201   209
VPX1           67   113   114   158   216   217   334
VPX2           68   115   116   159   217   334   336
VPY1           67   118   119   158   216   217   335
VPY2           68   120   121   159   217   335   337
WBUF          300   304   307
WHITE          23   124   172   173   200   208
WHITE1         24   124   174   177   200   208
WHITE2         24   124   178   181   200   208
WS            319   328   334   335
X              77    81
X1             91    97   104   112   114   115   116   119   130
X2             91    97   104   112   113   114   116   121   131
XHOME         319
XLEN          319   334
```

Y	77	81							
Y1	91	98	105	114	117	119	120	121	130
Y2	91	98	105	116	117	118	119	121	131
YHOME	319								
YLEN	319	335							

```
    1. { FCLKIO.TEXT ----------------------------------------------------------
    2. {
    3. {       FCLKIO -- Corvus CONCEPT FORTRAN Clock Processing Unit
    4. {
    5. {       (c) Copyright 1982 Corvus Systems, Inc.
    6. {                          San Jose, California
    7. {
    8. {       All Rights Reserved
    9. {
   10. {       v 1.0  10-22-82  LEF  Original unit
   11. {
   12. {---------------------------------------------------------------------
   13. {$R-}
   14.
   15. UNIT FclkIO;
   16.
   17. INTERFACE
   18.
   19. USES {$U CCLIB} CCclkIO;
   20.
   21. PROCEDURE ClkInt;
   22. PROCEDURE ClkRd  (var CPB: ClkPB);
   23. PROCEDURE ClkWr  (var CPB: ClkPB);
   24. PROCEDURE ClkDay (var DateStr: ClkStr40; ln: integer);
   25. PROCEDURE ClkDt1 (var DateStr: ClkStr40; ln: integer);
   26. PROCEDURE ClkDt2 (var DateStr: ClkStr40; ln: integer);
   27. PROCEDURE ClkDt3 (var DateStr: ClkStr40; ln: integer);
   28. PROCEDURE ClkTm1 (var DateStr: ClkStr40; ln: integer);
   29. PROCEDURE ClkTm2 (var DateStr: ClkStr40; ln: integer);
   30.
   31. IMPLEMENTATION
   32.
   33. PROCEDURE ClkInt; begin CCclkIOinit; end;
   34. PROCEDURE ClkRd;  begin ClkRead (CPB); end;
   35. PROCEDURE ClkWr;  begin ClkWrite (CPB); end;
   36. PROCEDURE ClkDay; begin ClkWeekDay (DateStr); end;
   37. PROCEDURE ClkDt1; begin ClkDate1 (DateStr); end;
   38. PROCEDURE ClkDt2; begin ClkDate2 (DateStr); end;
   39. PROCEDURE ClkDt3; begin ClkDate3 (DateStr); end;
   40. PROCEDURE ClkTm1; begin ClkTime1 (DateStr); end;
   41. PROCEDURE ClkTm2; begin ClkTime2 (DateStr); end;
   42.
   43. END.
   44.
```

```
CCCLKIO          19
CCCLKIOINI       33
CLKDATE1         37
CLKDATE2         38
CLKDATE3         39
CLKDAY           24     36
CLKDT1           25     37
CLKDT2           26     38
CLKDT3           27     39
CLKINT           21     33
CLKPB            22     23
CLKRD            22     34
CLKREAD          34
CLKSTR40         24     25     26     27     28     29
CLKTIME1         40
CLKTIME2         41
CLKTM1           28     40
CLKTM2           29     41
CLKWEEKDAY       36
CLKWR            23     35
CLKWRITE         35
CPB              22     23     34     35
DATESTR          24     25     26     27     28     29     36     37     38     39     40
                 41
FCLKIO           15
```

```
{ FCRTIO.TEXT ----------------------------------------------------------!
{
{       FCRTIO -- Corvus CONCEPT FORTRAN CRT Control Unit
{
{       (c) Copyright 1982 Corvus Systems, Inc.
{                          San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  10-20-82  LEF  Original unit
{
{----------------------------------------------------------------------!
{$R-}

UNIT FrtIO;

INTERFACE

USES {$U CCLIB} CCdefn, CCcrtIO;

TYPE CrtArr80 = packed array [1..80] of char;

PROCEDURE CrtInt;
PROCEDURE GoXY    (var x,y: LongInt);
PROCEDURE CrtAct  (var cmd: LongInt);
PROCEDURE CrtTtl  (var txt: CrtArr80; ln:  integer);
PROCEDURE CrtPmt  (var txt: CrtArr80; ln1: integer;
                   var opt: CrtArr80; ln2: integer);
PROCEDURE CrtPau  (var ch: char);
FUNCTION  Ucase   (var ch: char):                              char;
FUNCTION  GetI    (var num: integer):                          CrtStatus;
FUNCTION  GetLI   (var num: LongInt):                          CrtStatus;
FUNCTION  GetSt   (var buf: CrtArr80; ln: integer):            CrtStatus;
FUNCTION  GetB:                                                char;
FUNCTION  Tone    (var timbre,duration,period: LongInt): LongInt;


IMPLEMENTATION

PROCEDURE MakeString (a: CrtArr80; ln: integer; var s: string80);
    var i: integer;
    begin
    s := '';
    for i := 1 to ln do begin
        s := concat (s,' '); s[length(s)] := a[i]; end;
    end;

{$P}
```

```
49.  PROCEDURE CrtInt;
50.      begin CCcrtIOinit; end;
51.
52.  PROCEDURE GoXY;
53.      begin GoToXY (ord(x),ord(y)); end;
54.
55.  PROCEDURE CrtAct;
56.      var CC: record case integer of
57.                  1: (li:  LongInt);
58.                  2: (fi:  array [0..3] of -128..127;
59.                      cmd: CrtCommand);
60.                  end;
61.      begin CC.li := cmd; CrtAction (CC.cmd); end;
62.
63.  PROCEDURE CrtTtl;
64.      var s: string80;
65.      begin MakeString (txt,ln,s); CrtTitle (s); end;
66.
67.  PROCEDURE CrtPmt;
68.      var s1,s2: string80;
69.      begin MakeString (txt,ln1,s1); MakeString (opt,ln2,s2);
70.      CrtPrompt (s1,s2); end;
71.
72.  PROCEDURE CrtPau;
73.      begin CrtPause (ch); end;
74.
75.  FUNCTION Ucase;
76.      begin Ucase := UpperCase (ch); end;
77.
78.  FUNCTION GetLI;
79.      begin GetLI := GetLongNum (num); end;
80.
81.  FUNCTION GetI;
82.      begin GetI := GetNum (num); end;
83.
84.  FUNCTION GetB;
85.      begin GetB := GetByte; end;
86.
87.  FUNCTION GetSt;
88.      var s: string80; status: CrtStatus; i: integer;
89.      begin MakeString (buf,ln,s);
90.      status := GetString (s);
91.      if s = '' then s := ' ';
92.      for i := 0 to length(s)+1 do buf[i+1] := s[i];
93.      GetSt := status;
94.      end;
95.
96.  FUNCTION Tone;
97.      begin
98.      Tone := ord4(BellTone (ord(timbre),ord(duration),ord(period)));
99.      end;
100.
101. END.
102.
```

STRING80	40	64	68	88
TIMBRE	35	98		
TONE	35	96	98	
TXT	26	27	65	69
UCASE	30	75	76	
UPPERCASE	76			
X	24	53		
Y	24	53		

```
{ FGRFIO.TEXT ---------------------------------------------------------}
{
{       FGRFIO -- Corvus CONCEPT FORTRAN Graphics Support Unit
{
{       (c) Copyright 1982 Corvus Systems, Inc.
{                          San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  10-23-82  LEF  Original unit
{
{-------------------------------------------------------------------------}
{$R-}

UNIT FgrfIO;

INTERFACE

USES {$U CCLIB} CCdefn, CCgrfIO;

PROCEDURE GrInit;
PROCEDURE GrSetO (var x1,y1,qual:          LongInt);
PROCEDURE GrPlot (var x1,y1,mode:          LongInt);
PROCEDURE GrDraw (var x1,y1,x2,y2,mode:    LongInt);
PROCEDURE GrFill (var x1,y1,wd,ht,den:     LongInt);
PROCEDURE GrCopy (var x1,y1,wd,ht,x2,y2:   LongInt);

IMPLEMENTATION

PROCEDURE GrInit; begin CCgrfIOinit; end;
PROCEDURE GrSetO; begin SetOrigin (ord (x1), ord (y1),
                                   ord (qual));          end;
PROCEDURE GrPlot; begin PlotPoint (ord (x1), ord (y1),
                                   ord (mode));          end;
PROCEDURE GrDraw; begin DrawLine  (ord (x1), ord (y1),
                                   ord (x2), ord (y2),
                                   ord (mode));          end;
PROCEDURE GrFill; begin FillBox   (ord (x1), ord (y1),
                                   ord (wd), ord (ht),
                                   ord (den));           end;
PROCEDURE GrCopy; begin CopyBox   (ord (x1), ord (y1),
                                   ord (wd), ord (ht),
                                   ord (x2), ord (y2)); end;

END
```

CCDEFN	19									
CCGRFIO	19									
CCGRFIOINI	30									
COPYBOX	41									
DEN	25	40								
DRAWLINE	35									
FGRFIO	15									
FILLBOX	38									
GRCOPY	26	41								
GRDRAW	24	35								
GRFILL	25	38								
GRINIT	21	30								
GRPLOT	23	33								
GRSETO	22	31								
HT	25	26	39	42						
LONGINT	22	23	24	25	26					
MODE	23	24	34	37						
PLOTPOINT	33									
QUAL	22	32								
SETORIGIN	31									
WD	25	26	39	42						
X1	22	23	24	25	26	31	33	35	38	41
X2	24	26	36	43						
Y1	22	23	24	25	26	31	33	35	38	41
Y2	24	26	36	43						

```
 1. { FLBLIO.TEXT ----------------------------------------------------------}
 2. {
 3. {       FLBLIO -- Corvus CONCEPT FORTRAN Label Processing Unit
 4. {
 5. {       (c) Copyright 1982 by Corvus Systems, Inc.
 6. {                             San Jose, California
 7. {
 8. {       All Rights Reserved
 9. {
10. {       v 1.0  11-01-82  LEF  Original unit
11. {
12. {---------------------------------------------------------------------}
13. {$R-}
14.
15. UNIT FlblIO;
16.
17. INTERFACE
18.
19. USES {$U CCLIB} CCdefn, CClblIO;
20.
21. TYPE LblArr80 = packed array [1..80] of char;
22.
23. PROCEDURE LbInit;
24. PROCEDURE LbInt;
25. PROCEDURE LbOn;
26. PROCEDURE LbOff;
27. FUNCTION  LbSet (var KN: LongInt;
28.                  var LblStr: LblArr80; ln1: integer;
29.                  var RetStr: LblArr80; ln2: integer): LongInt;
30.
31. IMPLEMENTATION
32.
33. PROCEDURE LbInit; begin CClblIOinit; end;
34. PROCEDURE LbInt;  begin LblsInit;    end;
35. PROCEDURE LbOn;   begin LblsOn;      end;
36. PROCEDURE LbOff;  begin LblsOff;     end;
37. FUNCTION  LbSet;
38.     var i: integer; ls: LblKeyStr; rs: LblRtnStr;
39.     begin
40.     ls := ''; rs := '';
41.     for i := 1 to ln1 do begin
42.         ls := concat (ls,' '); ls[length(ls)] := LblStr[i]; end;
43.     for i := 1 to ln2 do begin
44.         rs := concat (rs,' '); rs[length(rs)] := RetStr[i]; end;
45.     LbSet := ord4(LblSet (ord(KN),ls,rs));
46.     end;
47.
48. END.
```

1	21	41	43		
80	21				
CCDEFN	19				
CCLBLIO	19				
CCLBLIOINI	33				
FLBLIO	15				
I	38	41	42	43	44
KN	27	45			
LBINIT	23	33			
LBINT	24	34			
LBLARR80	21	28	29		
LBLKEYSTR	38				
LBLRTNSTR	38				
LBLSET	45				
LBLSINIT	34				
LBLSOFF	36				
LBLSON	35				
LBLSTR	28	42			
LBOFF	26	36			
LBON	25	35			
LBSET	27	37	45		
LN1	28	41			
LN2	29	43			
LONGINT	27	29			
LS	38	40	42	45	
ORD4	45				
RETSTR	29	44			
RS	38	40	44	45	

```
    1  { FOMNIO.TEXT ------------------------------------------------------------}
    2  {
    3  {       FOMNIO -- Corvus CONCEPT FORTRAN OMNINET Commands Unit
    4  {
    5  {       (c) Copyright 1982 Corvus Systems, Inc.
    6  {                          San Jose, California
    7  {
    8  {       All Rights Reserved
    9  {
   10  {       v 1.0  10-26-82  LEF  Original unit
   11  {
   12  {-------------------------------------------------------------------------}
   13  {$R-}
   14
   15  UNIT FomnIO;
   16
   17  INTERFACE
   18
   19  USES
   20  {$U CCI.IB} CCdefn, CComnIO;
   21
   22  PROCEDURE OmInit;
   23  PROCEDURE OmSndM (var rslt,BP,RP,sn,dln,hln,dst: LongInt);
   24  PROCEDURE OmSetR (var rslt,BP,RP,sn,dln,hln:     LongInt);
   25  PROCEDURE OmEndR (var rslt,sn:                   LongInt);
   26  PROCEDURE OmITrn (var rslt:                      LongInt);
   27  PROCEDURE OmEcho (var rslt,dest:                 LongInt);
   28  PROCEDURE OmWho  (var rslt:                      LongInt);
   29
   30  IMPLEMENTATION
   31
   32  PROCEDURE OmInit; begin CComnIOinit; end;
   33  PROCEDURE OmSndM; begin
   34                    OCsndMesg (@BP,@RP,
   35                               ord(sn),ord(dln),ord(hln),ord(dst));
   36                    rslt := ord4(OCresult); end;
   37  PROCEDURE OmSetR; begin
   38                    OCsetRecv (@BP,@RP,
   39                               ord(sn),ord(dln),ord(hln));
   40                    rslt := ord4(OCresult); end;
   41  PROCEDURE OmEndR; begin OCendRecv (ord(sn));
   42                    rslt := ord4(OCresult); end;
   43  PROCEDURE OmITrn; begin OCinitTrans; rslt := ord4(OCresult); end;
   44  PROCEDURE OmEcho; begin OCechoTrans (ord(dest));
   45                    rslt := ord4(OCresult); end;
   46  PROCEDURE OmWho;  begin OCwhoAmI; rslt := ord4(OCresult); end;
   47
   48  END.
   49
```

