THE CORVUS CONCEPT

PASCAL LIBRARY USER GUIDE

Part No.: 7100-04978
Document No.: CCC/34-33/1.0
Release Date: September, 1983

Table of Contents

Chapter 3 -- Clock Control Unit (continued)

Chapter 4 -- Display Control Unit

Chapter 5 -- DataComm/Printer Control Unit

Chapter 10 -- Omninet Transporter Interface Unit (continued)

Chapter 11 -- Window Control Unit

Chapter 12 -- TurtleGraphics Unit

Chapter 13 -- Miscellaneous Functions and Procedures

---- C2LIB Units ----

---- C2LIB Units ----

General Information

The Corvus Concept System Library User Guide is a reference guide for Corvus Concept Library support. This guide is not a tutorial. Readers should be familiar with Pascal programming concepts.

This guide briefly describes the functions and procedures found in the Corvus Libraries. Some of the functions and procedures are meant to be used by advanced programmers. When this is the case, this guide refers to one of the Corvus technical reference manuals.

About CCLIB

The CCLIB.OBJ library file contains support units and subroutines for the Corvus Concept in a Pascal environment.

To use CCLIB, units must be declared in the USES section of the program. In a program, this section appears immediately after the program heading. In a unit, this section appears immediately after the interface heading.

The format of the uses section is as follows:

```
USES {$U /VolName/CCLIB} ccDEFN, OtherUnitNames;
```

For example, if CCLIB is in a volume named CCUTIL and the unit being used is ccCRTio, then the uses section would look like this:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccCRTio;
```

If another library or unit in a separate file is being used along with CCLIB, the volume and file name where this can be found must be specified:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccCRTio,
     {$U /KLLVOL/MUSIC} Sounds;
```

When using units in CCLIB, unit ccDEFN must be declared before other units, if it is needed. Other CCLIB units may be declared in any order. In the above example for instance, ccCRTio could not be declared before ccDEFN.

The order of units in library CCLIB is as follows:

```
ccDEFN          - Global definitions
ccHEXOUT        - Output hexadecimal numbers
ccLNGINT        - Long integer manipulations
ccCLKio         - Clock control unit
ccCRTio         - Display control unit
ccDCPio         - DataComm/Printer control unit
ccDIRio         - Directory control unit
ccGRFio         - Graphics display unit
ccLBLio         - Function key label unit
ccOMNio         - Omninet interface unit
ccOTCio         - Omninet Transporter interface unit
ccWNDio         - Window control unit
TurtleGraphics  - TurtleGraphics unit
```

About C2LIB

The C2LIB.OBJ library file contains units related to the Corvus disk controller. Included in this library is a unit to communicate directly with the Corvus disk controller, a unit to interface with the disk controller pipe commands, and a unit to interface with the disk controller semaphore commands.

When linking programs that use units from C2LIB, libraries must be specified in the order shown in this example:

```
LINKER - MC68000 Object Code Linker  n.n                dd-mmm-yy
(C) Copyright 1982 Silicon Valley Software, Inc.

Listing file - <return>              - no listing file
Output file  - pgmname               - executable program name
Input file [.OBJ] - pgmname          - output of Pascal compiler
Input file [.OBJ] - /CCUTIL/C2LIB    - Concept disk unit library
Input file [.OBJ] - /CCUTIL/CCLIB    - Concept Pascal library
Input file [.OBJ] - !PASLIB          - system Pascal library
Input file [.OBJ] - <return>         - end of input files
Linking segment '         '
  Initial memavail = nnnnnn
  Final memavail =  nnnnnn

The output is executable.
```


The Support Units

CCLIB contains three support units which are used by other units. Each unit is described below.

ccDEFN Unit

The Corvus Concept Global Definitions Unit defines system-wide constants and data structures.

The ccDEFN unit USES no other units. Several others units use ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN;
```

ccDEFN Unit Constants

Constants defined in ccDEFN are:

Corvus Concept I/O Result Codes

Identifier	Value	Description
IOok	00	Good result, no error
IOEinvdev	02	Invalid unit number/invalid device
IOEioreq	03	Invalid I/O request
IOEnebhrd	04	Nebulous hardware error
IOEoffln	05	Drive off line

(continued on next page)

Corvus Concept I/O Result Codes (continued)

Identifier	Value	Description
IOEwrprot	16	Device write protected
IOEseek	17	Seek error
IOEinvblk	18	Invalid block number
IOEnotrn	21	Transporter not ready
IOEtimot	22	Timed out waiting for Omninet event
IOEnobuf	23	Read without a valid write buffer
IOEflpto	24	Timeout error
IOEnoTO	25	Cannot restore to track 0
IOEnfmtd	26	Disk not formatted
IOEinvsct	27	Invalid sector length error
IOEwrngC	28	Read wrong track
IOEbdtrk	29	Track marked as bad (IBM spec)
IOEquereq	30	Queued request warning
IOEwndfn	32	Invalid window function
IOEwndbe	33	Window create boundary
IOEwndcs	34	Invalid character set
IOEwnddc	35	Delete current window
IOEwndds	36	Delete system window
IOEwndiw	37	Inactive window
IOEwndwr	38	Invalid window record
IOEwndwn	39	Invalid system window number

(continued on next page)

Corvus Concept I/O Result Codes (continued)

Identifier	Value	Description
IOEnodsp	40	Display driver not available
IOEnokyb	41	Keyboard driver not available
IOEnotim	42	Timer driver not available
IOEnoomn	43	OMNINET driver not available
IOEnoprt	44	Printer driver not available
IOEnfdrv	45	No floppy drive at slot
IOEnodtc	46	DataComm driver not available
IOEtblid	50	Invalid table entry ID
IOEtblfl	51	Table full
IOEtbliu	52	Table entry in use
IOEkybte	53	Keyboard transmission error
IOEuiopm	54	Invalid unit I/O parameter
IOEprmln	55	Invalid parameter block length
IOEfnccd	56	Invalid function code
IOEclkmf	57	Clock (hardware) malfunction
IOEirdsbl	60	Input to read buffer disabled
IOEordsbl	61	Output to read buffer disabled
IOEiwdsbl	62	Input to write buffer disabled
IOEowdsbl	63	Output to write buffer disabled
IOEbszerr	64	Buffer size error
IOEwszerr	65	Write size error
IOErszerr	66	Read size error

(continued on next page)

Corvus Concept I/O Result Codes (continued)

Identifier	Value	Description
IOEuarter	67	UART hardware error
IOEpaderr	68	Proportional spacing error
IOEdrvTO	70	Corvus drive time out
IOEbadcmd	71	Invalid Corvus disk command
IOEsvrdrv	72	Severe Corvus disk hardware problem
IOEtrnpdt	73	Error in Transporter command block

Miscellaneous

Identifier	Value	Description
MaxWindow	20	Maximum number of system windows
SysComPLoc	$0180	System common pointer location
LongStrMax	1030	Maximum "long" string length

ccDEFN Unit Types

Data types defined in ccDEFN are:

Data Type	Definition
Byte	-128..127;
Bytes	array [0..32766] of Byte;
Words	array [0..32766] of integer;
String32	string[32];
String64	string[64];
String80	string[80];
pByte	^Byte;
pBytes	^Bytes;
pWords	^Words;
pString32	^String32;
pString64	^String64;
pString80	^String80;

Data Type	Description		
SlotType	Device types for Concept I/O slots		
	NoDisk	0	No disk
	LocalDisk	1	Corvus local disk
	OmninetDisk	2	Corvus Omninet disk server
	FlpyC8Disk	3	Corvus 8" SSSD floppy disk
	FlpyC5Disk	4	...reserved
	FlpyA5Disk	5	Apple 5" floppy disk
	BankDisk	6	...reserved
	FlpyF8Disk	7	Corvus 8" DSDD floppy disk
	FlpyF5Disk	8	Corvus 5" DSDD floppy disk
	FlpyF3Disk	9	...reserved

ccDEFN Unit Variables

Variables defined in ccDEFN are:

None.

ccDEFN Unit Functions and Procedures

Procedures defined in ccDEFN are:

None.

Functions defined in ccDEFN are:

None.

ccHEXOUT Unit

The Output Hexadecimal Numbers Unit is used for displaying hexadecimal data. Normally, this unit is only needed during system development.

The ccHEXOUT unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccHEXOUT;
```

ccHEXOUT Unit Constants

Constants defined in ccHEXOUT are:

None.

ccHEXOUT Unit Types

Data types defined in ccHEXOUT are:

None.

ccHEXOUT Unit Variables

Variables defined in ccHEXOUT are:

None.

ccHEXOUT Unit Functions and Procedures

Procedures defined in ccHEXOUT are:

Procedure	Description
ccHEXinit	Unit initialization
PutHexByte	Output byte in hex
PutHexWord	Output integer in hex
PutHexLong	Output long integer in hex
DumpHex	Dump memory in hex bytes

Functions defined in ccHEXOUT are:

None.

ccHEXinit Procedure

ccHEXinit initializes the ccHEXOUT unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccHEXinit;
```

An example of this procedure is:

```
ccHEXinit;
```

PutHexByte Procedure ----------

PutHexByte writes to the current OUTPUT device (usually the display screen) the hexadecimal equivalent of the specified byte value. The definition of this procedure is:

```
PROCEDURE PutHexByte (Bvalue: byte);
```

Parameter	Data Type	Description
Bvalue	byte	Byte value to display

The procedure outputs 2 hexadecimal characters.

An example of this procedure is:

```
var b: byte;
....
b := 32;
PutHexByte (b); write ('  '); PutHexByte (b*2);
```

The output generated is "20 40".

PutHexWord Procedure

PutHexWord writes to the current OUTPUT device (usually the display screen) the hexadecimal equivalent of the specified integer value. The definition of this procedure is:

```
PROCEDURE PutHexWord (Wvalue: integer);
```

Parameter	Data Type	Description
Wvalue	integer	integer value to display

The procedure outputs 4 hexadecimal characters.

An example of this procedure is:

```
var i: integer;
....
i := 32;
PutHexWord (i); write ('  '); PutHexWord (i*2);
```

The output generated is "0020 0040".

PutHexLong Procedure ----------

PutHexLong writes to the current OUTPUT device (usually the display screen) the hexadecimal equivalent of the specified long integer value. The definition of this procedure is:

```
PROCEDURE PutHexLong (Lvalue: longint);
```

Parameter	Data Type	Description
Lvalue	long integer	Long integer value to display

The procedure outputs 8 hexadecimal characters.

An example of this procedure is:

```
var li: longint;
....
li := 32;
PutHexLong (li); write ('  '); PutHexLong (li*2);
```

The output generated is "00000020 00000040".

DumpHex Procedure

DumpHex writes to the current OUTPUT device (usually the display screen) a byte hex dump. The definition of this procedure is:

```
PROCEDURE DumpHex (BufPtr: pBytes; Len: integer);
```

Parameter	Data Type	Description
BufPtr	pBytes	Dump buffer pointer
Len	integer	Length of buffer to dump

The procedure outputs a byte hex dump of memory pointed to by BufPtr for Len bytes.

In the following program DumpHex is used to dump 200 bytes starting at location $700.

```
program hextst;
uses {$U /CCUTIL/CCLIB} ccDEFN, ccHEXOUT;
var p: pBytes;
begin
ccHEXinit;
p := pointer ($700);
DumpHex (p,200);
writeln; writeln;
end.
```

The output of this program is:

```
05 00 00 00  00 00 00 00  00 00 00 00  00 38 00 00
00 00 00 00  00 01 11 30  00 01 11 90  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 01 0D 4E  00 08 D5 5E  00 08 A4 90  00 00 00 00
00 00 02 CF  02 2F 00 7E  00 82 00 0E  00 00 02 2F
00 1C 00 24  00 60 00 00  00 00 00 00  00 00 00 00
00 00 03 00  00 02 00 00  00 00 00 00  00 00 00 00

00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00  00 00 00 00  00 00 00 00
00 00 00 00  00 00 00 00
```

ccLNGINT Unit

The Long integer Manipulations Unit is used to assemble and disassemble integers and long integers.

The ccLNGINT unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccLNGINT;
```

ccLNGINT Unit Constants

Constants defined in ccLNGINT are:

None.

ccLNGINT Unit Types

Data types defined in ccLNGINT are:

None.

ccLNGINT Unit Variables

Variables defined in ccLNGINT are:

None.

ccLNGINT Unit Functions and Procedures

Procedures defined in ccLNGINT are:

Procedure	Description
ByteLInt	Convert bytes to long integer
Byte2Int	Convert bytes to integer

Functions defined in ccLNGINT are:

Function	Description
LIntByte	Get byte value from long integer
Int2Byte	Get byte value from integer

ByteLInt Procedure

ByteLInt places four bytes into the specified long integer. The definition of this procedure is:

```
PROCEDURE ByteLInt (VAR Num: longint;
                    Byte0,Byte1,Byte2,Byte3: byte);
```

Parameter	Data Type	Description
Num	long integer	Long integer result
Byte0	byte	Byte 0 of long integer (MSB)
Byte1	byte	Byte 1 of long integer
Byte2	byte	Byte 2 of long integer
Byte3	byte	Byte 3 of long integer (LSB)

The procedure constructs a long integer from four bytes.

An example of this procedure is:

```
var li: longint; b0,b1,b2,b3: byte;
....
b0 := $12;
b1 := $34;
b2 := $56;
b3 := $78;
ByteLInt (li,b0,b1,b2,b3);
```

After this code is executed, li contains $12345678.

Byte2Int Procedure

Byte2Int places two bytes into the specified integer. The definition of this procedure is:

```
PROCEDURE Byte2Int (VAR Num: integer; Byte0,Byte1: byte);
```

Parameter	Data Type	Description
Num	integer	integer result
Byte0	byte	Byte 0 of integer (MSB)
Byte1	byte	Byte 1 of integer (LSB)

The procedure constructs an integer from two bytes.

An example of this procedure is:

```
var ii: integer; b0,b1: byte;
....
b0 := $AB;
b1 := $CD;
Byte2Int (ii,b0,b1);
```

After this code is executed, ii contains $ABCD.

LIntByte Function

LIntByte returns the specified byte of a long integer. The definition of this function is:

```
FUNCTION LIntByte (WhichByte: integer; Num: longint): byte;
```

Parameter	Data Type	Description
WhichByte	integer	Which byte to return (0..3)
Num	long integer	Long integer source for byte

The function returns byte WhichByte from long integer Num. WhichByte has a range of 0 (most significant byte) to 3 (least significant byte).

An example of this function is:

```
var li: longint; b0,b1,b2,b3: byte;
....
li := $12345678;
b0 := LIntByte (0,li);
b1 := LIntByte (1,li);
b2 := LIntByte (2,li);
b3 := LIntByte (3,li);
```

After this code is executed, b0 contains $12, b1 contains $34, b2 contains $56, and b3 contains $78.

Int2Byte Function

Int2Byte returns the specified byte of an integer. The definition of this function is:

```
FUNCTION Int2Byte (WhichByte,Num: integer): byte;
```

Parameter	Data Type	Description
WhichByte	integer	Which byte to return (0..1)
Num	long integer	integer source for byte

The function returns byte WhichByte from integer Num. WhichByte has a range of 0 (most significant byte) to 1 (least significant byte).

An example of this function is:

```
var ii: integer; b0,b1: byte;
....
ii := $ABCD;
b0 := Int2Byte (0,ii);
b1 := Int2Byte (1,ii);
```

After this code is executed, b0 contains $AB and b1 contains $CD.

The Clock Control Unit

ccCLKio

The Clock Control Unit is used to interface with the Corvus Concept system clock.

The ccCLKio unit USES no other units.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccCLKio;
```

ccCLKio Unit Constants

Constants defined in ccCLKio are:

None.

ccCLKio Unit Types

Data types defined in ccCLKio are:

```
+=============+=================================================+
| Data Type   | Description                                     |
+=============+=================================================+
| ClkStr40    | Clock unit string                               |
+---+---------+-------------------------------------------------+
    | string[40];                                               |
+---+---------+-------------------------------------------------+
| ClkPB       | Clock parameter block record                    |
+---+---------+-------------------------------------------------+
    | DayofWeek: integer;     { 1..7 for Sun..Sat     }         |
    | Month:     integer;     { 1..12                 }         |
    | Day:       integer;     { 1..31                 }         |
    | Hour:      integer;     { 0..23                 }         |
    | Mins:      integer;     { 0..59                 }         |
    | Secs:      integer;     { 0..59                 }         |
    | Tenths:    integer;     { 0..9                  }         |
    | LeapYear:  integer;     { 0..3 (0 = leap year) }         |
    | Year:      integer;     { 0..99                 }         |
    +-----------------------------------------------------------+
```

Data Type	Description
pClkDateRcd	Date record pointer

```
ClkDateRcd : Date record (packed)
    year:       0..100;     { year  }
    day:        0..31;      { day   }
    month:      0..12;      { month }
```

ccCLKio Unit Variables

Variables defined in ccCLKio are:

None.

ccCLKio Unit Functions and Procedures

Procedures defined in ccCLKio are:

Procedure	Description
ccCLKioInit	Unit initialization
ClkRead	Read clock parameters
ClkWrite	Write clock parameters
ClkWeekDay	Get day of week string
ClkDate1	Get day string ("dy-mon-yr")
ClkDate2	Get day string ("month dy, year")
ClkDate3	Get day string ("dy month year")
ClkTime1	Get time string ("hr:mi:sc")
ClkTime2	Get time string ("hr:mi am")
CvDateStr	Convert date string to date record

Functions defined in ccCLKio are:

None.

ccCLKioInit Procedure

ccCLKioInit initializes the ccCLKio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccCLKioInit;
```