```
 1. { FTURTLE.TEXT ------------------------------------------------------!
 2. {
 3. {       FTURTLE -- Corvus CONCEPT FORTRAN TurtleGraphics Unit
 4. {
 5. {       (c) Copyright 1982 Corvus Systems, Inc.
 6. {                          San Jose, California
 7. {
 8. {       All Rights Reserved
 9. {
10. {       v 1.0  10-23-82  LEF  Original unit
11. {
12. {------------------------------------------------------------------------!
13. {$R-}
14.
15. UNIT Fturtle;
16.
17. INTERFACE
18.
19. USES {$U CCLIB} TurtleGraphics;
20.
21. PROCEDURE InitTu;
22. PROCEDURE GrafMo;
23. PROCEDURE TextMo;
24. PROCEDURE ViewPo (var left,right,bottom,top: LongInt);
25. PROCEDURE PenCol (var c:                     LongInt);
26. PROCEDURE FilScr (var c:                     LongInt);
27. PROCEDURE TTrn   (var degrees:               LongInt);
28. PROCEDURE TTrnTo (var degrees:               LongInt);
29. PROCEDURE TMov   (var dist:                  LongInt);
30. PROCEDURE TMovTo (var nxtX,nxtY:             LongInt);
31. FUNCTION  TurtlX: LongInt;
32. FUNCTION  TurtlY: LongInt;
33. FUNCTION  TurtlA: LongInt;
34. FUNCTION  ScrBit: boolean;
35.
36. {$P}
```

```
37. IMPLEMENTATION
38. VAR SC:     record case integer of
39.                 1: (li: LongInt);
40.                 2: (fl: array [1..3] of -128..127;
41.                     cl: ScreenColor);
42.             end;
43. PROCEDURE InitTu; begin InitTurtle; end;
44. PROCEDURE GrafMo; begin GrafMode; end;
45. PROCEDURE TextMo; begin TextMode; end;
46. PROCEDURE ViewPo; begin ViewPort (ord(left), ord(right),
47.                                   ord(bottom), ord(top)); end;
48. PROCEDURE PenCol; begin SC.li := c; PenColor (SC.cl); end;
49. PROCEDURE FilScr; begin SC.li := c; FillScreen (SC.cl); end;
50. PROCEDURE TTrn;   begin Turn (ord(degrees)); end;
51. PROCEDURE TTrnTo; begin TurnTo (ord(degrees)); end;
52. PROCEDURE TMov;   begin Move (ord(dist)); end;
53. PROCEDURE TMovTo; begin MoveTo (ord(nxtX),ord(nxtY)); end;
54. FUNCTION  TurtlX; begin TurtlX := ord4(TurtleX); end;
55. FUNCTION  TurtlY; begin TurtlY := ord4(TurtleY); end;
56. FUNCTION  TurtlA; begin TurtlA := ord4(TurtleAng); end;
57. FUNCTION  ScrBit; begin ScrBit := ScreenBit; end;
58.
59. END.
```

```
{ FWNDIO.TEXT ------------------------------------------------------------}
{
{       FWNDIO -- Corvus CONCEPT FORTRAN Window Processing Unit
{
{       (c) Copyright 1982 by Corvus Systems, Inc.
{                             San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  10-23-82  LEF  Original unit
{
{------------------------------------------------------------------------}
{$R-}

UNIT FwndIO;

INTERFACE

USES {$U CCLIB} CCdefn, CCwndIO;

TYPE WndArr80 = packed array [1..80] of char;

PROCEDURE WnInit;
FUNCTION  WnSys  (var wn: LongInt):                                LongInt;
FUNCTION  WnCre  (var WR: WndRcd; var homex,homey,
                      width,lngth,flags: LongInt):                 LongInt;
FUNCTION  WnSel  (var WR: WndRcd):                                 LongInt;
FUNCTION  WnDel  (var WR: WndRcd):                                 LongInt;
FUNCTION  WnClr  (var WR: WndRcd):                                 LongInt;
FUNCTION  WnStat (var homex,homey,width,lngth,
                      curx,cury: integer):                         LongInt;
FUNCTION  WnLoad (var name: WndArr80; ln: integer):                LongInt;

IMPLEMENTATION

PROCEDURE WnInit; begin CCwndIOinit; end;
FUNCTION WnSys;  begin WnSys  := ord4(WinSystem (ord(wn))); end;
FUNCTION WnCre;  begin WnCre  := ord4(WinCreate (WR,ord(homex),ord(home!
                                                    ord(width),ord(lngt!
                                                    ord(flags))); end;
FUNCTION WnSel;  begin WnSel  := ord4(WinSelect (WR)); end;
FUNCTION WnDel;  begin WnDel  := ord4(WinDelete (WR)); end;
FUNCTION WnClr;  begin WnClr  := ord4(WinClear  (WR)); end;
FUNCTION WnStat; begin WnStat := ord4(WinStatus (homex,homey,width,lngt!
                                                 curx,cury)); end;
FUNCTION WnLoad; var i: integer; s: string80;
                 begin s := '';
                 for i := 1 to ln do begin
                     s := concat (s,' '); s[length(s)] := name[i]; end;
                 WnLoad := ord4(WinLoadCh (s));
                 end;
END.
```

```
    1. { DRVIO.TEXT ------------------------------------------------------------}
    2. {
    3. {       DRVIO -- Corvus Disk Drive I/O unit
    4. {
    5. {       (c) Copyright 1982 Corvus Systems, Inc.
    6. {                          San Jose, California
    7. {
    8. {       All Rights Reserved
    9. {
   10. {       v 1.0  05-28-82  DP   Original unit
   11. {         1.0e 23-Sep-82 DP   Fixed firmware message
   12. {       v 2.0  09-16-82  cr/jk revh mods
   13. {
   14. { Purpose: This unit is used by all of the Corvus utilities which talk
   15. {          directly to the Corvus drive (i.e., not through the operating
   16. {          system driver).  It can be used for both OMNINET and local
   17. {          disks.  It can access any slot and any server.
   18. {
   19. {----------------------------------------------------------------------}
   20.
   21. {!CC} UNIT CCdrvIO;
   22.
   23. INTERFACE
   24.
   25. USES
   26. {!CC} {$U /CCUTIL/CCLIB} CCdefn, CCLngInt;
   27.
   28. CONST
   29.       DrvIOversion = '2.0 ';     { Unit revision level                 !
   30.       CDbuf_max    = 1023;       { max. no of bytes on send to OMNINET +!
   31.       DrvBlkSize   = 512;
   32.       SndRcvMax    = 530;
   33. {!CC} low_slot     = 1;
   34. {!CC} high_slot    = 5;
   35.       low_server   = 0;
   36.       high_server  = 63;
   37.       MUX          = 64;      { max server + 1 }
   38.       DrMax        = 7;       { Max nmbr of drives on disk server or Mux}
   39.
   40.
   41. TYPE
   42.     SndRcvStr    = RECORD
   43.                      sln: INTEGER; {send length}
   44.                      rln: INTEGER; {recv length}
   45.                      CASE integer OF
   46.                          1: (c:    PACKED ARRAY [1..SndRcvMax] OF CHAR);
   47. {!CC}                  2: (b:           ARRAY [1..SndRcvMax] OF byte);
   48.                      END;
   49.
   50.     DrvBlk       = RECORD CASE INTEGER OF
   51.                          1: (c:    PACKED ARRAY [1..DrvBlkSize] OF CHAR);
   52. {!CC}                  2: (b:           ARRAY [1..DrvBlkSize] OF byte);
   53.                      END;
   54.
```

```
55. {!CC}     cd_buf       = ARRAY [0..cdbuf_max] OF byte;
56.
57.       host_types  = (user_station,
58.                      file_server,
59.                      printer_server,
60.                      name_server,
61.                      modem_server,
62.                      db_server,
63.                      ON_interconnect,
64.                      X25_gateway,
65.                      SNA_gateway);
66.
67.       valid_slot  = low_slot..high_slot;
68.
69.       valid_server= low_server..high_server;
70.
71.       CDaddr      = RECORD
72.                       Slotno:     Byte;       { Slot number              !
73.                       Kind:       SlotType;   { Type of interface in slo!
74.                       Netno:      Byte;       { Network number (UNUSED)  !
75.                       Stationno:  Byte;       { OMNINET station address  !
76.                       Driveno:    Byte;       { Disk drive number        !
77.                       Blkno:      longint;    { Disk block number        !
78.                       END;
79.
80.       DrRev       = (NoDrv,RevA,RevB,RevH);
81.       DrSizes     = (OldTenMB,FiveMB,TenMB,TwentyMB,FortyMB,SixtyMB,Hundre!
82.       PhysDrInfo  = RECORD
83.                       spt:        INTEGER;    { Sectors/track            !
84.                       tpc:        INTEGER;    { Tracks/Sector            !
85.                       cpd:        INTEGER;    { Cylinders/Drive          !
86.                       Capacity:   LONGINT;    { Total nmbr of 512 byte blocks!
87.                       DrSize:     DrSizes;    { Drive size               !
88.                       DrType:     DrRev;      { Drive controller revision!
89.                       PhysDr:     BOOLEAN;    { true if a physical drive !
90.                       ROMvers:    INTEGER;    { ROM version              !
91.                       FirmMsg:    STRING[8];  { Firmware message (i.e. CF17.3!
92.                       FirmVers:   INTEGER;    { Firmware version number  !
93.                       END;
94.       PDrArray    = ARRAY [1..DrMax] OF PhysDrInfo;
95.
96.       Sprtrks     = ARRAY [1..DrMax] OF INTEGER;
97.
98.
99.  VAR
100.      spares   :  Sprtrks;
101.
102. FUNCTION  CDSlot     (     Slotnum: integer): BOOLEAN;
103. FUNCTION  CDSlotInfo (     Slotnum: integer): SlotType;
104. FUNCTION  CDBootInfo (VAR Slotnum: integer;
105.                       VAR Srvrnum: integer): SlotType;
106. FUNCTION  CDServer   (     Server: integer ): BOOLEAN;
107. PROCEDURE Initslot   (VAR NetLoc: CDaddr  );
108. PROCEDURE CDsend     (     NetLoc: CDaddr; VAR st: SndRcvStr);
```

```
109.  PROCEDURE CDrecv      (     NetLoc: CDaddr; VAR st: SndRcvStr);
110.  FUNCTION  CDread      (     NetLoc:  CDaddr; { network address of drive  '
111.                         VAR buf:     CD_buf; { data that is read          '
112.                              len:     integer { number of bytes to read    '
113.                         )    integer;         { returns disk error code    '
114.  FUNCTION  CDwrite     (     NetLoc:  CDaddr; { network address of drive  '
115.                         VAR buf:     CD_buf; { data to be written         '
116.                              len:     integer { number of bytes to write   '
117.                         ):   integer;         { returns disk error code
118.  PROCEDURE DrvInit   (NetLoc:          CDaddr;
119.                       VAR NumDrives:   INTEGER;
120.                       VAR PhysDrives: PDrArray);
121.  PROCEDURE CCdrvIOinit;
122.
123.
124.  IMPLEMENTATION
125.
126.  {$P}
```

```
127
128.  CONST
129.       Broadcast_add    = 255;
130.
131.       Misc_Error       = 255;     { Miscellaneous IO error       }
132.       Misc_Omni_Error  = 254;     { Miscellaneous OMNINET error  }
133.       Inv_srvr         = 253;     { Invalid server number        }
134.       Inv_Slot         = 252;     { Invalid slot number          }
135.
136.       TenMBSize = 18936;  { Nmbr of blocks on a ten megabyte drive }
137.
138.  VAR
139.  {$CC}  Active_slot:   Valid_slot; { Current IO slot in use              }
140.  {$CC}                            { must be global with this name for Apple!
141.         Cur_Kind:      SlotType;    { Current interface media type       }
142.  {$CC}  Disk_server:   integer;     { Current OMNINET disk server address }
143.  {$CC}                            { must be global with this name for Apple!
144.
145.  {$CC}  FUNCTION OSactSlt: integer;                          EXTERNAL;
146.  {$CC}  FUNCTION OSactSrv: integer;                          EXTERNAL;
147.  {$CC}  FUNCTION OSsltType (slotnum: integer): SlotType;     EXTERNAL;
148.  {$CC}  FUNCTION OSSltDv : integer;                          EXTERNAL;
149.
150.  {$P}
```

```
151  {-------------------------------------------------------------------------!
152  { Procedure:    CDBOOTINFO                                                !
153  {                                                                         !
154  { Description. This procedure returns the boot slot number and type       !
155  {                                                                         !
156  {-------------------------------------------------------------------------!
157
158  FUNCTION CDBootInfo ((VAR Slotnum: integer;
159                        VAR Srvrnum: integer): SlotType);
160      BEGIN
161      Slotnum := USActSlt;
162      Srvrnum := USActSrv;
163      IF (Slotnum < low_slot) OR (Slotnum > high_slot)
164  {!CC}   THEN CDBootInfo := NoDisk ELSE CDBootInfo := OSSltType (slotnum);
165      END;
166
167
168  {-------------------------------------------------------------------------!
169  { Procedure:    CDSLOTINFO                                                !
170  {                                                                         !
171  { Description: This procedure when given a slot number determine the ty!
172  {               of interface it any the slot is allocated to....          !
173  {                                                                         !
174  {-------------------------------------------------------------------------!
175
176  FUNCTION CDSlotInfo ((Slotnum: Valid_Slot): SlotType);
177      BEGIN
178  {!CC} CDSlotInfo := OSSltType (Slotnum);
179      END;
180
181  {$P}
```