An example of this procedure is:

```
ccCLKioInit;
```

ClkRead Procedure

ClkRead reads the system clock and places the current clock values in the specified clock parameter block. The definition of this procedure is:

```
PROCEDURE ClkRead (var CPB: ClkPB);
```

Parameter	Data Type	Description
CPB	ClkPB	Clock parameter block

The procedure updates the following values in the specified clock parameter block:

Field	Range
DayofWeek	1..7 for Sun..Sat
Month	1..12
Day	1..31
Hour	0..23
Mins	0..59
Secs	0..59
Tenths	0..9
LeapYear	0..3 (0 = leap year)
Year	0..99

An example of this procedure is:

```
var CPB: ClkPB;
....
ClkRead (CPB);        { get current clock values }
with CPB do writeln ('Current date and time is',
                     Year:4,Month:4,Day:4,
                     Hour:4,Mins:4,Secs:4,Tenths:4);
```

ClkWrite Procedure

ClkWrite updates the system clock with clock values in the specified clock parameter block. The definition of this procedure is:

```
PROCEDURE ClkWrite (CPB: ClkPB);
```

Parameter	Data Type	Description
CPB	ClkPB	Clock parameter block

The procedure updates the system clock using the following values in the specified clock parameter block:

Field	Range
DayofWeek	Computed by procedure
Month	1..12
Day	1..31
Hour	0..23
Mins	0..59
Secs	Set to 0 in procedure
Tenths	Set to 0 in procedure
LeapYear	Computed by procedure
Year	0..99

An example of this procedure is:

```
var CPB: ClkPB; newYear,newMonth,newDay: integer;
....
{ more code }           { get new date              }
....
ClkRead (CPB);          { get current clock values }
with CPB do begin
    Year  := newYear;   { set new year              }
    Month := newMonth;  { set new month             }
    Day   := newDay;    { set new day               }
    end;
ClkWrite (CPB);         { update clock values       }
```

ClkWeekDay Procedure

ClkWeekDay moves the current day of week to the specified string. The procedure reads the system clock before returning the date string. The definition of this procedure is:

```
PROCEDURE ClkWeekDay (var DateStr: ClkStr40);
```

Parameter	Data Type	Description
DateStr	ClkStr40	Day of week string

Day of the week is one of the following:

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

An example of this procedure is:

```
var DateString: ClkStr40;
....
ClkWeekDay (DateString);
writeln ('Today is ',DateString);
```

Output from this example is:

```
Today is Saturday
```


ClkDate1 Procedure

ClkDate1 moves the current system date in "dy-mon-yr" format to the specified string. The procedure reads the system clock before returning the date string. The definition of this procedure is:

```
PROCEDURE ClkDate1 (var DateStr: ClkStr40);
```

Parameter	Data Type	Description
DateStr	ClkStr40	Current system date string

The procedure constructs a string containing the current system date with a two digit day, the first three characters of the month, and a two digit year. Date components are separated with a hyphen.

An example of this procedure is:

```
var DateString: ClkStr40;
....
ClkDate1 (DateString);
writeln ('The date is ',DateString);
```

Output from this example is:

```
The date is 23-Oct-82
```

ClkDate2 Procedure

ClkDate2 moves the current system date in "month dy, year" format to the specified string. The procedure reads the system clock before returning the date string. The definition of this procedure is:

```
PROCEDURE ClkDate2 (var DateStr: ClkStr40);
```

Parameter	Data Type	Description
DateStr	ClkStr40	Current system date string

The procedure constructs a string containing the current system date with the full month name, a one or two digit day, and a four digit year.

An example of this procedure is:

```
var DateString: ClkStr40;
....
ClkDate2 (DateString);
writeln ('The date is ',DateString);
```

Output from this example is:

```
The date is October 23, 1982
```


ClkDate3 Procedure

ClkDate3 moves the current system date in "dy month year" format to the specified string. The procedure reads the system clock before returning the date string. The definition of this procedure is:

```
PROCEDURE ClkDate3 (var DateStr: ClkStr40);
```

Parameter	Data Type	Description
DateStr	ClkStr40	Current system date string

The procedure constructs a string containing the current system date with a one or two digit day, the full month name, and a four digit year. This is the common European form of the date.

An example of this procedure is:

```
var DateString: ClkStr40;
....
ClkDate3 (DateString);
writeln ('The date is ',DateString);
```

Output from this example is:

```
The date is 23 October 1982
```

ClkTime1 Procedure

ClkTime1 moves the current system time in "hr:mi:sc" format to the specified string. The procedure reads the system clock before returning the time string. The definition of this procedure is:

```
PROCEDURE ClkTime1 (var TimeStr: ClkStr40);
```

Parameter	Data Type	Description
TimeStr	ClkStr40	Current system time string

The procedure constructs a string containing the current system time with a two digit hour, a two digit minute, and a two digit second. Time components are separated with a colon. Hours are in the 24-hour format with a range of 0 to 23.

An example of this procedure is:

```
var TimeString: ClkStr40;
....
ClkTime1 (TimeString);
writeln ('The time is ',TimeString);
```

Output from this example is:

```
The time is 02:21:45
```


ClkTime2 Procedure

ClkTime2 moves the current system time in "hr:mi am" format to the specified string. The procedure reads the system clock before returning the time string. The definition of this procedure is:

```
PROCEDURE ClkTime2 (var TimeStr: ClkStr40);
```

Parameter	Data Type	Description
TimeStr	ClkStr40	Current system time string

The procedure constructs a string containing the current system time with a two digit hour, a two digit minute, and an am/pm indicator. Hours and minutes are separated with a colon. Hours are in the 12-hour format. If the time is between midnight and noon, the time has an am indicator, otherwise, the time has a pm indicator.

An example of this procedure is:

```
var TimeString: ClkStr40;
....
ClkTime2 (TimeString);
writeln ('The time is ',TimeString);
```

Output from this example is:

```
The time is 2:21 am
```

CvDateStr Procedure

CvDateStr converts a string to a packed date record. The definition of this procedure is:

```
PROCEDURE CvDateStr (DateStr: ClkStr40;
                     var Drcd: ClkDateRcd);
```

Parameter	Data Type	Description
DateStr	ClkStr40	Date string
Drcd	ClkDateRcd	Date record

The procedure evaluates the specifed string and computes a packed date record. The date string may be in any of the valid date formats:

dy-mon-yr

month dy, year

dy month year

If the date string is not evaluated successfully, a packed date record with all zeros is generated.

An example of this procedure is:

```
var CurrDate: ClkStr40; PackedDate: ClkDateRcd;
....
write ('Enter current date: ');
readln (CurrDate);
CvDateStr (CurrDate,PackedDate);
```


The Display Control Unit

ccCRTio

The Display Control Unit is used to interface with the display driver. The unit also contains several functions and procedures for user interaction with the system.

The ccCRTio unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccCRTio;
```

ccCRTio Unit Constants

Constants defined in ccCRTio are:

Identifier	Description
ccCRTioversion	Current unit version number string

ccCRTio Unit Types

Data types defined in ccCRTio are:

Data Type	Description		
CrtStatus	CrtAction function result		
	Normal	0	No error
	Escape	1	User pressed ESC key
	Error	2	Error in numeric conversion

Data Type	Description		
CrtRdx	Radix for number conversions		
	BinRdx	0	Binary radix (base 2)
	OctRdx	1	Octal radix (base 8)
	DecRdx	2	Decimal radix (base 10)
	HexRdx	3	Hexadecimal radix (base 16)
CrtCommand	Command codes for CrtAction		
	BsupOff	40	Do not suppress blanks in user input
	BsupOn	39	Suppress blanks in user input
	CursorBtab	9	Back tab
	CursorDown	5	Move cursor down
	CursorFtab	8	Forward tab
	CursorHome	3	Move cursor home
	CursorInvrse	13	Set inverse cursor (box cursor)
	CursorLeft	7	Move cursor left
	CursorOff	10	Do not display cursor
	CursorOn	11	Display cursor
	CursorRight	6	Move cursor right
	CursorUndscr	12	Set underline cursor

(continued on next page)

Data Type	Description		
CrtCommand	Command codes for CrtAction (continued)		
	CursorUp	4	Move cursor up
	DefNumOff	44	Do not output default num values
	DefNumOn	43	Output default numeric values
	DefStrOff	42	Do not output default strings
	DefStrOn	41	Output default strings
	DeleteChar	17	Delete character at cursor
	DeleteLine	15	Delete line at cursor
	EchoOff	34	Do not echo user input
	EchoOn	33	Echo user input
	EraseALL	2	Clear window and move cursor to upper left corner of window
	ErasEOL	1	Clear to end of line
	ErasEOS	0	Clear to end of window
	GrfMode	26	Set graphics mode
	HeartBeat	46	Output activity indicator
	InsertChar	16	Insert character at cursor
	InsertLine	14	Insert line at cursor
	InsertOff	18	Character insert mode OFF
	InsertOn	19	Character insert mode ON
	InvrtScreen	28	Invert screen video
	PagingOff	22	Paging mode OFF
	PagingOn	23	Paging mode ON

(continued on next page)

Data Type	Description		
CrtCommand	Command codes for CrtAction (continued)		
	ScrollOff	20	Scroll mode OFF
	ScrollOn	21	Scroll mode ON
	StartBeat	45	Initialize activity indicator
	TxtMode	27	Set text mode
	TypAhdOff	36	Type ahead not allowed
	TypAhdOn	35	Type ahead allowed
	UcaseOff	38	Do not convert input to upper case
	UcaseOn	37	Convert input to upper case
	VdoInv	30	Set inverse video
	VdoInvUnd	32	Set inverse underline video
	VdoNor	29	Set normal video
	VdoNorUnd	31	Set normal underline video
	WrapOff	24	Line wrap OFF
	WrapOn	25	Line wrap ON

ccCRTio Unit Variables

Variables defined in ccCRTio are:

Variable	Data Type	Description (default)
Beep	char	Bell character
CrtEcho	boolean	Echo input flag (TRUE)
CrtNdef	boolean	Output default number (TRUE)
CrtSdef	boolean	Output default string (FALSE)
CrtShft	boolean	Convert to uppercase (TRUE)
CrtBsup	boolean	Blank suppress (FALSE)
CrtTahd	boolean	Type ahead allowed (TRUE)
WndowLin	integer	Initial window size - lines
WndowCol	integer	Initial window size - columns
The following are used by the CrtTitle procedure		
CrtTpgm	string[16]	Program name string
CrtTvrs	string[16]	Program version number string
CrtTcpy	string[80]	Copyright notice string

ccCRTio Unit Functions and Procedures

Procedures defined in ccCRTio are:

Procedure	Description
ccCRTioInit	Unit initialization
CvIntStr	Convert integer to string
CvLIntStr	Convert long integer to string
CrtTitle	Clear window and display title banner at top of window
CrtPrompt	Get data from user with prompt
CrtPause	Wait for user response
GoToXY	Position cursor
CrtAction	Display command processing

Functions defined in ccCRTio are:

Function	Description
UpperCase	Convert character to upper case
GetByte	Get character from user
GetString	Get string from user
GetNum	Get numeric data (integer)
GetLongNum	Get numeric data (long integer)
CvStrInt	Convert string to integer
CvStrLInt	Convert string to long integer
BellTone	Generate speaker tones

ccCRTioInit Procedure

ccCRTioInit initializes the ccCRTio unit. This procedure must be called before any other procedures or functions in this unit are called. The definition of this procedure is:

```
PROCEDURE ccCRTioInit;
```

This procedure initializes the following variables:

Variable	Value	Description
CrtEcho	TRUE	Echo input flag
CrtNdef	TRUE	Output default number
CrtSdef	FALSE	Output default string
CrtShft	TRUE	Convert to uppercase
CrtBsup	FALSE	Blank suppress
CrtTahd	TRUE	Type ahead allowed
WndowLin	nn	Window size - lines
WndowCol	nn	Window size - columns
CrtTpgm	'pgmid'	Program name string
CrtTvrs	'0.0'	Program version number string
CrtTcpy		Copyright notice string

An example of this procedure is:

```
ccCRTioInit;
```

UpperCase Function

UpperCase converts a lower case character (a..z) to an upper case character. The definition of this function is:

```
FUNCTION UpperCase (Ch: char): char;
```

Parameter	Data Type	Description
Ch	char	Character to convert to upper case

An example of this function is:

```
UcChar := UpperCase ( AnyChar );
```

where the parameter AnyChar is a character. If the character is a lower case letter, UcChar is assigned the upper case equivalent of AnyChar. Otherwise, UcChar is assigned the value of AnyChar.

Another example is:

```
var i: integer; S: string[64];

S := 'This is an uppercase function test';
for i := 1 to length (S) do S[i] := UpperCase (S[i]);
```

This example converts all characters in string S to upper case characters.

GetNum Function

GetNum reads a number from INPUT and stores it in an integer variable. The definition of this function is:

```
FUNCTION GetNum (var Num: integer): CrtStatus;
```

Parameter	Data Type	Description
Num	integer	Integer number from INPUT

The function returns a result of Escape if the user presses the ESC key. Otherwise, the function result is Normal and the specified integer variable contains the input number. If the user presses RETURN with no other data, the default value is placed in the integer variable (see CrtNdef).

The first character entered, if not numeric, may specify the conversion radix. The conversion radix characters are:

```
      %  - input is an octal number (base 8)
+, -, #  - input is a decimal number (base 10)
   $, !  - input is a hexadecimal number (base 16)
```

Decimal is the default radix. Valid characters are 0 to 7 for octal radix, 0 to 9 for decimal radix, and 0 to 9 plus A to F for hexadecimal radix. Invalid characters, based on input radix, are not echoed and cause the bell to sound for user correction. If the numeric value overflows the maximum integer value, a truncated value is returned.

If CrtEcho is TRUE, input characters are echoed as input. If CrtEcho is FALSE, input characters are not echoed.

If CrtNdef is TRUE, the current value of Num is used as the default value. The current value of Num is displayed in decimal before accepting user input. The cursor is placed at the first character of the default value. If CrtNdef is FALSE, no default value is output before accepting input.

If CrtTahd is TRUE, data is accepted from the type ahead buffer until empty. Then data is accepted from the user. If CrtTahd is FALSE, the keyboard type ahead buffer is cleared before accepting user input.

An example of this function is:

```
if GetNum ( Int ) = Escape
    then { ESC key processing }
    else { normal processing };
```

If the user presses the ESC key, Int contains 0 and the ESC key processing section is executed. Otherwise, integer variable Int contains the input number and the normal processing section is executed.

GetLongNum Function

GetLongNum reads a number from INPUT and stores it in a long integer variable. The definition of this function is:

```
FUNCTION GetLongNum (var Num: LongInt): CrtStatus;
```

Parameter	Data Type	Description
Num	LongInt	Long integer from INPUT

The function returns a result of Escape if the user presses the ESC key. Otherwise, the function result is Normal and the specified long integer variable contains the input number. If the user presses RETURN with no other data, the default value is placed in the long integer variable (see CrtNdef).

The first character entered, if not numeric, may specify the conversion radix. The conversion radix characters are:

```
        %  - input is an octal number (base 8)
+, -, #  - input is a decimal number (base 10)
   $, !  - input is a hexadecimal number (base 16)
```

Decimal is the default radix. Valid characters are 0 to 7 for octal radix, 0 to 9 for decimal radix, and 0 to 9 plus A to F for hexadecimal radix. Invalid characters, based on input radix, are not echoed and cause the bell to sound for user correction. If the numeric value overflows the maximum long integer value, a truncated value is returned.

If CrtEcho is TRUE, input characters are echoed as input. If CrtEcho is FALSE, input characters are not echoed.

If CrtNdef is TRUE, the current value of Num is used as the default value. The current value of Num is displayed in decimal before accepting user input. The cursor is placed at the first character of the default value. If CrtNdef is FALSE, no default value is output before accepting input.

If CrtTahd is TRUE, data is accepted from the type ahead buffer until empty. Then data is accepted from the user. If CrtTahd is FALSE, the keyboard type ahead buffer is cleared before accepting user input.

An example of this function is:

```
if GetLongNum ( LgInt ) = Escape
    then { ESC key processing }
    else { normal processing };
```

If the user presses the ESC key, LgInt contains 0 and the ESC key processing section is executed. Otherwise, long integer variable LgInt contains the input number and the normal processing section is executed.

GetString Function

GetString reads an input string and stores it in a string variable with a maximum length of 80 characters. The definition of this function is:

```
FUNCTION GetString (var StrBuf: String80): CrtStatus;
```

Parameter	Data Type	Description
StrBuf	string80	String from INPUT

The function returns a result of Escape if the user presses the ESC key. Otherwise, the function result is Normal and the specified string variable contains the input string. If the user presses RETURN with no other data, the default string value is placed in the string variable (see CrtSdef). If more than 80 characters are entered, the bell is sounded and the character is not added to the string. When the string length is 80, only the backspace and RETURN keys are valid.

If CrtBsup is TRUE, all blank characters are removed from the input string. If CrtBsup is FALSE, blank characters are returned as entered.

If CrtEcho is TRUE, input characters are echoed as input. If CrtEcho is FALSE, input characters are not echoed.

If CrtSdef is TRUE, the current value of StrBuf is used as the default value. The current value of StrBuf is output before accepting user input. The cursor is placed at the first character of the default value. If CrtSdef is FALSE, no default string is output before accepting input.

If CrtShft is TRUE, all lower case characters are converted to upper case in the input string. If CrtShft is FALSE, lower case characters are returned as entered.

If CrtTahd is TRUE, data is accepted from the type ahead buffer until empty. Then data is accepted from the user. If CrtTahd is FALSE, the keyboard type ahead buffer is cleared before accepting user input.

An example of this function is:

```
if GetString ( UserReply ) = Escape
    then { ESC key processing }
    else { normal processing };
```

If the user presses the ESC key, the ESC key processing section is executed. Otherwise, string variable UserReply contains the input string and the normal processing section is executed.

GetByte Function

GetByte reads an input byte and returns a character. The definition of this function is:

```
FUNCTION GetByte: char;
```

The function returns the input character. If the RETURN key is pressed, a space is returned. If the ESC key is pressed, a ! is returned. If a lower case character is entered, a converted upper case character is returned.

If CrtEcho is TRUE, the input character is echoed at the current cursor position. If CrtEcho is FALSE, the character is not echoed.

If CrtTahd is TRUE, the character is accepted from the type ahead buffer if not empty. Otherwise, the character is accepted from the user. If CrtTahd is FALSE, the keyboard type ahead buffer is cleared before accepting the character.

An example of this function is:

```
Ch := GetByte;
```

Ch is the variable in which the character is stored.

CvStrInt Function

CvStrInt converts a numeric string, with a maximum length of 80 characters, into its integer equivalent. The definition of this function is:

```
FUNCTION CvStrInt (StrBuf: String80;
                   var Num: integer): CrtStatus;
```

Parameter	Data Type	Description
StrBuf	string80	Numeric string to convert
Num	integer	Integer value of string

The function returns a result of Error if the numeric string contains invalid characters. Otherwise, the function result is Normal and the specified integer variable contains the converted numeric string value.

The first character of the string, if not numeric, may specify the conversion radix. The conversion radix characters are:

```
        %  - input is an octal number (base 8)
+, -, #  - input is a decimal number (base 10)
   $, !  - input is a hexadecimal number (base 16)
```

Decimal is the default radix. Valid characters are 0 to 7 for octal radix, 0 to 9 for decimal radix, and 0 to 9 plus A to F for hexadecimal radix. Invalid characters, based on input radix, cause the function result to be set to Error. If the numeric value overflows the maximum integer value, a truncated value is returned.

An example of this function is:

```
Status := CvStrInt ( StringVar, IntVar );
```

Status is of the type CrtStatus. StringVar is the numeric string variable of type String80. IntVar is the integer variable in which the numeric value of StringVar is stored.

CvStrLInt Function

CvStrLInt converts a numeric string, with a maximum length of 80 characters, into its long integer equivalent. The definition of this function is:

```
FUNCTION CvStrLInt (StrBuf: String80;
                    var Num: LongInt): CrtStatus;
```

Parameter	Data Type	Description
StrBuf	string80	Numeric string to convert
Num	LongInt	Long integer value of string

The function returns a result of Error if the numeric string contains invalid characters. Otherwise, the function result is Normal and the specified long integer variable contains the converted numeric string value.

The first character of the string, if not numeric, may specify the conversion radix. The conversion radix characters are:

```
        %  - input is an octal number (base 8)
 +, -, #  - input is a decimal number (base 10)
    $, !  - input is a hexadecimal number (base 16)
```

Decimal is the default radix. Valid characters are 0 to 7 for octal radix, 0 to 9 for decimal radix, and 0 to 9 plus A to F for hexadecimal radix. Invalid characters, based on input radix, cause the function result to be set to Error. If the numeric value overflows the maximum long integer value, a truncated value is returned.

An example of this function is:

```
Status := CvStrLInt ( StringVar, LIntVar );
```

Status is of the type CrtStatus. StringVar is the numeric string variable of type String80. LIntVar is the long integer variable in which the numeric value of StringVar is stored.

CvIntStr Procedure

CvIntStr converts an integer into its numeric string equivalent. The definition of this procedure is:

```
PROCEDURE CvIntStr (Num: integer;
                    var StrBuf: String80; Rdx: CrtRdx);
```

Parameter	Data Type	Description
Num	integer	Integer value
StrBuf	string80	Numeric string equivalent
Rdx	CrtRdx	Conversion radix

The procedure converts the specified integer variable to a numeric string based on the conversion radix. The conversion radix may be BinRdx for binary, OctRdx for octal, DecRdx for decimal, or HexRdx for hexadecimal. An example of this procedure is:

```
CvIntStr ( IntVar, StringVar, DecRdx );
```

IntVar is the integer to be converted. StringVar is a string with a maximum length of 80 characters in which the converted number is stored. DecRdx is of type CrtRdx, which indicates the converted integer is to appear in decimal.

CvLIntStr Procedure

CvLIntStr converts a long integer into its numeric string equivalent. The definition of this procedure is:

```
PROCEDURE CvLIntStr (Num: LongInt;
                     var StrBuf: String80; Rdx: CrtRdx);
```

Parameter	Data Type	Description
Num	LongInt	Long integer value
StrBuf	string80	Numeric string equivalent
Rdx	CrtRdx	Conversion radix

The procedure converts the specified long integer variable to a numeric string based on the conversion radix. The conversion radix may be BinRdx for binary, OctRdx for octal, DecRdx for decimal, or HexRdx for hexadecimal. An example of this procedure is:

```
CvLIntStr ( LIntVar, StringVar, HexRdx );
```

LIntVar is the long integer to be converted. StringVar is a string with a maximum length of 80 characters in which the converted number is stored. HexRdx is of type CrtRdx, which indicates the converted long integer is to appear in hexadecimal.

CrtTitle Procedure

CrtTitle clears the current window and then writes a program title banner using inverse video on the first two lines of the window. The definition of this procedure is:

```
PROCEDURE CrtTitle (Txt: String80);
```

Parameter	Data Type	Description
Txt	string80	Title text

The procedure clears the current window and displays the program name (CrtTpgm), version number (CrtTvsn), and title text (txt) on the first line in the window. The copyright notice (CrtTcpy) is displayed on the next line. The default copyright notice is for Corvus Systems. The general form in the current window is:

```
CrtTpgm [CrtTvrs]: txt
CrtTcpy
```

An example of this procedure is:

```
CrtTpgm := 'TSTPGM'; CrtTvrs := '1.0';
CrtTcpy := 'Copyright 1982 KLL Inc.');
CrtTitle ('Test Program Title Text');
```

CrtPause Procedure

CrtPause waits for an input character. The definition of this procedure is:

```
PROCEDURE CrtPause (var Ch: char);
```

Parameter	Data Type	Description
Ch	char	Character input by user

The procedure selects the command window and outputs the message:

```
Press <space> to continue
```

The system waits until a character is input. The character is returned in the specified character variable. If the RETURN key is pressed, a space is returned. If the ESC key is pressed, a ! is returned. If a lower case character is entered, a converted upper case character is returned.

An example of this procedure is:

```
CrtPause (Ch);
```

Ch is the variable in which the user input is stored.

CrtPrompt Procedure

CrtPrompt writes a prompt line with optional prompt options. The definition of this procedure is:

```
PROCEDURE CrtPrompt (Txt,Opt: String80);
```

Parameter	Data Type	Description
Txt	string80	Prompt line text
Opt	string80	Prompt line options

The procedure outputs a prompt line at the current cursor position. Txt is the prompt line text and Opt is a list of valid replies. If the Txt parameter is null (string with no characters), the default prompt:

```
Please select option:
```

is used. If the opt parameter is null, no options are included in the prompt line. Otherwise, the specified options are included enclosed in brackets. An example of this procedure is:

```
CrtPrompt ('Enter your choice',
           'A(dd S(ubtract M(ultiply D(ivide');
case GetByte of
          'A': {add processing}
          'S': {subtract processing}
          'M': {multiply processing}
          'D': {divide processing}
    otherwise: {error processing
          end; {case GetByte of}
```

Both the prompt line and option line can be up to 80 characters each.

This example generates the following output:

```
Enter your choice [A(dd S(ubtract M(ultiply D(ivide]:
```

GoToXY Procedure

GoToXY positions the cursor to a given character position in the current window. The definition of this procedure is:

```
PROCEDURE GoToXY (X,Y: integer);
```

Parameter	Data Type	Description
X	integer	Character coordinates at which to place the cursor
Y	integer	

The procedure positions the cursor at the specified location. The origin (0,0) is the upper left hand corner of the current window. The X coordinate must be in the range of 0 to the number of characters per line less 1 in the current window. The Y coordinate must be in the range of 0 to the number of lines per window less 1 in the current window. If the X coordinate or Y coordinate is invalid, no cursor movement occurs.

An example of this procedure is:

```
var PosX,PosY: integer;
....
PosX = 0;
PosY = 10;
GoToXY ( PosX, PosY );
```

PosX is the X coordinate and PosY is the Y coordinate.

BellTone Function

BellTone is used to make varied sounds with the Concept speaker. The definition of this function is:

```
FUNCTION BellTone (Timbre: byte;
                   Duration,Period: integer): integer;
```

Parameter	Data Type	Description
Timbre	byte	Timbre of tone (0 to 127)
Duration	integer	Duration of tone in 50 milli-second increments
Period	integer	Time between speaker tones (Period equals 1/frequency)

The function returns the I/O result after producing the specified sound with the Concept speaker. Duration is the length of the tone in increments of 50 milliseconds, eg, 1 is a very short note and 20 is a very long note.

An example of this procedure is:

```
IOst := BellTone ( Timbre, Length, Period );
.. or ..
IOst := BellTone (31,10,254); { pitch 0 for .5 seconds }
```

Pitch Parameters for Three Octaves

Period	First Octave		Second Octave		Third Octave	
	Pitch	Timbre	Pitch	Timbre	Pitch	Timbre
254	0	31	12	51	24	81
240	1	31	13	51	25	81
226	2	31	14	51	26	81
213	3	31	15	51	27	81
201	4	31	16	51	28	81
190	5	31	17	51	29	81
179	6	31	18	51	30	81
169	7	31	19	51	31	81
159	8	31	20	51	32	81
150	9	31	21	51	33	81
142	10	31	22	51	34	81
134	11	31	23	51	35	81

CrtAction Procedure

CrtAction performs many different display control functions. The definition of this procedure is:

```
PROCEDURE CrtAction (Cmd: CrtCommand);
```

Parameter	Data Type	Description
Cmd	CrtCommand	Action command from table below

The procedure performs the specified command which is of the type CrtCommand.

An example of this procedure is:

```
CrtAction (EraseALL);
```

Each command is described below.

CrtAction (BsupOff) -- blank suppress user input OFF

BsupOff sets variable CrtBsup to FALSE. If CrtBsup is FALSE, blank characters entered by the user (when using the GetString function) are returned as entered.

CrtAction (BsupOn) -- blank suppress user input ON

BsupOn sets variable CrtBsup to TRUE. If CrtBsup is TRUE, blank characters entered by the user (when using the GetString function) are removed.

CrtAction (CursorBtab) -- back tab

CursorBtab moves the cursor to the next tab stop to the left of the cursor position. Tab stops occur every eighth character position starting at the left edge of the window. If the cursor is at the left edge of the window, it does not move.

CrtAction (CursorDown) -- move cursor down

CursorDown moves the cursor down one line. If the cursor is at the bottom edge of the current window, it is placed on the first line of the current window if wrap is on. If wrap is off, no cursor movement occurs. CursorDown is non-destructive to the data displayed on the screen.

CrtAction (CursorFtab) -- forward tab

CursorFtab moves the cursor to the next tab stop to the right of the cursor position. Tab stops occur every eighth character position starting at the left edge of the window. If the cursor is within eitht characters of the right edge of the window, it does not move.

CrtAction (CursorHome) -- move cursor home

CursorHome moves the cursor to the upper left corner of the current window. CursorHome is non-destructive to the data displayed on the screen.

CrtAction (CursorInvrse) -- set inverse cursor

CursorInvrse sets the cursor to the inverse character box. See CursorUndscr.

CrtAction (CursorLeft) -- move cursor left

CursorLeft moves the cursor left one character position. If the cursor is at the left edge of the current window, it is placed at the last character position of the previous line. CursorLeft is non-destructive to the data displayed on the screen.

CrtAction (CursorOff) -- display cursor OFF

CursorOff turns off the cursor in the current window.

CrtAction (CursorOn) -- display cursor ON

CursorOn turns on the cursor in the current window. A box cursor appears if CursorInvrse is in effect or an underline is CursorUndscr is in effect.

CrtAction (CursorRight) -- move cursor right

CursorRight moves the cursor right one character position. If the cursor is at the right edge of the current window, it is placed at the first character position of the next line. CursorRight is non-destructive to the data displayed on the screen.

CrtAction (CursorUndscr) -- set underline cursor

CursorUndscr sets the cursor to the underline character. See CursorInvrse.

CrtAction (CursorUp) -- move cursor up

CursorUp moves the cursor up one line. If the cursor is at the top edge of the current window, it does not move. CursorUp is non-destructive to the data displayed on the screen.

CrtAction (DefNumOff) -- output default numeric values OFF

DefNumOff sets variable CrtNdef to FALSE. If CrtNdef is FALSE, 0 is used as the default numeric value when using the GetNum and GetLongNum functions.

CrtAction (DefNumOn) -- output default numeric values ON

DefNumOn sets variable CrtNdef to TRUE. If CrtNdef is TRUE, the specified variable is used as the default numeric value when using the GetNum and GetLongNum functions.

CrtAction (DefStrOff) -- output default strings OFF

DefStrOff sets variable CrtSdef to FALSE. If CrtSdef is FALSE, a nil string (no characters) is used as the default string value when using the GetString function.

CrtAction (DefStrOn) -- output default strings ON

DefStrOn sets variable CrtSdef to TRUE. If CrtSdef is TRUE, the specified string variable is used as the default string value when using the GetString function.

CrtAction (DeleteChar) -- delete character at cursor

The character at the cursor position is deleted. The rest of the line to the right of the cursor is shifted left one character with a blank fill at the end of the line.

CrtAction (DeleteLine) -- delete line at cursor

DeleteLine deletes the line at the cursor position by moving all lines from the line below the cursor up one line. The last line is blank. The cursor does not move.

CrtAction (EchoOff) -- echo user input OFF

EchoOff sets variable CrtEcho to FALSE. If CrtEcho is FALSE, user input data is not echoed when using the Get.... functions.

CrtAction (EchoOn) -- echo user input ON

EchoOn sets variable CrtEcho to TRUE. If CrtEcho is TRUE, input data is echoed when using the Get.... functions.

CrtAction (EraseALL) -- clear screen and home

EraseALL clears all data from the current window and places the cursor at the upper left corner of the window.

CrtAction (ErasEOL) -- clear to end of line

ErasEOL clears all data from the cursor position to the end of the cursor line. The cursor is not moved.

CrtAction (ErasEOS) -- clear to end of screen

ErasEOS clears all data from the cursor position to the end of the current window. The cursor is not moved.

CrtAction (GrfMode) -- set graphics mode

GrfMode sets the current window to graphics mode. Graphics mode affect the window commands WinCreate and WinStatus. In graphics mode the parameters passed by these functions are intrepreted as pixel quantities instead of character cell quantities. See TxtMode.

CrtAction (HeartBeat) -- output activity indicator

HeartBeat outputs a period on the current line to indicate processing activity. If the current line fills, a carriage return is output and the next line on the display screen is filled. HeartBeat must be initialized by StartBeat.

CrtAction (InsertChar) -- insert character at cursor

InsertChar inserts a character at the cursor positions by moving all characters from the cursor position one character to the right. The character at the right edge of the window "falls off" and vanishes. The character at the cursor position is blank.

CrtAction (InsertOff) -- character insert mode OFF

InsertOff sets the current window character insert mode off. When insert mode is off, all characters displayed overwrite the existing text at the cursor position.

CrtAction (InsertOn) -- character insert mode ON

InsertOn sets the current window character insert mode on. When insert mode is on, the line is moved to the right to accomodate the new characters. Existing text is not overwritten.

CrtAction (InsertLine) -- insert line at cursor

InsertLine inserts a line at the cursor position by moving all lines from the cursor line to the bottom of the current window down one line. The last line is lost off the bottom of the window. The inserted line is blank with the cursor on the inserted line.

CrtAction (InvrtScreen) -- invert screen video

InvrtScreen inverts all data displayed in the current window. White on black becomes black on white or black on white becomes white on black. The background color definition is inverted and all subsequent characters, normal or inverse, are displayed relative to the new background color. InvrtScreen is non-destructive to the data displayed on the screen.

CrtAction (PagingOff) -- paging mode OFF

PagingOff sets the current window paging mode off. When paging mode is off, the window is not cleared when the cursor reaches the bottom of the window. See PagingOn.

CrtAction (PagingOn) -- paging mode ON

PagingOn sets the current window paging mode on. When paging mode is on and the cursor is moved past the bottom line of the window, the cursor disappears and the bell sounds. The user must press CNTL-Q to clear the screen and home the cursor.

CrtAction (ScrollOff) -- scroll mode OFF

ScrollOff prevents scrolling in the current window. When scroll mode is off, the display screen does not scroll when a line feed is output on the bottom line of the current window. Instead, the cursor moves to the upper left position in the current window.

CrtAction (ScrollOn) -- scroll mode ON

ScrollOn allows the current window to scroll. When scroll mode is on, the display screen data scrolls up one line when a line feed is output on the bottom line of the current window. The top line of the window falls off the top and the bottom line of the window is cleared.

CrtAction (StartBeat) -- initialize activity indicator

StartBeat outputs a carriage return and the initial period to indicate processing activity. HeartBeat is used to output additional periods as processing progresses.

CrtAction (TxtMode) -- set text mode

TxtMode sets the current window to text mode. Text mode affect the window commands WinCreate and WinStatus. In text mode the parameters passed in these functions are intrepreted as character cell quantities instead of pixel quantities. See GrfMode.

CrtAction (TypAhdOff) -- type ahead allowed OFF

TypAhdOff sets variable CrtTahd to FALSE. If CrtTahd is FALSE, the type-ahead buffer is cleared before accepting input from the user when using the Get.... functions.

CrtAction (TypAhdOn) -- type ahead allowed ON

TypAhdOn sets variable CrtTahd to TRUE. If CrtTahd is TRUE, the type-ahead buffer is used while accepting input from the user when using the Get.... functions. User input may be entered before being promped and is saved until requested by the program.

CrtAction (UcaseOff) -- convert user input to uppercase OFF

UcaseOff sets variable CrtShft to FALSE. If CrtShft is FALSE, lower case characters (a..z) entered by the user (when using the GetString function) are returned as entered.

CrtAction (UcaseOn) -- convert user input to uppercase ON

UcaseOn sets variable CrtShft to TRUE. If CrtShft is TRUE, lower case characters (a..z) entered by the user (when using the GetString function) are returned converted to upper case characters (A..Z). All other characters are returned as entered.

CrtAction (VdoInv) -- set inverse video

VdoInv sets the current window to output inverse video characters. All subsequent characters are displayed in inverse video until another video command is encountered. Inverse is relative to the current background color (black or white).

CrtAction (VdoInvUnd) -- set inverse underline video

VdoInvUnd sets the current window to output underlined inverse video characters. All subsequent characters are displayed underlined in inverse video until another video command is encountered. Inverse is relative to the current background color (black or white).

CrtAction (VdoNor) -- set normal video

VdoNor sets the current window to output normal video characters (not underlined, not inverse). All subsequent characters are displayed normally until another video command is encountered. Normal is relative to the current background color (black or white).

CrtAction (VdoNorUnd) -- set normal underline video

VdoNorUnd sets the current window to output underlined video characters. All subsequent characters are displayed underlined until another video command is encountered. Normal is relative to the current background color (black or white).

CrtAction (WrapOff) -- line wrap OFF

WrapOff sets the current window wrap mode off. While wrap mode is off, the cursor stops when it reaches the right or left edge of the window.

CrtAction (WrapOn) -- line wrap ON

WrapOn sets the current window wrap mode on. While wrap mode is on, the cursor is set to the left edge of the next line when the cursor moves past the right edge of the current window. Also, the cursor is set to the right edge of the previous line when the cursor moves past the left edge (backspace, cursor left) of the current window.

The Data Comm/Printer Interface Unit

ccDCPio

The Data Comm/Printer Interface Unit is used to set data communication parameters and protocols for the Corvus Concept data communications and printer drivers.

The ccDCPio unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccDCPio;
```