```
182. {-------------------------------------------------------------------------}
183. { Procedure:    CDSLOT                                                     }
184. {                                                                          }
185. { Description:                                                             }
186. {                                                                          }
187. {-------------------------------------------------------------------------}
188.
189. FUNCTION CDSlot {(slotnum: valid_slot): BOOLEAN };
190.     BEGIN
191. {!CC} IF OSslttype(slotnum) IN [LocalDisk, OmninetDisk]
192.         THEN BEGIN
193.             Active_slot := slotnum;
194.             CDSlot := TRUE;
195.             END
196.         ELSE CDSlot := FALSE;
197.     END;
198.
199.
200. {-------------------------------------------------------------------------}
201. { Procedure:    CDSERVER                                                  }
202. {                                                                         }
203. { Description:                                                            }
204. {                                                                         }
205. {-------------------------------------------------------------------------}
206.
207. FUNCTION CDServer { Server: valid_server ): BOOLEAN };
208.     BEGIN
209.     {                                                  }
210.     { validate that servernum is a disk server }
211.     {                                                  }
212.     Disk_server := Server;
213.     END;
214.
215.
216. {-------------------------------------------------------------------------}
217. {-------------------------------------------------------------------------}
218.
219. PROCEDURE Initslot {(VAR Netloc: CDaddr)};
220.     VAR x,y: INTEGER;
221.     BEGIN
222.     WITH Netloc DO BEGIN
223.         Kind      := CDbootInfo (x,y);
224.         Slotno    := x;
225.         Driveno   := 1;
226.         Netno     := 0;
227.         Stationno := y;
228.         Blkno     := 0;
229.         END;
230.     END;
231.
232. {$P}
```

```
233.  {-------------------------------------------------------------------------
234.  { Procedure:    CDSEND                                                    
235.  {                                                                        
236.  { Description: This procedure send a disk command to the specified driv
237.  {                                                                        
238.  {-------------------------------------------------------------------------
239.  
240.  PROCEDURE CDSend ((NetLoc: CDaddr; VAR st: SndRcvStr));
241.      VAR Drive_unit:    INTEGER;     { unit for sending/receiving commands
242.  
243.      BEGIN
244.      IF (Netloc.Slotno >= Low_slot) OR (Netloc.Slotno <= high_slot)
245.         THEN BEGIN
246.            Active_slot := NetLoc.Slotno;
247.  {!CC}     Drive_unit := OSSltDv;
248.  
249.            Cur_Kind := NetLoc.Kind;
250.            IF Cur_Kind = LocalDisk
251.  {!CC}        THEN UNITWRITE (Drive_unit,st.c,st.sln,0,Active_slot)
252.                 ELSE
253.            IF Cur_kind = Omninetdisk
254.               THEN BEGIN
255.                  IF (NetLoc.Stationno >= Low_server) OR (NetLoc.Stationno
256.                     THEN BEGIN
257.                        Disk_server := NetLoc.Stationno;
258.  {!CC}                 UNITWRITE (Drive_unit,st.c,st.sln,0,Disk_server*25
259.                        END
260.                  END;
261.            END;
262.      END;
263.  
264.  {$P}
```

```
265  {-------------------------------------------------------------------------!
266  { Procedure:    CDRECV                                                    !
267  {                                                                         !
268  { Description:  This procedure receives the response from the drive afte!
269  {               sending a drive command.                                  !
270  {                                                                         !
271  {-------------------------------------------------------------------------!
272
273  PROCEDURE CDRecv ((NetLoc: CDaddr; VAR st: SndRcvStr));
274      VAR Drive_unit:    INTEGER;     { unit for sending/receiving commands!
275  {!CC}   ior:           INTEGER;
276
277      BEGIN
278  {!CC}   ior := 0;
279      IF (Netloc.Slotno < Low_slot) OR (Netloc.Slotno > High_slot)
280          THEN BEGIN St.c[1] := CHR(Inv_slot); st.rln := 1; END
281          ELSE BEGIN
282              Active_slot := NetLoc.Slotno;
283  {!CC}       Drive_unit := OSSltDv;
284
285              Cur_Kind := NetLoc.Kind;
286              IF Cur_Kind = LocalDisk
287  {!CC}           THEN BEGIN UNITREAD (Drive_unit,st.c,st.rln,0,Active_slot)!
288                  ELSE
289                    IF Cur_kind = Omninetdisk
290                        THEN BEGIN
291                          IF (NetLoc.Stationno < Low_server) OR (NetLoc.Stati!
292                              THEN BEGIN St.c[1] := CHR(Inv_srvr); st.rln := 1!
293                              ELSE BEGIN
294                                  Disk_server := NetLoc.Stationno;
295  {!CC}                           UNITREAD (Drive_unit,st.c,st.rln,0,Disk_serve!
296  {!CC}                           ior := IORESULT;
297                                  END
298                          END
299                      ELSE BEGIN St.c[1] := CHR(Inv_slot); st.rln := 1; END;
300              END;
301  {!CC}   IF (ior <> 0) AND (ior <> 4)  { 4 is disk error > 127 }
302  {!CC}       THEN BEGIN st.c[1] := CHR(misc_error);  st.rln := 1; END;
303      END;
304
305  {$P}
```

```
306. {-------------------------------------------------------------------------!
307. { Procedure:    CDREAD                                                     !
308. {                                                                          !
309. { Description:                                                            !
310. {                                                                          !
311. {-------------------------------------------------------------------------!
312.
313. FUNCTION CDread {(NetLoc: CDaddr; VAR buf: CD_buf; len: integer): integ!
314.     VAR xcv: SndRcvStr; Move_len,Count,T: integer;
315.     BEGIN
316.     Count := 0;
317.     REPEAT
318.         WITH NetLoc DO BEGIN
319.             {                                  }
320.             { build read command...            }
321.             {                                  }
322.             xcv.sln := 4;   xcv.rln := 513;
323.             xcv.b[1] := 50;
324.             T := LIntByte (1,Blkno);
325.             T := T MOD 16;                { save lower four bits          }
326.             xcv.b[2] := t*16 + Driveno; { and store in upper four bits }
327.             xcv.b[3] := LIntByte (3,Blkno);
328.             xcv.b[4] := LIntByte (2,Blkno);
329.
330.             CDsend (NetLoc,xcv); CDrecv (NetLoc,xcv);
331.
332.             IF Len > 512 THEN Move_len := 512
333.                          ELSE Move_len := Len;
334.             {$R-} MOVELEFT (xcv.b[2],Buf[Count*512],Move_len); {$R+}
335.             Count := Count+1;
336.             Blkno := Blkno+1;
337.             len:= len-512;
338.             END;
339.         UNTIL (ORD(xcv.c[1]) > 127) OR (len <= 0);
340.     IF ORD(xcv.c[1]) > 127 THEN CDRead := ORD(xcv.c[1]) ELSE CDRead := !
341.     END;
342.
343. {$P}
```

```
344.  {-------------------------------------------------------------------------!
345.  { Procedure:    CDWRITE                                                    !
346.  {                                                                         !
347.  { Description:                                                           !
348.  {                                                                         !
349.  {-------------------------------------------------------------------------!
350.
351.  FUNCTION CDwrite ((NetLoc: CDaddr; VAR buf: CD_buf; len: integer): inte!
352.      VAR xcv: SndRcvStr; Move_len,Count,T: integer;
353.      BEGIN
354.      Count := 0;
355.      WITH NetLoc DO BEGIN
356.          REPEAT
357.          {                                       }
358.          { build write command...                }
359.          {                                       }
360.          xcv.sln := 516; xcv.rln := 1;
361.          xcv.b[1] := 51;
362.          T := LIntByte(1,Blkno);
363.          T := T MOD 16;                { save lower four bits          }
364.          xcv.b[2] := T*16 + Driveno; { and store in upper four bits }
365.          xcv.b[3] := LIntByte (3,Blkno);
366.          xcv.b[4] := LIntByte (2,Blkno);
367.          {$R-} MOVELEFT (Buf[Count*512],xcv.b[5],512); {$R+}
368.
369.          CDsend (NetLoc,xcv); CDrecv (NetLoc,xcv);
370.
371.          Count := Count+1;
372.          Blkno := Blkno+1;
373.          Len := Len-512;
374.          UNTIL (ORD(xcv.c[1]) > 127) OR (len <= 0);
375.          END;
376.      IF ORD(xcv.c[1]) > 127 THEN CDWrite := ORD(xcv.c[1]) ELSE CDWrite := !
377.      END;
378.
379.  {$P}
```

```
380  PROCEDURE DrvInit ((     NetLoc: CDaddr;
381                           VAR NumDrives:  INTEGER;
382                           VAR PhysDrives: PDrArray));
383       VAR x  INTEGER; xcv: SndRcvStr; MaxSpTrk: INTEGER;
384
385       PROCEDURE SetRevA;
386           VAR i: integer;
387           BEGIN
388           NumDrives := xcv.b[1] mod 8;
389           FOR i := 1 TO NumDrives DO
390              WITH PhysDrives[NumDrives] DO BEGIN
391                   Spt := 18;
392                   Tpc := 3;
393                   Cpd := 350;
394                   Capacity := TenMBSize;
395                   DrType := RevA;
396                   spares[1] := 7;
397                   DrSize := OldTenMB;
398                   END;
399           END; {SetRevA}
400
401       PROCEDURE SetDrv;
402           Var i: integer;
403           BEGIN
404           FOR i := 1 TO DrMax DO BEGIN
405              xcv.sln := 2; xcv.rln := 129;
406              xcv.b[1] := 16;    {status command}
407              xcv.b[2] := i;
408              CDSend (NetLoc,xcv); CDRecv (NetLoc,xcv);
409              IF ORD(xcv.c[1]) > 127
410                   THEN WITH PhysDrives[i] DO BEGIN
411                        DrType := Nodrv;
412                        PhysDr := FALSE;
413                        Capacity := 0;
414                        RomVers := 0;  FirmVers := 0;
415                        FirmMsg := '        ';
416                        END
417                   ELSE WITH PhysDrives[i] DO BEGIN
418                        NumDrives := i;
419                        Spt := ORD(xcv.c[35]);
420                        Tpc := ORD(xcv.c[36]);
421                        Cpd := ORD(xcv.c[38]);
422                        x   := ORD(xcv.c[37]);
423                        Cpd := (Cpd*256)+x;
424
425                      IF Cpd = 358 THEN
426                        BEGIN
427                           DrType := RevB;
428                           MaxSpTrk := 7;
429                           DrSize := TenMB; END   ELSE
430                         IF Cpd = 144 THEN
431                           BEGIN
432                              DrType := RevB;
433                              MaxSpTrk := 7;
```

```
434.                              DrSize := FiveMB; END  ELSE
435.                             IF Cpd = 388 THEN
436.                               BEGIN
437.                                DrType := RevB;
438.                                MaxSpTrk:= 7;
439.                                DrSize := TwentyMB; END ELSE
440.
441.                               IF cpd = 306 THEN
442.                                 BEGIN
443.                                   DrType := RevH;
444.                                   IF Tpc = 2 THEN
445.                                    BEGIN
446.                                     MaxSpTrk:= 31;
447.                                     DrSize := FiveMB; END ELSE
448.                                   IF Tpc = 4 THEN
449.                                     BEGIN
450.                                      MaxSpTrk:= 31;
451.                                      DrSize := TenMB; END ELSE
452.                                   IF Tpc = 6 THEN
453.                                       BEGIN
454.                                        MaxSpTrk:= 31;
455.                                        DrSize := TwentyMB;
456.                                 END;
457.                         END;
458.                        IF xcv.b[107] = 1 THEN BEGIN
459.                            PhysDr := TRUE;
460.                            ByteLInt (Capacity,0,xcv.b[41],xcv.b[40],xcv.b[3!
461.                            END
462.                         ELSE BEGIN
463.                            PhysDr := FALSE;
464.                            ByteLInt (Capacity,0,xcv.b[110],xcv.b[109],xcv.b!
465.                            END;
466.                        ROMvers := ORD(xcv.c[34]);
467.                        FirmVers := ORD(xcv.c[33]);
468.                        FirmMsg := '          ';
469. {!CC}                  MOVELEFT(xcv.b[1],FirmMsg,9);
470. {!CC}                  FirmMsg[0] := CHR(8);
471.                        END;
472.                 spares[i] := MaxSpTrk;
473.               END; {FOR}
474.           END; {SetDrv}
475.
476.     BEGIN
477.     {                                                    }
478.     { send old reset command to determine drive type }
479.     {                                                    }
480.     xcv.sln := 1;  xcv.rln := 1;
481.     xcv.b[1] := 0;
482.     CDSend (NetLoc,xcv); CDRecv (NetLoc,xcv);
483.     IF ORD(xcv.c[1]) > 127 then SetDrv
484.                            else SetRevA;
485.     END; {DrvInit}
486.
487. {$P}
```

```
488. {-------------------------------------------------------------------------!
489. { Procedure:    CCdrvIOinit                                                !
490. {                                                                          !
491. { Description: CCdrvIO unit initialization                                 !
492. {                                                                          !
493. {-------------------------------------------------------------------------!
494.
495. PROCEDURE CCdrvIOinit;
496.     BEGIN END;
497.
498. END.
499.
500.
501.
502.
```

```
{ PIPES.TEXT ----------------------------------------------------------}
{
{       PIPES -- Corvus Disk Pipes Unit
{
{       Copyright 1982 Corvus Systems, Inc.
{                      San Jose, California
{
{       All Rights Reserved
{
{       v 1.0  01-08-82  LEF  Original unit (taken from PIPES by PHB)
{       v 1.1  03-24-82  LEF  Added OMNINET support
{       v 1.2  06-16-82  DP   Const II mods, clean-up
{       v 1.3  07-31-82  DP   Changes PIPESINIT parameters to LONGINT
{
{---------------------------------------------------------------------}

{!CC} UNIT CCpipes;

INTERFACE

USES
{!CC} {$U CCLIB} CCDefn, CCLngInt,
{!CC} {$U C2LIB} CCDrvIO;

CONST
    PipesVersion     = '1.3'; {current version number}
    PnameLen         = 8;     {size of a pipe name}

    {pipe return codes . . .}
    PipeOk           = 0;     {successful return code}
    PipeEmpty        = -8;    {tried to read an empty pipe}
    PipeNotOpen      = -9;    {pipe was not open for read or write}
    PipeFull         = -10;   {tried to  write to a full pipe}
    PipeOpErr        = -11;   {tried to open (for reading) an open pipe}
    PipeNotThere     = -12;   {pipe does not exist}
    PipeNoRoom       = -13;   {the pipe data structures are full, and the'
                               is no room for new pipes at the moment. . .}
    PipeBadCmd       = -14;   {illegal command}
    PipesNotInitted = -15;   {pipes not initialized}
    {an error code less than -127 is a fatal disk error}
    PipeDskErr       = -255;

TYPE
    PNameStr  = STRING[PnameLen];

VAR
    PipeDebug:    BOOLEAN;

{$P}
```

```
50. FUNCTION pipestatus (VAR names,ptrs: DrvBlk): INTEGER;
51. FUNCTION pipeoprd   (pname: PNameStr): INTEGER;
52. FUNCTION pipeopwr   (pname: PNameStr): INTEGER;
53. FUNCTION pipeclrd   (npipe: INTEGER): INTEGER;
54. FUNCTION pipeclwr   (npipe: INTEGER): INTEGER;
55. FUNCTION pipepurge  (npipe: INTEGER): INTEGER;
56. FUNCTION piperead   (npipe: INTEGER; VAR info: DrvBlk): INTEGER;
57. FUNCTION pipewrite  (npipe,wlen: INTEGER; VAR info: DrvBlk): INTEGER;
58. FUNCTION pipesinit  (baddr,bsize: LONGINT): INTEGER;
59. PROCEDURE CCpipeinit(Netloc: CDaddr);
60.
61. IMPLEMENTATION
62.
63. CONST
64.          FiveByte = 26;   {=$1A, indicates a four byte opcode}
65.          TenByte  = 27;   {=$1B, ten byte opcode...}
66.
67.     {the following constants are used to select the type of request}
68.          OpnRd    = 192; {open pipe for read = $C0 }
69.          OpnWt    = 128; {open for write     = $80 }
70.
71.          Rd       = 32;   {=$20, read pipe}
72.          Wrt      = 33;   {=$21, write pipe}
73.
74.          Close    = 64;   {=$40, close read or close write}
75.          Status   = 65;   {=$41, pipe status command}
76.
77.          PInit    = 160; {initialize the pipes... = $A0 }
78.
79.     {pipe state constants...}
80.          ClsWt    = 254; { Close write = $FE }
81.          ClsRd    = 253; { Close read  = $FD }
82.          Purge    = 0;
83.
84. TYPE PipeName = PACKED ARRAY [1..PnameLen] OF CHAR;
85.
86. VAR  rcode:   INTEGER;
87.      pbuf:    SndRcvStr;
88.      PipeNetloc: CDaddr;
89.
90. {$P}
```

```
 91. FUNCTION result: INTEGER;
 92. {************************************************************}
 93. { result - sends the command in pbuf to the drive and receives  }
 94. { the results.... all pipe or disk errors are negative numbers  }
 95. { here....                                                      }
 96. { ************************************************************}
 97.     VAR status: INTEGER;
 98.     BEGIN
 99.     WITH pbuf DO BEGIN
100.         IF pipedebug THEN WRITE ('req =',b[1]:5,b[2]:5,'    ');
101.         CDsend (PipeNetloc, pbuf); CDrecv (PipeNetloc, pbuf);
102.         IF pipedebug THEN WRITE ('rec =',b[1]:5,b[2]:5,'    ');
103.         IF ord(c[1]) {dcode} > 127
104.             THEN status := ord(c[1]) {dcode}
105.             ELSE status := ord(c[2])*(-1) {ppcode};
106.         IF pipedebug THEN WRITELN ('res =',status:6);
107.         result := status;
108.         END;
109.     END;
110.
111. {$P}
```

```
112  PROCEDURE getname (src: PNameStr; dest: INTEGER);
113  {***************************************************************}
114  { getname - modifies dest so that it is exactly len chars long.  }
115  { if src is less than len characters long, dest is padded with   }
116  { blanks   if src is longer than len chars, dest is the first    }
117  { len chars of src.....                                          }
118  {***************************************************************}
119      VAR n: INTEGER;
120      BEGIN
121      FOR n := 1 TO Pnamelen DO
122          IF n <= LENGTH(src) THEN pbuf.c[dest-1+n] := src[n]
123                              ELSE pbuf.c[dest-1+n] := ' ';
124      END;
125
126  FUNCTION pipestatus;
127  {***************************************************************}
128  { FUNCTION pipestatus (VAR names,ptrs: DrvBlk  ): INTEGER;       }
129  { pipestatus determines the status of the pipes by reading the   }
130  { name and pointer tables from the disk.  Each table is 512      }
131  { bytes in length, so 1024 data bytes are returned...            }
132  {***************************************************************}
133      VAR i: INTEGER; initnames: string[16];
134      BEGIN
135      WITH pbuf DO BEGIN
136          sln  := 5;
137          rln  := 513;
138          b[1] := FiveByte;      {size}
139          b[2] := ord(Status);   {command}
140          b[3] := 1; b[4] := 0; b[5] := 0;
141          END; {WITH}
142      CDsend (PipeNetloc,pbuf); CDrecv (PipeNetloc,pbuf);
143      IF pipedebug THEN BEGIN WRITELN('pipe names');
144         FOR i:= 1 to 33 do write(pbuf.b[i]); writeln; end;
145      rcode := ORD(pbuf.c[1]) {dcode};
146      IF rcode < 128 THEN BEGIN
147         rcode := 0;   {possible soft error, so ignore}
148         MOVELEFT (pbuf.b[2],  names.b[1], DrvBlkSize);
149         IF pbuf.sln<3 THEN rcode := -ORD(pbuf.c[2]) ELSE
150            BEGIN
151            initnames := 'WOOFWOOFFOOWFOOW';
152            FOR i := 1 TO 8 DO BEGIN
153               IF names.c[i] <> initnames[i] THEN rcode := pipesnotinitte!
154               IF names.c[i+504] <> initnames[i+8] THEN rcode := pipesnot!
155               END;
156            END;
157         END;
158
159      IF rcode=0 THEN BEGIN
160         WITH pbuf DO BEGIN
161            sln  := 5;
162            rln  := 513;
163            b[1] := FiveByte;      {size}
164            b[2] := ord(Status);   {command}
165            b[3] := 2; b[4] := 0; b[5] := 0;
```

```
166.            END; {WITH}
167.         CDsend (PipeNetloc,pbuf); CDrecv (PipeNetloc,pbuf);
168.         IF pipedebug THEN BEGIN WRITELN('pipe ptrs'); FOR i:= 1 to 33 do
169.         write(pbuf.b[i]); writeln; end;
170.         rcode := ORD(pbuf.c[1]) {dcode};
171.         IF rcode < 127 THEN BEGIN
172.            rcode := 0;  {possible soft error, so ignore}
173.            MOVELEFT (pbuf.b[2],ptrs.b[1], DrvBlkSize);
174.            IF pbuf.sln<3 THEN rcode := -ORD(pbuf.c[2]);
175.            END;
176.         END;
177.
178.      pipestatus := rcode;
179.      END; {pipestatus}
180.
181.  {$P}
```

```
182. FUNCTION pipeoprd;
183. { ************************************************************ }
184. { FUNCTION pipeoprd (pname: STRING): INTEGER                    }
185. { Opens pipe pname for reading.  A pipe may not be open for both }
186. { read and write.  IF spooling is true then the entire pipe list }
187. { searched until the name matches and the pipe is closed for read}
188. { If spooling is false then we only try to open the first one    }
189. { which matches.....                                            }
190. { Returns the pipe number if successful, an error code otherwise.}
191. { ************************************************************ }
192.     BEGIN
193.     WITH pbuf DO BEGIN
194.         sln  := 10;
195.         rln  := 12;
196.         b[1] := TenByte;        {size}
197.         c[2] := CHR(OpnRd);     {command}
198.         getname (pname,3);      {pipe name}
199.         END; {WITH}
200.     rcode := result;
201.     IF rcode < 0
202.         THEN pipeoprd := rcode
203.         ELSE pipeoprd := pbuf.b[3] {pipeno};
204.     END; {pipeoprd}
205.
206. FUNCTION pipeopwr;
207. { ************************************************************ }
208. { FUNCTION pipeopwr (pname: STRING): INTEGER;                   }
209. { Open a pipe for writing.  Always allocates a new pipe.        }
210. { Returns the pipe number or an error code...                   }
211. { ************************************************************ }
212.     BEGIN
213.     WITH pbuf DO BEGIN
214. {$R-}    sln  := 10;
215.         rln  := 12;
216.         b[1] := TenByte;        {size}
217.         c[2] := CHR(OpnWt);     {command}
218.         getname (pname,3);      {pipe name}
219. {$R+}    END; {WITH}
220.     rcode := result;
221.     IF rcode < 0
222.         THEN pipeopwr := rcode
223.         ELSE pipeopwr := pbuf.b[3] {pipeno};
224.     END; {pipeopwr}
225.
226. {$P}
```

```
227. FUNCTION closeit (npipe: INTEGER; which: BYTE): INTEGER;
228. { ********************************************************** }
229. { closeit closes pipes for read, write, or purge depending on  }
230. { the value of which....                                       }
231. { Returns OkCode if successful, error code otherwise.         }
232. { ********************************************************** }
233.     BEGIN
234.     WITH pbuf DO BEGIN
235. {$R-}    sln  := 5;
236.          rln  := 12;
237.          b[1] := FiveByte;      {size}
238.          b[2] := ord(Close);    {command}
239.          b[3] := npipe;         {pipenum}
240.          b[4] := ord(which);    {state}
241.          b[5] := 0;
242. {$R+}    END; {WITH}
243.     closeit := result;
244.     END;  {closeit}
245.
246. FUNCTION pipeclrd;
247. { ********************************************************** }
248. { FUNCTION pipeclrd (npipe: INTEGER): INTEGER;                }
249. { close a pipe for reading.  IF the pipe is empty, it will be  }
250. { deallocated.... Returns an error code.                      }
251. { ********************************************************** }
252.     BEGIN pipeclrd := closeit (npipe,ClsRd); END;
253.
254. FUNCTION pipeclwr;
255. { ********************************************************** }
256. { FUNCTION pipeclwr (npipe: INTEGER): INTEGER;                }
257. { close a pipe for writing...                                 }
258. { ********************************************************** }
259.     BEGIN pipeclwr := closeit (npipe,ClsWt); END;
260.
261. FUNCTION pipepurge;
262. { ********************************************************** }
263. { FUNCTION pipepurge (npipe: INTEGER): INTEGER; delete a pipe  }
264. { ********************************************************** }
265.     BEGIN pipepurge := closeit (npipe,Purge); END;
266.
267. {$P}
```

```
268. FUNCTION pipewrite;
269. { ************************************************************************}
270. { FUNCTION pipewrite (npipe,wlen: INTEGER; info: DrvBlk): INTEGER;       }
271. { Write wlen bytes to pipe number npipe.  0 < wlen <= 512                 }
272. { Returns the number of bytes written or an error code.                  }
273. { ************************************************************************}
274.     BEGIN
275.     WITH pbuf DO BEGIN
276.         sln  := wlen+5;
277.         rln  := 12;
278.         b[1] := FiveByte;       {size}
279.         b[2] := Wrt;            {command}
280.         b[3] := npipe;          {pipenum}
281.         b[4] := wlen MOD 256;   {len.lo}
282.         b[5] := wlen DIV 256;   {len.hi}
283.         END; {WITH}
284.     MOVELEFT (info.b[1],pbuf.b[6],wlen);
285.     rcode := result;
286.     IF rcode < 0
287.         THEN pipewrite := rcode
288.         ELSE pipewrite := pbuf.b[4]*256+pbuf.b[3] {len};
289.     END; {pipewrite}
290.
291. FUNCTION piperead;
292. { ************************************************************************}
293. { FUNCTION piperead (npipe: INTEGER; VAR info: DrvBlk  ): INTEGER;       }
294. { Read upto 512 bytes from pipe npipe.                                     }
295. { Returns number of bytes read or error code.                             }
296. { ************************************************************************}
297.     BEGIN
298.     WITH pbuf DO BEGIN
299.         sln  := 5;
300.         rln  := 516;
301.         b[1] := FiveByte;       {size}
302.         b[2] := Rd;             {command}
303.         b[3] := npipe;          {pipenum}
304.         b[4] := 0;              {len.lo}
305.         b[5] := 2;              {len.hi}
306.         END; {WITH}
307.     rcode := result;
308.     IF rcode >= 0 THEN BEGIN
309.         rcode := pbuf.b[4]*256+pbuf.b[3] {len};
310.         MOVELEFT (pbuf.b[5],info.b[1],rcode);
311.         END;
312.     piperead := rcode;
313.     END; {piperead}
314.
315. {$P}
```

```
316  FUNCTION pipesinit;
317  { ************************************************************** }
318  { FUNCTION pipesinit (baddr,bsize: INTEGER): INTEGER;             }
319  { initialize the pipe data structures.  baddr is the block number  }
320  { of the start of the pipe buffer, bsize is the length in blocks.  }
321  { ************************************************************** }
322      BEGIN
323      IF ((baddr < 0) OR (bsize < 0)) THEN BEGIN
324          {allow negative numbers if you want to start at > 32k}
325          pipesinit := PipeDskErr;
326          EXIT (pipesinit);
327          END;
328      WITH pbuf DO BEGIN
329  {$R-}   sln  := 10;
330          rln  := 12;
331          b[1] := TenByte;          {size}
332          b[2] := ord(PInit);       {command}
333          b[3] := LIntByte(3, baddr);       {addr.lo}
334          b[4] := LIntByte(2, baddr);       {addr.hi}
335          b[5] := LIntByte(3, bsize);       {bufsize.lo}
336          b[6] := LIntByte(2, bsize);       {bufsize.hi}
337  {$R+}   END, {WITH}
338      pipesinit := result;
339      END;
340
341  PROCEDURE CCpipeinit {(Netloc: CDaddr)};
342      BEGIN
343      pipeDebug := FALSE;
344      PipeNetloc := Netloc;
345      END;
346
347  END.
348
```

```
    1. { SEMA4.TEXT ------------------------------------------------------------}
    2. {
    3. {        SEMA4 -- Corvus Disk Sema4s Unit
    4. {
    5. {        Copyright 1982 by Corvus Systems, Inc.
    6. {                           San Jose, California
    7. {
    8. {        All Rights Reserved
    9. {
   10. {        v 1.0  01-08-82  LEF  Original unit (taken from SEMA4 by PHB)
   11. {        v 1.1  06-15-82  DP   Const II mods, clean-up
   12. {
   13. {----------------------------------------------------------------------}
   14.
   15. {!CC} UNIT CCSEMA4;
   16.
   17. INTERFACE
   18.
   19. USES
   20. {!CC} {$U CCLIB} CCdefn,
   21. {!CC} {$U C2LIB} CCdrvIO;
   22.
   23. CONST
   24.    Sema4Rev    = '1.1';
   25.
   26.    { Return codes for the semaphore unit.                                !
   27.    {    negative function return values indicate error conditions        !
   28.    {    0 return means no error (and not set prior to operation)         !
   29.    {    $80 (128) return means key set prior to operation                !
   30.
   31.    SemWasSet  = 128;   { the prior state of this semaphore was locked    !
   32.    SemNotSet  = 0;     { prior state was unlocked                        !
   33.    SemFull    = -253;  { semaphore table is full (32 active semaphores) !
   34.    SemDskErr  = -255;  { disk error during write thru                    !
   35.
   36. TYPE
   37.    SemStr      = STRING[8];
   38.    SemKeys     = PACKED ARRAY [1..8] OF CHAR;
   39.    SemKeyList = RECORD CASE integer OF
   40.                    1: (skey:         ARRAY [1..32]  OF SemKeys);
   41. {!CC}              2: (sbyt:         ARRAY [1..256] OF byte);
   42.                    END;
   43.
   44. VAR
   45.    Sema4debug:   BOOLEAN;
   46.
   47. {$P}
```

```
48.  FUNCTION SemLock   (key: SemStr): INTEGER;
49.  FUNCTION SemUnlock (key: SemStr): INTEGER;
50.  FUNCTION SemClear: INTEGER;
51.  FUNCTION SemStatus (VAR kbuf: SemKeyList): INTEGER;
52.  PROCEDURE CCSema4Init(Netloc: CDaddr);
53.
54.
55.  IMPLEMENTATION
56.
57.  VAR
58.     xcv: SndRcvstr;
59.     SemNetloc: CDaddr;
60.
61.  {$P}
```

```
62.  { ************************************************************ }
63.  { SemClear sends a command which inititalizes the semaphore table }
64.  { to blanks...                                                  }
65.  { ************************************************************ }
66.  FUNCTION SemClear;
67.      BEGIN
68.      WITH xcv DO BEGIN
69.          sln  := 5; xcv.rln := 2;
70.          b[1] := 26; {5 byte commands are now 1A}{vs. A in rev A drives}
71.          b[2] := 16;
72.          b[3] := 0;  {don't care about the rest of the bytes...}
73.          b[4] := 0;
74.          b[5] := 0;
75.          END;
76.      CDsend(SemNetLoc,xcv); CDrecv(SemNetLoc,xcv);
77.      IF sema4debug THEN writeln('sem clear:  ',xcv.b[1],',',xcv.b[2]);
78.      IF ORD(xcv.c[1]) > 127
79.          THEN SemClear := -ORD(xcv.c[1])
80.          ELSE SemClear := 0;
81.      END; { SemClear }
82.
83.  {$P}
```

```
 84. FUNCTION ComKey (key: SemStr): INTEGER;
 85.     VAR i: INTEGER;
 86.     BEGIN
 87.     WITH xcv DO BEGIN
 88.         sln  := 10;   xcv.rln := 12;
 89.         b[1] := 11;
 90.         FOR i := 1 TO 8 DO
 91.             IF i <= LENGTH(key)
 92.                 THEN c[i+2] := key[i]
 93.                 ELSE c[i+2] := ' ';
 94.         END;
 95.     CDsend(SemNetLoc,xcv); CDrecv(SemNetLoc,xcv);
 96.     IF Sema4debug THEN WRITELN('comkey results:   ',xcv.b[1],',',xcv.b[2!
 97.     IF ORD(xcv.c[1]) > 127
 98.         THEN ComKey := -ORD(xcv.c[1])
 99.         ELSE IF ORD(xcv.c[2]) > 127
100.                 THEN ComKey := -ORD(xcv.c[2])
101.                 ELSE ComKey :=  ORD(xcv.c[2]);
102.
103.     END;
104.
105.
106. { ********************************************************* }
107. { FUNCTION SemLock (key: SemStr): INTEGER;                   }
108. { KEY is an eight character string which is written into the }
109. { semaphore table IF it was not there already...             }
110. { ********************************************************* }
111. FUNCTION SemLock;
112.     BEGIN
113.     xcv.b[2] := 1;
114.     SemLock := ComKey (key);
115.     END;
116.
117.
118. { ********************************************************* }
119. { FUNCTION SemUnlock (key: SemStr): INTEGER;                 }
120. { delete a key from the sem table and indicate whether or not}
121. { it was there before.... return codes are described above.  }
122. { ********************************************************* }
123. FUNCTION SemUnlock;
124.     BEGIN
125.     xcv.b[2] := 17;
126.     SemUnlock := ComKey (key);
127.     END;
128.
129. {$P}
```

```
130.  {*********************************************************}
131.  { FUNCTION SemStatus (kbuf: SemKeyList): INTEGER;          }
132.  { returns the actual semaphore table                       }
133.  {*********************************************************}
134.  FUNCTION SemStatus;
135.     BEGIN
136.     xcv.sln  := 5;  xcv.rln := 257;
137.     xcv.b[1] := 26;
138.     xcv.b[2] := 65;
139.     xcv.b[3] := 3;
140.     xcv.b[4] := 0;
141.     xcv.b[5] := 0;
142.     CDsend(SemNetLoc,xcv); CDrecv(SemNetLoc,xcv);
143.     IF sema4debug then writeln('sem status:  ',xcv.b[1],',',xcv.b[2]);
144.     IF ORD(xcv.c[1]) > 127 THEN
145.        BEGIN
146.        SemStatus := -ORD(xcv.c[1]);
147.        EXIT (SemStatus);
148.        END;
149.     MOVELEFT (xcv.b[2],kbuf.sbyt[1],256);
150.     SemStatus := 0;
151.     END;
152.
153.
154.  PROCEDURE CCSema4Init ((Netloc: CDaddr));
155.     BEGIN
156.     Sema4debug := FALSE;
157.     SemNetloc := Netloc;
158.     END;
159.
160.  END.
161.
162.
```

```
                        1* , File: cclib.bit.text
                        2* , Date: 13-May-82
                        3*
                        4* ,
                        5* ; Corvus CONCEPT bit manipulation functions
                        6* ,
                        7*
                        8*          GLOBAL BITFLIP,BITSET,BITCLEAR,BITTEST,SHIFTRT,SHIFTLT,MAKEBYTE
                        9*
                       10* ,
                       11* , Function BitFlip (data,bitnum: integer): integer;
                       12* ,
0000  205F             13* BITFLIP MOVE.L  (SP)+,A0         ; A0 = return address
0002  4C9F  0003       14*         MOVEM.W (SP)+,D0-D1      ; D0 = bit nmbr, D1 = data word
0006  0141             15*         BCHG    D0,D1            ; flip the bit
0008  3E81             16*         MOVE.W  D1,(SP)          ; place changed word on stack
000A  4ED0             17*         JMP     (A0)             ; return to Pascal
                       18*
                       19* ,
                       20* , Function BitSet (data,bitnum: integer): integer;
                       21* ,
000C  205F             22* BITSET  MOVE.L  (SP)+,A0         ; A0 = return address
000E  4C9F  0003       23*         MOVEM.W (SP)+,D0-D1      ; D0 = bit nmbr, D1 = data word
0012  01C1             24*         BSET    D0,D1            ; set the bit
0014  3E81             25*         MOVE.W  D1,(SP)          ; place changed word on stack
0016  4ED0             26*         JMP     (A0)             ; return to Pascal
                       27*
                       28* ,
                       29* , Function BitClear (data,bitnum: integer): integer;
                       30* ,
0018                   31* BITCLEAR
0018  205F             32*         MOVE.L  (SP)+,A0         ; A0 = return address
001A  4C9F  0003       33*         MOVEM.W (SP)+,D0-D1      ; D0 = bit nmbr, D1 = data word
001E  0181             34*         BCLR    D0,D1            ; clear the bit
0020  3E81             35*         MOVE.W  D1,(SP)          ; place changed word on stack
0022  4ED0             36*         JMP     (A0)             ; return to Pascal
                       37*
                       38* ,
                       39* , Function BitTest (data,bitnum: integer): boolean;
                       40* ,
0024  205F             41* BITTEST MOVE.L  (SP)+,A0         ; A0 = return address
0026  4C9F  0003       42*         MOVEM.W (SP)+,D0-D1      ; D0 = bit nmbr, D1 = data word
002A  4257             43*         CLR.W   (SP)             ; assume false = 0
002C  0101             44*         BTST    D0,D1            ; test the bit
002E  6704             45*         BEQ.S   BTI              ;
0030  1EBC  0001       46*         MOVE.B  #1,(SP)          ; bit is on, return true
0034  4ED0             47* BTI     JMP     (A0)             ; return to Pascal
                       48*
```

```
                        50* ;
                        51* ; Function ShiftRt (data: integer): integer;
                        52* ;
0036  205F              53* SHIFTRT MOVE.L  (SP)+,A0         ; A0 = return address
0038  301F              54*         MOVE.W  (SP)+,D0         ; D0 = word to be shifted
003A  E248              55*         LSR.W   #1,D0            ; shift it right
003C  3E80              56*         MOVE.W  D0,(SP)          ; push result on stack
003E  4ED0              57*         JMP     (A0)             ; return to Pascal
                        58*
                        59* ,
                        60* ; Function ShiftLt (data: integer): integer,
                        61* ,
0040  205F              62* SHIFTLT MOVE.L  (SP)+,A0         ; A0 = return address
0042  301F              63*         MOVE.W  (SP)+,D0         ; D0 = word to be shifted
0044  E348              64*         LSL.W   #1,D0            ; shift it left
0046  3E80              65*         MOVE.W  D0,(SP)          ; push result on stack
0048  4ED0              66*         JMP     (A0)             ; return to Pascal
                        67*
                        68* ,
                        69* ; Function MakeByte (n: integer): byte;
                        70* ,
004A                    71* MAKEBYTE
004A  205F              72*         MOVE.L  (SP)+,A0
004C  301F              73*         MOVE.W  (SP)+,D0         ; get n
004E  1E80              74*         MOVE.B  D0,(SP)          ; return function value
0050  4ED0              75*         JMP     (A0)             ; return to Pascal
                        76*
                        77*         END

*BITCLEAR   000018+ *BITSET     00000C+  BIT        000034+ *SHIFTLT    000040+
*BITFLIP    000000+ *BITTEST    000024+ *MAKEBYTE   00004A+ *SHIFTRT    000036+

0 errors  77 lines  File CCLIB.BIT.TEXT
```

```
1* , File: cclib.asm.text
2* , Date: 11-Jan-83
3*
4* ;
5* , Corvus CONCEPT operating system interface
6* ;
7*
8*          IDENT   CCLIBASM
9*          GLOBAL  OSactSit,OSactSrv,OSaltSit,OSaltSrv,OSvrtCrt
10*         GLOBAL  OSsitType,OSdevType,OSsysSize,OScurSP
11*         GLOBAL  OSstrmDv,OSprtrDv
12*         GLOBAL  OSmaxDev,OSdispDv,OSkybdDv,OStimDv
13*         GLOBAL  OSomniDv,OSdcm2Dv,OSdcm1Dv,OSsitDv,OSextCRT
14*         GLOBAL  pOSuserID,pOSsysWnd,pOScurWnd,pOScurKbd
15*         GLOBAL  pOSdevNam,pOSdate,pOSsysVol,pOScurVol
16*         GLOBAL  pOSsyswrs,pOSsysDat
17*         GLOBAL  xGetDir,xPutDir,KeyPress,BrkPress
18*
19*          include '/ccos/os.gbl.asm.text'
```

```
                    21* ,
                    22* ; File  os gbl asm text
                    23* ; Date  23-Jan-83
                    24* .
                    25* ,
                    26* ; Corvus CONCEPT operating system data structure equates
                    27* .
                    28*
                    29* ,
                    30* ; Additional Corvus CONCEPT I/O result codes
                    31* ,
00000000            32* IOok        equ 0   ;Good result, no error
00000001            33* IOEinvdev   equ 2   ;Invalid unit number/invalid device
00000003            34* IOEioreq    equ 3   ;Invalid I/O request
                    35*                     ;
00000015            36* IOEnotrn    equ 21  ;Transporter not ready
00000016            37* IOEtimot    equ 22  ;Timed out waiting for Omninet event
00000017            38* IOEnobuf    equ 23  ;Read without a valid write buffer
                    39*                     ;
00000020            40* IOEwndfn    equ 32  ;Invalid window function
00000021            41* IOEwndbe    equ 33  ;Window create boundary
00000022            42* IOEwndcs    equ 34  ;Invalid character set
00000023            43* IOEwnddc    equ 35  ;Delete current window
00000024            44* IOEwndds    equ 36  ;Delete system window
00000025            45* IOEwndnw    equ 37  ;Inactive window
00000026            46* IOEwndwr    equ 38  ;Invalid window record
00000027            47* IOEwndwn    equ 39  ;Invalid system window number
                    48*                     ;
00000028            49* IOEnodsp    equ 40  ;Display driver not available
00000029            50* IOEnokyb    equ 41  ;Keyboard driver not available
0000002A            51* IOEnotim    equ 42  ;Timer driver not available
0000002B            52* IOEnoomn    equ 43  ;OMNINET driver not available
0000002C            53* IOEnoprt    equ 44  ;Printer driver not available
0000002D            54* IOEnfdrv    equ 45  ;No floppy drive at slot
0000002E            55* IOEnodtc    equ 46  ;DataComm driver not available
                    56*                     ;
00000032            57* IOEtbinv    equ 50  ;Invalid table entry ID
00000033            58* IOEtbfll    equ 51  ;Table full
00000034            59* IOEtbinu    equ 52  ;Table entry in use
00000035            60* IOEkybte    equ 53  ;Keyboard transmission error
00000036            61* IOEuiopm    equ 54  ;Invalid unit I/O parameter
00000037            62* IOEprmln    equ 55  ;Invalid parameter block length
00000038            63* IOEfncod    equ 56  ;Invalid function code
00000039            64* IOEclkmf    equ 57  ;Clock (hardware) malfunction
                    65*                     ;
0000003C            66* IOEirdsbl   equ 60  ;Input to read buffer disabled
0000003D            67* IOEordsbl   equ 61  ;Output to read buffer disabled
0000003E            68* IOEiwdsbl   equ 62  ;Input to write buffer disabled
0000003F            69* IOEowdsbl   equ 63  ;Output to write buffer disabled
00000040            70* IOEbszerr   equ 64  ;Buffer size error
00000041            71* IOEwszerr   equ 65  ;Write size error
00000042            72* IOErszerr   equ 66  ;Read size error
00000043            73* IOEuarter   equ 67  ;UART hardware error (overrun, parity, or framing)
00000044            74* IOEpaderr   equ 68  ;Proportional spacing error (excess pad chars req)
```

75*

```
                        77* ;
                        78* ; System Common Pointer
                        79* ;
00000180                80* pSysCom     equ  $0180   ;pointer to address of SYSCOM
00000184                81* SysKybdFlg equ  $0184   ;keyboard control flags
00000186                82* SysByteScn equ  $0186   ;display driver - bytes per scan line
                        83*
                        84* ;
                        85* ; System Common Equates
                        86* ;
00000000                87* SCiorslt equ   0        ;word - I/O result
00000002                88* SCprocno equ   2        ;word - current process number
00000004                89* SCfreehp equ   4        ;lint - free heap pointer
00000008                90* SCjtable equ   8        ;lint - jump table pointer
0000000C                91* SCsysout equ   12       ;lint - default output file pointer
00000010                92* SCsysin  equ   16       ;lint - default input file pointer
00000014                93* SCdevtab equ   20       ;lint - device (unit) table pointer
00000018                94* SCdirnam equ   24       ;lint - directory name string pointer
0000001C                95* SCutable equ   28       ;lint - user table pointer
00000020                96* SCtoday  equ   32       ;word - system date
00000022                97* SCcodejt equ   34       ;lint - code jump table pointer
00000026                98* SCnxtpro equ   38       ;word - next process number
00000028                99* SCnumpro equ   40       ;word - number of processes
0000002A               100* SCprotbl equ   42       ;lint - process table pointer
0000002E               101* SCbootnm equ   46       ;lint - boot device name pointer
00000032               102* SCmemmap equ   50       ;lint - memory map pointer
00000036               103* SCbootdv equ   54       ;word - boot device number
                       104*                         ;
                       105*                         ; CONCEPT additions
                       106* ;        equ   56       ;word - unused
                       107* ;        equ   58       ;word - unused
0000003C               108* SCslttbl equ   60       ;lint - slot table pointer
00000040               109* SCrootw  equ   64       ;lint - root window record pointer
00000044               110* SCcurrw  equ   68       ;lint - current window record pointer
00000048               111* SCcurrk  equ   72       ;lint - current keyboard record pointer
0000004C               112* SCuserid equ   76       ;word - Constellation user ID
0000004E               113* SCvrsnbr equ   78       ;lint - current version number string pointer
00000052               114* SCvrsdat equ   82       ;lint - current version date string pointer
00000056               115* SCwndtbl equ   86       ;lint - window table pointer
0000005A               116* SCsusinh equ   90       ;word - suspend inhibit count
0000005C               117* SCsusreq equ   92       ;word - suspend request if non-zero
                       118*
```

```
                        120* ;
                        121* ; System Vector Equates
                        122* ;
00000000                123* SVuwrite equ     0*4     ;unit write
00000004                124* SVuread  equ     1*4     ;unit read
00000008                125* SVuclear equ     2*4     ;unit clear
0000000C                126* SVubusy  equ     3*4     ;unit busy
00000010                127* SVput    equ     4*4     ;put
00000014                128* SVget    equ     5*4     ;get
00000018                129* SVinit   equ     6*4     ;init
0000001C                130* SVopen   equ     7*4     ;open
00000020                131* SVclose  equ     8*4     ;close
00000024                132* SVwrchar equ     9*4     ;writechar
00000028                133* SVrdchar equ     10*4    ;readchar
0000002C                134* SVblkio  equ     11*4    ;blockio
00000030                135* SVseek   equ     12*4    ;seek
00000034                136* SVnew    equ     13*4    ;new
00000038                137* SVdsp    equ     14*4    ;dispose
0000003C                138* SVmark   equ     15*4    ;mark
00000040                139* SVrlease equ     16*4    ;release
00000044                140* SVmavail equ     17*4    ;memory available
00000048                141* SVgetdir equ     18*4    ;get directory
0000004C                142* SVcrkpth equ     19*4    ;crack path name
00000050                143* SVustat  equ     20*4    ;unit status
00000054                144* SVnew4   equ     21*4    ;new     (longint)
00000058                145* SVdsp4   equ     22*4    ;dispose (longint)
                        146*                         ;
0000005C                147* SVcli    equ     23*4    ;command line interpreter
00000060                148* SVgetvnm equ     24*4    ;get volume names
00000064                149* SVvaldir equ     25*4    ;check valid directory
00000068                150* SVflpdir equ     26*4    ;flip directory
0000006C                151* SVschdir equ     27*4    ;search directory
00000070                152* SVdelent equ     28*4    ;delete directory entry
00000074                153* SVputdir equ     29*4    ;write directory
00000078                154* SVuinstl equ     30*4    ;unit install
                        155*
                        156* ;
                        157* ; Memory Map Equates
                        158* ;
00000000                159* MMlodta  equ     0       ;lint - low data pointer
00000004                160* MMhidta  equ     4       ;lint - high data pointer
00000008                161* MMlocod  equ     8       ;lint - low code pointer
0000000C                162* MMhicod  equ     12      ;lint - high code pointer
00000010                163* MMbtsw   equ     16      ;word - boot switches
00000012                164* MMbtdev  equ     18      ;word - boot device number
00000014                165* MMbtslt  equ     20      ;word - boot slot number
00000016                166* MMbtsrv  equ     22      ;word - boot server number
00000018                167* MMbtdrv  equ     24      ;word - boot drive number
0000001A                168* MMbtblk  equ     26      ;word - boot volume block number
                        169*
```

```
                        171*
                        172* ; Unit Table Equates
                        173*
00000002                174* UTiodrv equ     2       ,lint - I/O driver pointer
00000006                175* UTblf   equ     6       ,bool - blocked device flag
00000007                176* UTmtd   equ     7       ,bool - mounted device flag
00000008                177* UTdid   equ     8       ,str7 - device ID
00000010                178* UTsiz   equ     16      ,lint - device size
00000014                179* UTslt   equ     20      ,byte - device slot
00000015                180* UTsrv   equ     21      ,byte - device server
00000016                181* UTdrv   equ     22      ,byte - disk drive numbr
00000017                182* UTtyp   equ     23      ,byte - disk drive type
00000018                183* UTspt   equ     24      ,byte - sectors per track
00000019                184* UTtps   equ     25      ,byte - tracks per side
0000001A                185* UTro    equ     26      ,bool - device read only
0000001B                186* UTflp   equ     27      ,bool - volume directory flipped
0000001C                187* UTblk   equ     28      ,lint - disk base block
00000020                188* UTlen   equ     32      ,       entry length
                        189*
                        190* ,
                        191* ; Slot Table Equates
                        192* ,
00000000                193* STbtslt  equ     0       ,boot slot number
00000002                194* STbtsrv  equ     2       ,boot server number
00000004                195* STacslt  equ     4       ,active slot number
00000006                196* STacsrv  equ     6       ,active server number
00000008                197* STalslt  equ     8       ,alternate slot number
0000000A                198* STalsrv  equ     10      ,alternate server number
0000000C                199* STinfo   equ     12      ,array [1  8] of
                        200*                         ,
00000000                201* STnmbr   equ     0       ,   slot number (1-8)
00000001                202* STtype   equ     1       ,   device type (slottypes)
00000002                203* STndrv   equ     2       ,   number of drives
00000004                204* STinfoL  equ     4       ,   device info length
                        205*
```

```
                        207* .
                        208* , Character Set Record Equates
                        209* .
00000000                210* CStbiloc equ   0       ,character set data pointer
00000004                211* CSlpch   equ   4       ,scanlines per character (assume wide)
00000006                212* CSbpch   equ   6       ,bits per character (vertical height)
00000008                213* CSfrstch equ   8       ,first character code - ascii
0000000A                214* CSlastch equ   10      ,last character code - ascii
0000000C                215* CSmask   equ   12      ,mask used in positioning cells
00000010                216* CSattr1  equ   16      ,attributes
                        217*                       ,  bit 0 = 1 - vertical orientation
00000011                218* CSattr2  equ   17      ,currently unused
                        219*
                        220* ,
                        221* , Window Record Equates
                        222* ,
00000000                223* WRcharpt equ   0       ,character set pointer
00000004                224* WRhomept equ   4       ,home (upper left) pointer
00000008                225* WRcuradr equ   8       ,current location pointer
0000000C                226* WRhomeot equ   12      ,bit offset of home location
0000000E                227* WRbasex  equ   14      ,home x value, relative to root window
00000010                228* WRbasey  equ   16      ,home y value, relative to root window
00000012                229* WRlngthx equ   18      ,maximum x value, relative to window (bits)
00000014                230* WRlngthy equ   20      ,maximum y value, relative to window (bits)
00000016                231* WRcursx  equ   22      ,current x value (bits)
00000018                232* WRcursy  equ   24      ,current y value (bits)
0000001A                233* WRbitofs equ   26      ,bit offset of current address
0000001C                234* WRgrorgx equ   28      ,graphics - origin x (bits relative to home loc)
0000001E                235* WRgrorgy equ   30      ,graphics - origin y (bits relative to home loc)
00000020                236* WRattr1  equ   32      ,attributes
                        237*                       ,
00000000                238* inverse  equ   0       ,   inverse video mode
00000001                239* undscr   equ   1       ,   underscore mode
00000002                240* insmod   equ   2       ,   insert mode
00000003                241* viddefit equ   3       ,   0 = W on B,       1 = B on W
00000004                242* noautolf equ   4       ,   0 = auto LF w/CR, 1 = no auto LF
00000005                243* syswin   equ   5       ,   system defined window
00000006                244* active   equ   6       ,   active window
00000007                245* suspend  equ   7       ,   suspended window
                        246*                       ,
00000021                247* WRattr2  equ   33      ,attributes
                        248*                       ;
00000000                249* vert     equ   0       ,   1 = vertical,   0 = horizontal screen
00000001                250* graphic  equ   1       ,   1 = graphics,   0 = character mode
00000002                251* cursor   equ   2       ,   1 = cursor on,  0 = cursor off
00000003                252* invcurs  equ   3       ,   1 = inverse,    0 = underline cursor
00000004                253* wrapon   equ   4       ,   1 = wrap,       0 = clip at eoln
00000005                254* noscroll equ   5       ,   1 = no scroll,  0 = scroll
00000006                255* clrsc    equ   6       ,   1 = paging mode
00000007                256* vidset   equ   7       ,   1 = inverse     0 = normal
                        257*                       ,
00000022                258* WRstate  equ   34      ,used for decoding escape sequences
00000023                259* WRrcdlen equ   35      ,window description record length
00000024                260* WRattr3  equ   36      ,enhanced character set attributes
```

```
00000025          261* WRfill1  equ   37      ,currently unused
00000026          262* WRfill2  equ   38      ,currently unused
00000027          263* WRfill3  equ   39      ;currently unused
00000028          264* WRfill4  equ   40      ,currently unused
0000002C          265* WRwwsptr equ   44      ,window working storage pointer
                  266*                        ,
00000030          267* WRlength equ   48      ,actual window record length
                  268*
```

```
                        270*
                        271* ;
                        272* ; OSACTSLT - Get active slot function
                        273* ;
                        274* ; FUNCTION OSactSlt: integer;
                        275* ;
0000                    276* OSactSlt
0000  2278  0180        277*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0004  2269  003C        278*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0008  3F69  0004  0004  279*        move.w  STacslt(a1),4(sp)       ;Get active slot from slot table
000E  4E75              280*        rts                             ;Return
                        281*
                        282* ;
                        283* ; OSACTSRV - Get active server function
                        284* ;
                        285* ; FUNCTION OSactSrv: integer;
                        286* ;
0010                    287* OSactSrv
0010  2278  0180        288*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0014  2269  003C        289*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0018  3F69  0006  0004  290*        move.w  STacsrv(a1),4(sp)       ;Get active server from slot table
001E  4E75              291*        rts                             ;Return
                        292*
                        293* ;
                        294* ; OSALTSLT - Get alternate slot function
                        295* ;
                        296* ; FUNCTION OSaltSlt: integer;
                        297* ;
0020                    298* OSaltSlt
0020  2278  0180        299*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0024  2269  003C        300*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0028  3F69  0008  0004  301*        move.w  STalslt(a1),4(sp)       ;Get alternate slot from slot table
002E  4E75              302*        rts                             ;Return
                        303*
                        304* ;
                        305* ; OSALTSRV - Get alternate server function
                        306* ;
                        307* ; FUNCTION OSaltSrv: integer;
                        308* ;
0030                    309* OSaltSrv
0030  2278  0180        310*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0034  2269  003C        311*        move.l  SCslttbl(a1),a1         ;Get pointer to slot table
0038  3F69  000A  0004  312*        move.w  STalsrv(a1),4(sp)       ;Get alternate server from slot table
003E  4E75              313*        rts                             ;Return
                        314*
                        315* ;
                        316* ; OSVRTCRT - Get CRT orientation function
                        317* ;
                        318* ; FUNCTION OSvrtCrt: boolean; {TRUE if vertical, FALSE if horizontal}
                        319* ;
0040                    320* OSvrtCrt
0040  422F  0004        321*        clr.b   4(sp)                   ;Set function return to FALSE
0044  207C  0003  0F61  322*        movea.l #$30F61,a0              ;Get pointer to orientation switch
004A  0810  0003        323*        btst    #3,(a0)                 ;Vertical orientation?
```

```
004E 6700 0008          324*         beq     vrtcrtx             ;no, return
0052 1F7C 0001 0004     325*         move.b  #1,4(sp)            ;Set function return to TRUE
0056 4E75               326* vrtcrtx rts                         ;Return
                        327*
```

```
                        329* ;
                        330* ; OSSLTTYPE - Get device type for slot function
                        331* ;
                        332* ; FUNCTION OSsltType (slot: integer): slottype;
                        333* ;
005A                    334* OSsltType
005A  205F              335*        move.l  (sp)+,a0                ;Save return address
005C  301F              336*        move.w  (sp)+,d0                ;Get slot number
005E  5340              337*        subq.w  #1,d0                   ;Compute offset into slot table
0060  6D1C              338*        blt.s   slttyp8                 ;Error return if slot not valid
0062  0C40  0005        339*        cmpi.w  #5,d0                   ;*
0066  6C16              340*        bge.s   slttyp8                 ;Error return if slot not valid
0068  C0FC  0004        341*        mulu    #STinfoL,d0             ;*
006C  0640  000C        342*        addi.w  #STinfo,d0              ;*
0070  2278  0180        343*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
0074  2269  003C        344*        move.l  SCsltTbl(a1),a1         ;Get pointer to slot table
0078  1EB1  0001        345*        move.b  STtype(a1,d0.w),(sp)    ;Get slot type for slot
007C  6002              346*        bra.s   slttyp9                 ;Return
                        347*                                        ;
007E  4217              348* slttyp8 clr.b   (sp)                   ;Set slot type to no device
                        349*                                        ;
0080  4ED0              350* slttyp9 jmp     (a0)                    ;Return
                        351*
                        352* ;
                        353* ; OSDEVTYPE - Get device type for device function
                        354* ;
                        355* ; FUNCTION OSdevType (devno: integer): slottype;
                        356* ;
0082                    357* OSdevType
0082  205F              358*        move.l  (sp)+,a0                ;Save return address
0084  301F              359*        move.w  (sp)+,d0                ;Get device number
0086  C0FC  0020        360*        mulu    #UTlen,d0               ;Compute index into DevTab
008A  2278  0180        361*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
008E  2269  0014        362*        move.l  SCdevtab(a1),a1         ;Get pointer to device table
0092  D3FC  0000 0002   363*        adda.l  #2,a1                   ;Get pointer to device table entry
0098  D3C0              364*        adda.l  d0,a1                   ;*
009A  4241              365*        clr.w   d1                      ;Get slot number for device
009C  1229  0014        366*        move.b  UTslt(a1),d1            ;*
00A0  3F01              367*        move.w  d1,-(sp)                ;Push slot number
00A2  4850              368*        pea     (a0)                    ;Push return address
00A4  60B4              369*        bra.s   OSsltType               ;Get slot type for slot (device)
                        370*
                        371* ;
                        372* ; OSSYSSIZE - Get system size function
                        373* ;
                        374* ; FUNCTION OSsysSize: integer;
                        375* ;
00A6                    376* OSsysSize
00A6  3F7C  0100 0004   377*        move.w  #256,4(sp)              ;Set result to 256k
00AC  2278  0180        378*        move.l  pSysCom.w,a1            ;Get pointer to SysCom
00B0  2269  0032        379*        move.l  SCmemmap(a1),a1         ;Get pointer to memory map
00B4  0CA9  000C 0000   380*        cmpi.l  #$C0000,MMhicod(a1)     ;Is this a 512k system?
00BA  000C
00BC  6D06              381*        blt.s   ss1                     ;No, return
```

```
00BE  3F3C  0200  0004   382*          move.w  #512,4(sp)              ;Set result to 512t
00C4  4E75               383* ss1      rts                             ;Return
                         384*
                         385* ;
                         386* ; OSCURSP - Get current SP for system function
                         387* ;
                         388* ; FUNCTION OScurSP: longint;
                         389* ;
00C6  2278  0180         390* OScurSP  move.l  pSysCom.w,a1             ;Get pointer to SysCom
00CA  2269  0032         391*          move.l  SCmemmap(a1),a1         ;Get pointer to memory map
00CE  2F69  0004  0004   392*          move.l  MMhirdta(a1),4(sp)      ;Get current SP
00D4  4E75               393*          rts                             ;Return
                         394*
                         395* ;
                         396* ; OSEXTCRT - Check for external CRT function
                         397* ;
                         398* ; FUNCTION OSextCRT: boolean;
                         399* ;
00D6                     400* OSextCRT
00D6  205F               401*          move.l  (sp)+,a0                ;Save return address
00D8  548F               402*          addq.l  #2,sp                   ;Remove function result from stack
00DA  2278  0180         403*          move.l  pSysCom.w,a1            ;Get pointer to SysCom
00DE  2269  0014         404*          move.l  SCdevtab(a1),a1         ;Get pointer to device table
00E2  3019               405*          move.w  (a1)+,d0                ;Get number of devices
00E4  2449               406*          move.l  a1,a2                   ;Compute last device pointer
00E6  C0FC  0010         407*          mulu    #UTlen,d0               ;*
00EA  D5C0               408*          adda.l  d0,a2                   ;*
00EC  2269  0002         409*          move.l  UTiodrv(a1),a1          ;Get driver pointers
00F0  246A  0002         410*          move.l  UTiodrv(a2),a2          ;*
00F4  7001               411*          moveq   #1,d0                   ;Assume TRUE
00F6  B5C9               412*          cmpa.l  a1,a2                   ;Driver [0] = driver [MAXDEV]?
00F8  6700  0004         413*          beq     exortx                  ;Yes, return
00FC  7000               414*          moveq   #0,d0                   ;Set FALSE
00FE  1F00               415* exortx   move.b  d0,-(sp)                ;Set function result
0100  4ED0               416*          jmp     (a0)                    ;Return
                         417*
```

```
                        419*  ;
                        420*  , OSstrmDv - Get SYSTERM device number function
                        421*  ,
                        422*  ; FUNCTION OSstrmDv: integer;
                        423*  ,
0102                    424*  OSstrmDv
0102  3F7C  0002  0004  425*          move.w  #2,4(sp)                ;Set function result
0108  4E75              426*          rts                             ;Return
                        427*
                        428*  ,
                        429*  , OSprtrDv - Get PRINTER device number function
                        430*  ,
                        431*  , FUNCTION OSprtrDv: integer;
                        432*  ,
010A                    433*  OSprtrDv
010A  3F7C  0006  0004  434*          move.w  #6,4(sp)                ;Set function result
0110  4E75              435*          rts                             ;Return
                        436*
                        437*  ,
                        438*  , OSmaxDev - Get maximum device number function
                        439*  ,
                        440*  , FUNCTION OSmaxDev: integer;
                        441*  ,
0112                    442*  OSmaxDev
0112  2278  0180        443*          move.l  pSysCom.w,a1            ;Get pointer to SysCom
0116  2269  0014        444*          move.l  SCdevtab(a1),a1         ;Get pointer to device table
011A  3F51  0004        445*          move.w  (a1),4(sp)              ;Get number of devices
011E  4E75              446*          rts                             ;Return
                        447*
                        448*  ,
                        449*  , OSdispDv - Get DISPLAY driver device number function
                        450*  ;
                        451*  , FUNCTION OSdispDv: integer;
                        452*  ,
0120                    453*  OSdispDv
0120  4267              454*          clr.w   -(sp)                   ;Get number of devices
0122  61EE              455*          bsr.s   OSmaxDev                ;*
0124  301F              456*          move.w  (sp)+,d0                ;*
0126  3F40  0004        457*          move.w  d0,4(sp)                ;Set function result
012A  4E75              458*          rts                             ;Return
                        459*
                        460*  ,
                        461*  , OSkybdDv - Get KYBD driver device number function
                        462*  ,
                        463*  , FUNCTION OSkybdDv: integer;
                        464*  ,
012C                    465*  OSkybdDv
012C  4267              466*          clr.w   -(sp)                   ;Get number of devices
012E  61E2              467*          bsr.s   OSmaxDev                ;*
0130  301F              468*          move.w  (sp)+,d0                ;*
0132  5340              469*          subq    #1,d0                   ;Get device number
0134  3F40  0004        470*          move.w  d0,4(sp)                ;Set function result
0138  4E75              471*          rts                             ;Return
                        472*
```

```
                        473* ;
                        474* ; OStimDv - Get TIMER driver device number function
                        475* ;
                        476* ; FUNCTION OStimDv: integer;
                        477* ;
013A                    478* OStimDv
013A  4267              479*        clr.w   -(sp)                   ;Get number of devices
013C  61D4              480*        bsr.s   OSmaxDev                ;*
013E  301F              481*        move.w  (sp)+,d0                ;*
0140  5540              482*        subq    #2,d0                   ;Get device number
0142  3F40  0004        483*        move.w  d0,4(sp)                ;Set function result
0146  4E75              484*        rts                             ;Return
                        485*
                        486* ;
                        487* ; OSomniDv - Get OMNINET driver device number function
                        488* ;
                        489* ; FUNCTION OSomniDv: integer;
                        490* ;
0148                    491* OSomniDv
0148  4267              492*        clr.w   -(sp)                   ;Get number of devices
014A  61C6              493*        bsr.s   OSmaxDev                ;*
014C  301F              494*        move.w  (sp)+,d0                ;*
014E  5740              495*        subq    #3,d0                   ;Get device number
0150  3F40  0004        496*        move.w  d0,4(sp)                ;Set function result
0154  4E75              497*        rts                             ;Return
                        498*
                        499* ;
                        500* ; OSdcm2Dv - Get DTACOM2 driver device number function
                        501* ;
                        502* ; FUNCTION OSdcm2Dv: integer;
                        503* ;
0156                    504* OSdcm2Dv
0156  4267              505*        clr.w   -(sp)                   ;Get number of devices
0158  61B8              506*        bsr.s   OSmaxDev                ;*
015A  301F              507*        move.w  (sp)+,d0                ;*
015C  5940              508*        subq    #4,d0                   ;Get device number
015E  3F40  0004        509*        move.w  d0,4(sp)                ;Set function result
0162  4E75              510*        rts                             ;Return
                        511*
                        512* ;
                        513* ; OSdcm1Dv - Get DTACOM1 driver device number function
                        514* ;
                        515* ; FUNCTION OSdcm1Dv: integer;
                        516* ;
0164                    517* OSdcm1Dv
0164  4267              518*        clr.w   -(sp)                   ;Get number of devices
0166  61AA              519*        bsr.s   OSmaxDev                ;*
0168  301F              520*        move.w  (sp)+,d0                ;*
016A  5B40              521*        subq    #5,d0                   ;Get device number
016C  3F40  0004        522*        move.w  d0,4(sp)                ;Set function result
0170  4E75              523*        rts                             ;Return
                        524*
                        525* ;
                        526* ; OSsltDv - Get SLOTIO driver device number function
```

```
                        527*  ;
                        528*  ; FUNCTION OSsltDv: integer;
                        529*  ;
0172                    530*  OSsltDv
0172  4267               531*          clr.w   -(sp)                   ;Get number of devices
0174  619C               532*          bsr.s   OSmaxDev                ;*
0176  301F               533*          move.w  (sp)+,d0                ;*
0178  5D40               534*          subq    #6,d0                   ;Get device number
017A  3F40  0004          535*          move.w  d0,4(sp)                ;Set function result
017E  4E75                536*          rts                             ;Return
                        537*
```

```
                        539* ;
                        540* ; pOSuserID - Get Constellation user ID pointer
                        541* ;
                        542* ; FUNCTION pOSuserID: pointer;
                        543* ;
0180                    544* pOSuserID
0180  2F78 0180 0004    545*        move.l  pSysCom.w,4(sp)         ;Get pointer to SysCom
0186  06AF 0000 004C    546*        addi.l  #SCuserID,4(sp)         ;Get pointer to user ID
018C  0004
018E  4E75              547*        rts                             ;Return
                        548*
                        549* ;
                        550* ; pOScurKbd - Get current keyboard record pointer
                        551* ;
                        552* ; FUNCTION pOScurKbd: pointer;
                        553* ;
0190                    554* pOScurKbd
0190  2078 0180         555*        move.l  pSysCom.w,a0            ;Get pointer to SysCom
0194  2F68 0048 0004    556*        move.l  SCcurrk(a0),4(sp)       ;Get current keyboard pointer
019A  4E75              557*        rts                             ;Return
                        558*
                        559* ;
                        560* ; pOScurWnd - Get current window record pointer
                        561* ;
                        562* ; FUNCTION pOScurWnd: pointer;
                        563* ;
019C                    564* pOScurWnd
019C  2078 0180         565*        move.l  pSysCom.w,a0            ;Get pointer to SysCom
01A0  2F68 0044 0004    566*        move.l  SCcurrw(a0),4(sp)       ;Get current window pointer
01A6  4E75              567*        rts                             ;Return
                        568*
                        569* ;
                        570* ; pOSsysWnd - Get system window record pointer
                        571* ;
                        572* ; FUNCTION pOSsysWnd (wndnbr: integer): pointer;
                        573* ;
01A8                    574* pOSsysWnd
01A8  205F              575*        move.l  (sp)+,a0                ;Save return address
01AA  301F              576*        move.w  (sp)+,d0                ;Get system window number
01AC  2F08              577*        move.l  a0,-(sp)                ;Restore return address
01AE  E548              578*        lsl.w   #2,d0                   ;Get index to window pointer
01B0  2078 0180         579*        move.l  pSysCom.w,a0            ;Get pointer to SysCom
01B4  2068 0056         580*        move.l  SCwndtbl(a0),a0         ;Get pointer to window table
01B8  2F70 0000 0004    581*        move.l  0(a0,d0),4(sp)          ;Get window pointer
01BE  4E75              582*        rts                             ;Return
                        583*
                        584* ;
                        585* ; pOSdevNam - Get device name pointer
                        586* ;
                        587* ; FUNCTION pOSdevNam (untnbr: integer): pointer;
                        588* ;
01C0                    589* pOSdevNam
01C0  205F              590*        move.l  (sp)+,a0                ;Save return address
01C2  301F              591*        move.w  (sp)+,d0                ;Get unit number
```

```
01C4  C0FC  0020          592*          mulu    #UTlen,d0              ,Compute entry index
01C8  2F08                593*          move.l  a0,-(sp)               ;Restore return address
01CA  2078  0180          594*          move.l  pSysCom.w,a0           ,Get pointer to SysCom
01CE  2068  0014          595*          move.l  SCdevtab(a0),a0        ,Get pointer to device table
01D2  D1FC  0000  0002    596*          adda.l  #2,a0                  ,Get pointer to device ID
01D8  D1C0                597*          adda.l  d0,a0                  ;*
01DA  D1FC  0000  0008    598*          adda.l  #UTdid,a0              ;*
01E0  2F48  0004          599*          move.l  a0,4(sp)               ,Set function result
01E4  4E75                600*          rts                            ,Return
                          601*
                          602* ,
                          603* , pOSdate - Get system date pointer
                          604* ,
                          605* , FUNCTION pOSdate: pointer,
                          606* ,
01E6                      607* pOSdate
01E6  2F78  0180  0004    608*          move.l  pSysCom.w,4(sp)        ,Get pointer to SysCom
01EC  06AF  0000  0020    609*          addi.l  #SCtoday,4(sp)         ,Get pointer to system date
01F2  0004
01F4  4E75                610*          rts                            ,Return
                          611*
                          612* ;
                          613* , pOSsysVol - Get system volume name pointer
                          614* ,
                          615* , FUNCTION pOSsysVol  pointer,
                          616* ;
01F6                      617* pOSsysVol
01F6  2078  0180          618*          move.l  pSysCom.w,a0           ,Get pointer to SysCom
01FA  2F68  002E  0004    619*          move.l  SCbootnm(a0),4(sp)     ,Get system volume name pointer
0200  4E75                620*          rts                            ,Return
                          621*
                          622* ,
                          623* , pOScurVol - Get current volume name pointer
                          624* ,
                          625* , FUNCTION pOScurVol  pointer,
                          626* ,
0202                      627* pOScurVol
0202  2078  0180          628*          move.l  pSysCom.w,a0           ,Get pointer to SysCom
0206  2F68  0018  0004    629*          move.l  SCdirnam(a0),4(sp)     ,Get current volume name pointer
020C  4E75                630*          rts                            ,Return
                          631*
                          632* ;
                          633* , pOSsysVrs - Get OS version number string pointer
                          634* ,
                          635* ; FUNCTION pOSsysVrs: pointer;
                          636* ,
020E                      637* pOSsysVrs
020E  2078  0180          638*          move.l  pSysCom.w,a0           ,Get pointer to SysCom
0212  2F68  004E  0004    639*          move.l  SCvrsnbr(a0),4(sp)     ,Get OS version number pointer
0218  4E75                640*          rts                            ,Return
                          641*
                          642* ;
                          643* , pOSsysDat - Get OS version date string pointer
                          644* ,
```

```
                        645* ; FUNCTION pOSsysDat: pointer;
                        646* ;
021A                    647* pOSsysDat
021A  2078  0160        648*        move.l  pSysCom.w,a0            ;Get pointer to SysCom
021E  2F68  00S2  0004  649*        move.l  SCvrsdat(a0),4(sp)      ;Get OS version date pointer
0224  4E75              650*        rts                             ;Return
                        651*
```

```
                        653* ;
                        654* ; JSVECT - Jump to routine in system vector
                        655* ;
                        656* ; Parameters: D0.W - offset in system vector
                        657* ;
0226  2078  0180        658* JSVECT  MOVE.L  pSysCom.W,A0    ; (A0) = syscom
022A  2068  0008        659*        MOVE.L  SCjtable(A0),A0 ; (A0) = sysvect
022E  2070  0000        660*        MOVE.L  0(A0,D0.W),A0   ; (A0) = desired routine
0232  4ED0              661*        JMP     (A0)            ; Go to it!
                        662*
                        663* ;
                        664* ; JUVECT - Jump to routine in user vector
                        665* ;
                        666* ; Parameters: D0.W - offset in user vector
                        667* ;
0234  2078  0180        668* JUVECT  MOVE.L  pSysCom.W,A0    ; (A0) = syscom
0238  2068  001C        669*        MOVE.L  SCutable(A0),A0 ; (A0) = uservect
023C  2070  0000        670*        MOVE.L  0(A0,D0.W),A0   ; (A0) = desired routine
0240  4ED0              671*        JMP     (A0)            ; Go to it!
                        672*
                        673* ;
                        674* ; IGETDIR - Read a directory
                        675* ;
                        676* ; procedure xgetdir (fvid: vid; var fdir: directory; var DevBlocked: Boolean;
                        677* ;                     var fdevno: integer; var DevValid: Boolean); external;
                        678* ;
0242  7048              679* IGETDIR MOVEQ   #SVgetdir,D0
0244  60E0              680*        BRA.S   JSVECT
                        681*
                        682* ;
                        683* ; IPUTDIR - Write a directory
                        684* ;
                        685* ; procedure xputdir (var fdir: directory; fdevno: integer);
                        686* ;
0246  303C  0094        687* IPUTDIR MOVE.W  #SVputdir,D0
024A  60DA              688*        BRA.S   JSVECT
                        689*
                        690* ;
                        691* ; KeyPress - Test for any key
                        692* ;
                        693* ; function KeyPress: boolean;
                        694* ;
024C                    695* KeyPress
024C  205F              696*        move.l  (sp)+,a0        ;pop caller return address
024E  3F3C  0001        697*        move.w  #1,-(sp)        ;push function code
0252  4850              698*        pea     (a0)            ;push caller return address
0254  303C  000C        699*        move.w  #SVubusy,d0     ;set CCOS function offset
0258  60CC              700*        bra.s   JSVECT          ;do unit status
                        701*
                        702* ;
                        703* ; BrkPress - Test for break key
                        704* ;
                        705* ; function BrkPress: boolean;
                        706* ;
```

```
025A                    707* BrkPress
025A  4267              708*        clr.w   -(sp)           ;get keyboard driver unit number
025C  6100  FECE        709*        bsr     OSkybdDv        ;*
0260  301F              710*        move.w  (sp)+,d0        ;pop keyboard driver unit number
0262  205F              711*        move.l  (sp)+,a0        ;pop caller return address
0264  220F              712*        move.l  sp,d1           ;get pointer to result
0266  4850              713*        pea     (a0)            ;push caller return address
0268  3F00              714*        move.w  d0,-(sp)        ;push unit number
026A  2F01              715*        move.l  d1,-(sp)        ;push buffer address
026C  2F3C  0000  0001  716*        move.l  #1,-(sp)        ;push function code
0272  487A  0008+       717*        pea     bp1             ;push our return address
0276  303C  0044        718*        move.w  #SVustat,d0     ;set CCOS function offset
027A  60AA              719*        bra.s   JSVECT          ;do unit status
027C  205F              720* bp1    move.l  (sp)+,a0        ;pop caller return address
027E  3017              721*        move.w  (sp),d0         ;convert unit status to boolean
0280  E148              722*        lsl.w   #8,d0           ;*
0282  3E80              723*        move.w  d0,(sp)         ;*
0284  4ED0              724*        jmp     (a0)            ;return to caller
                        725*
```

```
                    727*        END

ACTIVE   00000006  IOEUIOPM 00000036  *OSVRTCRT  000040+  STALSRV  0000000A  UTBLK    0000001C
BP1        00027C+  IOEWNDBE 00000021  *POSCURKB  000190+  STBTSLT  00000000  UTDID    00000004
*BRKPRESS  00025A+  IOEWNDCS 00000022  *POSCURVO  000202+  STBTSRV  00000002  UTDRV    00000016
CLRSC    00000006  IOEWNDDC 00000023  *POSCURWN  00019C+  STINFO   0000000C  UTFLP    0000001B
CSATTR1  00000010  IOEWNDDS 00000024  *POSDATE    0001E4+  STINFOL  00000004  UTIODRV  00000002
CSATTR2  00000011  IOEWNDFN 00000020  *POSDEVNA   0001C0+  STNDRV   00000003  UTLEN    00000028
CSBPCH   00000006  IOEWNDIW 00000025  *POSSYSDA   00021A+  STNMBR   00000000  UTMTD    00000007
CSFRSTCH 00000008  IOEWNDWN 00000027  *POSSYSVO   0001F6+  STTYPE   00000001  UTRO     0000001A
CSLASTCH 0000000A  IOEWNDWR 00000026  *POSSYSVR   00020E+  SUSPEND  00000007  UTSIZ    00000010
CSLPCH   00000004  IOEWSZER 00000041  *POSSYSWN   0001A8+  SVBLKIO  0000002C  UTSLT    00000014
CSMASK   0000000C  IOOX     00000000  *POSUSERI   000180+  SVCLI    0000007C  UTSPT    00000018
CSTBLLOC 00000000  JSVECT     000226+  PSYSCOM  00000180  SVCLOSE  00000020  UTSRV    00000015
CURSON   00000002  JUVECT     000234+  SCBOOTDV 00000036  SVCRKPTH 00000060  UTTPS    00000019
EXCRTI     0000FE+ *KEYPRESS   00024C+  SCBOOTNM 0000002E  SVDELENT 00000090  UTTYP    00000017
GRAPHIC  00000001  MMBTBLK  0000001A  SCCODEJT 00000022  SVDSP    00000038  VERT     00000000
INSMOD   00000002  MMBTDEV  00000012  SCCURRK  00000048  SVDSP4   0000004C  VIDDEFLT 00000003
INVCURS  00000003  MMETDRV  00000018  SCCURRW  00000044  SVFLPDIR 00000088  VIDSET   00000007
INVRSE   00000000  MMBTSLT  00000014  SCDEVTAB 00000014  SVGET    00000014  VRTCRTI    000058+
IOEBSZER 00000040  MMBTSRV  00000016  SCDIRNAM 00000018  SVGETDIR 00000048  WRAPON   00000004
IOECLKMF 00000039  MMBTSW   00000010  SCFREEHP 00000004  SVGETVNM 00000080  WRATTR1  00000020
IOEFNCCD 00000038  MMHICOD  0000000C  SCIORSLT 00000000  SVINIT   00000018  WRATTR2  00000021
IOEINVDE 00000002  MMHIDTA  00000004  SCJTABLE 00000008  SVMARK   0000003C  WRATTR3  00000024
IOEIOREQ 00000003  MMLOCOD  00000008  SCMEMMAP 00000032  SVMAVAIL 00000044  WRBASEX  0000000E
IOEIRDSB 0000003C  MMLODTA  00000000  SCNUMPRO 00000028  SVNEW    00000034  WRBASEY  00000010
IOEIWDSB 0000003E  NOAUTOLF 00000004  SCNXTPRO 00000026  SVNEW4   00000068  WRBITOFS 0000001A
IOEKYBTE 00000035  NOSCROLL 00000005  SCPROCNO 00000002  SVOPEN   0000001C  WRCHARPT 00000008
IOEMFDRV 0000002D *OSACTSLT   000000+  SCPROTBL 0000002A  SVPUT    00000010  WRCURADR 00000008
IOENOBUF 00000017 *OSACTSRV   000010+  SCROOTW   00000040  SVPUTDIR 00000094  WRCURSX  00000016
IOENODSP 00000028 *OSALTSLT   000020+  SCSLTTBL 0000003C  SVRDCHAR 00000028  WRCURSY  00000018
IOENODTC 0000002E *OSALTSRV   000030+  SCSUSINH 0000005A  SVRLEASE 00000040  WRFILL1  00000025
IOENOKYB 00000029 *OSCURSP    0000C6+  SCSUSREQ 0000005C  SVSCHDIR 0000008C  WRFILL2  00000026
IOENOOMN 0000002B *OSDCM1DV   000164+  SCSYSIN   00000010  SVSEEK   00000030  WRFILL3  00000027
IOENOPRT 0000002C *OSDCM2DV   000156+  SCSYSOUT 0000000C  SVUBUSY   0000000C  WRFILL4  00000028
IOENOTIM 0000002A *OSDEVTYP   000082+  SCTODAY   00000020  SVUCLEAR 00000008  WRGRORGX 0000001C
IOENOTRN 00000015 *OSDISPDV   000120+  SCUSERID 0000004C  SVUINSTL 00000098  WRGRORGY 0000001E
IOEORDSB 0000003D *OSEXTCRT   0000D6+  SCUTABLE 0000001C  SVUREAD  00000004  WRHOMEOF 0000000C
IOEOWDSB 0000003F *OSKYBDDV   00012C+  SCVRSDAT 00000052  SVUSTAT  00000064  WRHOMEPT 00000004
IOEPADER 00000044 *OSMAXDEV   000112+  SCVRSNBR 0000004E  SVUWRITE 00000000  WRLENGTH 00000038
IOEPRMLN 00000037 *OSOMNIDV   000148+  SCWNDTBL 00000056  SVVALDIR 00000084  WRLNGTHI 00000012
IOERSZER 00000042 *OSPRTRDV   00010A+  SLTTYP8     00007E+  SVVRCHAR 00000024  WRLNGTHY 00000014
IOETBLFL 00000033 *OSSLTDV    000172+  SLTTYP9     000080+  SYSBYTES 00000186  WRRCDLEN 00000023
IOETBLID 00000032 *OSSLTTYP   00005A+  SSI         0000C4+  SYSKYBDF 00000184  WRSTATE  00000022
IOETBLIU 00000034 *OSSTRMDV   000102+  STACSLT  00000004  SYSWIN   00000005  WRWVSPTR 0000002C
IOETIMOT 00000016 *OSSYSSIZ   0000A6+  STACSRV  00000006  UNDSCR   00000001 *XGETDIR    000242+
IOEUARTE 00000043 *OSTIMDV    00013A+  STALSLT  00000008  UTBLF    00000006 *XPUTDIR    000246+

0 errors.  727 lines.  File CCLIB.OSI.TEXT
```