ccDCPio Unit Constants

Constants defined in ccDCPio are:

Baud Rate Codes

	Identifier	Value	Description
	Baud300	0	300 baud
	Baud600	1	600 baud
	Baud1200	2	1200 baud
	Baud2400	3	2400 baud
***	Baud4800	4	4800 baud
	Baud9600	5	9600 baud
	Baud19200	6	19200 baud
	*** = default value		

Parity Codes

	Identifier	Value	Description
***	ParDisabled	0	No parity
	ParOdd	1	Odd parity
	ParEven	2	Even parity
	ParMarkXNR	3	Transmit mark parity (receive parity expected, not checked)
	ParSpaceXNR	4	Transmit space parity (receive parity expected, not checked)

Parity is separate from word size
*** = default value

Data Comm Port Codes

	Identifier	Value	Description
	Port1	0	Data comm port 1
***	Port2	1	Data comm port 2

*** = default value

Character Size Codes

	Identifier	Value	Description
***	CharSz8	0	8 bit characters
	CharSz7	1	7 bit characters

Character size does not include parity
*** = default value

*** = default value

Protocol Codes

	Identifier	Value	Description
	LineCTSinverted	0	Clear to send - inverted
	LineCTSnormal	1	Clear to send - normal
	LineDSRinverted	2	Data set ready - inverted
***	LineDSRnormal	3	Data set ready - normal
	LineDCDinverted	4	Data carrier detect - inverted
	LineDCDnormal	5	Data carrier detect - normal
	XonXoff	6	X-on/X-off character protocol
	EnqAck	7	Enq/Ack character protocol
	EtxAck	8	Etx/Ack character protocol
	NoProtocol	9	No character protocol

Inverted is busy when 0, normal is busy when 1
*** = default value

Unit Number Codes

Identifier	Value	Description
PrinterUnit	0	Printer
DtaCom1Unit	1	DataComm 1
DtaCom2Unit	2	DataComm 2
DCPinvUnitNo	-1	Invalid unit number

DataComm Driver Unit Status Functions
(not used by unit ccDCPio)

Identifier	Value	Description
FCrdStatus	$07	Read buffer status
FCwrStatus	$08	Write buffer status
FCsetHIwater	$09	Set high water mark for read buffer
FCsetLOwater	$0A	Set low water mark for read buffer
FCrdOutDsbl	$0B	Toggle read buffer output disable - BUFFER TO USER
FCrdInDsbl	$0C	Toggle read buffer input disable - PORT TO BUFFER
FCwrOutDsbl	$0D	Toggle write buffer output disable - BUFFER TO PORT
FCwrInDsbl	$0E	Toggle write buffer input disable - USER TO BUFFER
FCwrBufChrs	$0F	Get number of characters in write buffer
FCrdBufChrs	$10	Get number of characters in read buffer
FCautoLF	$11	Toggle auto line feed flag
FCbtwnENQ	$12	Set the number of characters between ENQ's or ETX's
FCrdAltBuf	$13	Set an alternate read buffer
FCwrAltBuf	$14	Set an alternate write buffer

Printer Driver Unit Status Functions
(not used by unit ccDCPio)

Identifier	Value	Description
FCmodeChg	$80	Toggle transparent/translate mode
FCinstAlt	$81	Install AltChar translate table
FCattchPr	$82	Attach printer to unit
FCslctPitch	$83	Select pitch - 10 or 12
FCslctInch	$84	Select lines per inch - 6 or 8
FCinstAct	$85	Install printer action table
FCclpiStat	$86	Return state of CPI and LPI

ccDCPio Unit Types

Data types defined in ccDCPio are:

Data Type	Description
RdBufStatus	Data comm input buffer status block

```
    BufferSize:   integer;
    FreeSpace:    integer;
    HiWater:      integer;
    LowWater:     integer;
    InputDisbld:  boolean;
    OutputDsbld:  boolean;
    LostData:     boolean;
    AltBufAvail:  boolean;
    AltBufAddr:   pByte;
    AltBufSize:   integer;
```

WrBufStatus	Data comm output buffer status block

```
    BufferSize:   integer;
    FreeSpace:    integer;
    ChrBtwnENQ:   integer;
    InputDisbld:  boolean;
    OutputDsbld:  boolean;
    AutoLinFeed:  boolean;
    AltBufAvail:  boolean;
    AltBufAddr:   pByte;
    AltBufSize:   integer;
```

DCPstatusBlk	Printer status block

```
    CPI: integer;
    LPI: integer;
```

ccDCPio Unit Variables

Variables defined in ccDCPio are:

Variable	Data Type	Description
PrtAvail	boolean	Printer available (assigned)
DC1Avail	boolean	Datacom 1 available (assigned)
DC2Avail	boolean	Datacom 2 available (assigned)
PRT	integer	Unit number of /PRINTER
DC1	integer	Unit number of /DTACOM1
DC2	integer	Unit number of /DTACOM2

ccDCPio Unit Functions and Procedures

Procedures defined in ccDCPio are:

Procedure	Description
ccDCPioInit	Unit initialization

Functions defined in ccDCPio are:

Function	Description
DCPstatus	Get data comm driver status
DCPrdFree	Get input buffer free space
DCPwrFree	Get output buffer free space

(continued on next page)

Functions defined in ccDCPio (continued)

Function	Description
DCPbaudRate	Set data comm driver baud rate
DCPparity	Set data comm driver parity
DCPcharSize	Set data comm driver character size
DCPhandShake	Set data comm driver protocol
DCPgetUnitNo	Get current driver unit
DCPsetUnitNo	Select current driver unit
DCPrdStatus	Get input buffer status
DCPwrStatus	Get output buffer status
DCPautoLF	Toggle auto linefeed switch
PrtDataCom	Set printer driver data comm port
PrtTblStatus	Get printer status (CPI/LPI)

ccDCPioInit Procedure

ccDCPioInit initializes the ccDCPio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccDCPioInit;
```

An example of this procedure is:

```
ccDCPioInit;
```

Boolean variable PrtAvail is TRUE if the printer driver is loaded and assigned the device name /PRINTER. Otherwise, PrtAvail is FALSE.

Boolean variable DC1Avail is TRUE if the data comm driver is loaded and assigned the device name /DTACOM1. Otherwise, DC1Avail is FALSE.

Boolean variable DC2Avail is TRUE if the data comm driver is loaded and assigned the device name /DTACOM2. Otherwise, DC2Avail is FALSE.

DCPstatus Function

DCPstatus returns the current data comm driver parameters. The definition of this function is:

```
FUNCTION DCPstatus (var BaudRate,Parity,DataComm,
                        CharSize,Protocol: integer): integer;
```

Parameter	Data Type	Description
BaudRate	integer	Current baud rate code
Parity	integer	Current parity code
DataComm	integer	Current data comm port code
CharSize	integer	Current character size code
Protocol	integer	Current protocol code

The function returns the IORESULT from the data comm driver. The five parameter values are also returned in the specified integer variables. The parameter codes are defined in the ccDCPio constants section.

An example of this function is:

```
var IOst: integer;
    curBaud,curParity,curDtaCom,curChSize,curProto: integer;
....
IOst := DCPstatus (curBaud,curParity,
                   curDtaCom,curChSize,curProto);
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPrdFree Function

DCPrdFree returns the number of unused bytes in the data comm driver input buffer in the specified integer variable. The definition of this function is:

```
FUNCTION DCPrdFree (var FreeBytes: integer): integer;
```

Parameter	Data Type	Description
FreeBytes	integer	Space remaining in data comm driver input buffer

The function returns the IORESULT from the data comm driver. The integer variable FreeBytes contains the number of unused bytes in the data comm driver input buffer.

An example of this function is:

```
var IOst,spaceleft: integer;
....
IOst := DCPrdFree (spaceleft);
if IOst = 0
    then writeln ('DataComm input buffer space = ',spaceleft:1)
    else writeln ('DataComm error: ',iost:1);
```


DCPwrFree Function

DCPwrFree returns the number of unused bytes in the data comm driver output buffer in the specified integer variable. The definition of this function is:

```
FUNCTION DCPwrFree (var FreeBytes: integer): integer;
```

Parameter	Data Type	Description
FreeBytes	integer	Space remaining in data comm driver output buffer

The function returns the IORESULT from the data comm driver. The integer variable FreeBytes contains the number of unused bytes in the data comm driver output buffer.

An example of this function is:

```
var IOst,spaceleft: integer;
....
IOst := DCPwrFree (spaceleft);
if IOst = 0
    then writeln ('DataComm output buffer space = ',spaceleft:1)
    else writeln ('DataComm error: ',iost:1);
```

DCPbaudRate Function

DCPbaudRate sets the data comm driver baud rate. The definition of this function is:

```
FUNCTION DCPbaudRate (BaudRate: integer): integer;
```

Parameter	Data Type	Description
BaudRate	integer	Printer baud rate code

The function returns the IORESULT from the data comm driver. The data comm driver baud rate is set to the specified baud rate. The baud rate code is one of the following:

Code Identifier	Value	Description
Baud300	0	300 baud
Baud600	1	600 baud
Baud1200	2	1200 baud
Baud2400	3	2400 baud
Baud4800	4	4800 baud
Baud9600	5	9600 baud
Baud19200	6	19200 baud

An example of this function is:

```
var IOst,rate: integer;
....
rate := Baud9600;
IOst := DCPbaudRate (rate); { set baud rate to 9600 }
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPparity Function

DCPparity sets the data comm driver parity. The definition of this function is:

```
FUNCTION DCPparity (Parity: integer): integer;
```

Parameter	Data Type	Description
Parity	integer	Printer parity code

The function returns the IORESULT from the data comm driver. The data comm driver parity is set to the specified parity. The parity code is one of the following:

Code Identifier	Value	Description
ParDisabled	0	No parity
ParOdd	1	Odd parity
ParEven	2	Even parity
ParMarkXNR	3	Transmit mark parity (receive parity expected, not checked)
ParSpaceXNR	4	Transmit space parity (receive parity expected, not checked)
Parity is separate from word size		

Both ParMarkXNR and ParSpaceXNR expect a parity on receive for character framing. However, the parity check is disabled.

An example of this function is:

```
var IOst,parity: integer;
....
parity := ParOdd;
IOst := DCPparity (parity); { set parity to odd }
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPcharSize Function

DCPcharSize sets the data comm driver character size. The definition of this function is:

```
FUNCTION DCPcharSize (CharSize: integer): integer;
```

Parameter	Data Type	Description
CharSize	integer	Printer character size code

The function returns the IORESULT from the data comm driver. The data comm driver character size is set to the specified character size. The character size code is one of the following:

Code Identifier	Value	Description
CharSz8	0	8 bit characters
CharSz7	1	7 bit characters
Character size does not include parity		

Parity is generated by the UART separate from character size.

An example of this function is:

```
var IOst,charsize: integer;
....
charsize := CharSz8;
IOst := DCPcharSize (charsize); { set character size to 8 }
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPhandShake Function

DCPhandShake sets the data comm driver protocol. The definition of this function is:

```
FUNCTION DCPhandShake (Protocol: integer): integer;
```

Parameter	Data Type	Description
Protocol	integer	Printer protocol code

The function returns the IORESULT from the data comm driver. The data comm driver protocol is set to the specified protocol. The protocol code is one of the following:

Code Identifier	Value	Description
LineCTSinverted	0	Clear to send - inverted
LineCTSnormal	1	Clear to send - normal
LineDSRinverted	2	Data set ready - inverted
LineDSRnormal	3	Data set ready - normal
LineDCDinverted	4	Data carrier detect - inverted
LineDCDnormal	5	Data carrier detect - normal
XonXoff	6	X-on/X-off character protocol
EnqAck	7	Enq/Ack character protocol
EtxAck	8	Etx/Ack character protocol
NoProtocol	9	No character protocol
Inverted is busy when 0, normal is busy when 1		

Line protocols use hardware control lines from the printer. Character protocols only exist on printers with the ability to transmit data.

An example of this function is:

```
var IOst,protocol: integer;
....
protocol := LineCTSnormal;
IOst := DCPhandShake (protocol); { set protocol to CTS normal }
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPgetUnitNo Function

DCPgetUnitNo returns the current driver unit code. The definition of this function is:

```
FUNCTION DCPgetUnitNo: integer;
```

The function returns the current driver unit code. The current driver unit code is one of the following:

Identifier	Value	Description
PrinterUnit	0	Printer
DtaCom1Unit	1	DataComm 1
DtaCom2Unit	2	DataComm 2
DCPinvUnitNo	-1	Unit number not set

An example of this function is:

```
var CurUnit: integer;
....
CurUnit := DCPgetUnitNo;
case CurUnit of
     PrinterUnit: writeln ('Current unit is Printer');
     DtaCom1Unit: writeln ('Current unit is DataComm 1');
     DtaCom2Unit: writeln ('Current unit is DataComm 2');
    DCPinvUnitNo: writeln ('Unit number not set');
              end; {case}
```


DCPsetUnitNo Function

DCPsetUnitNo selects the current driver unit. The definition of this function is:

```
FUNCTION DCPsetUnitNo (UnitNo: integer): integer;
```

Parameter	Data Type	Description
UnitNo	integer	Driver unit code

The function returns the IORESULT from the data comm driver. The current driver unit code is one of the following:

Identifier	Value	Description
PrinterUnit	0	Printer
DtaCom1Unit	1	DataComm 1
DtaCom2Unit	2	DataComm 2

An example of this function is:

```
var IOst: integer;
....
IOst := DCPsetUnitNo (PrinterUnit);
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

DCPrdStatus Function

DCPrdStatus returns the data comm input buffer status information. The definition of this function is:

```
FUNCTION DCPrdStatus (var RDst: RdBufStatus): integer;
```

Parameter	Data Type	Description
RDst	RdBufStatus	Status record

The function returns the IORESULT from the data comm driver.

An example of this function is:

```
var IOst: integer; Rstatus: RdBufStatus;
....
IOst := DCPrdStatus (Rstatus);
with Rstatus do begin
    ....
    end;
```

DCPwrStatus Function

DCPwrStatus returns the data comm output buffer status information. The definition of this function is:

```
FUNCTION DCPwrStatus (var WRst: WrBufStatus): integer;
```

Parameter	Data Type	Description
WRst	WrBufStatus	Status record

The function returns the IORESULT from the data comm driver.

An example of this function is:

```
var IOst: integer; Wstatus: WrBufStatus;
....
IOst := DCPwrStatus (Wstatus);
with Wstatus do begin
    ....
    end;
```

DCPautoLF Function

DCPautoLF toggles the current driver unit auto linefeed switch. The definition of this function is:

```
FUNCTION DCPautoLF: integer;
```

The function returns the IORESULT from the data comm driver.

An example of this function is:

```
var IOst: integer;
....
IOst := DCPautoLF;
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

PrtDataCom Function

PrtDataCom set the printer driver data comm port. The definition of this function is:

```
FUNCTION PrtDataCom (Port: integer): integer;
```

Parameter	Data Type	Description
Port	integer	Printer data comm port code

The function returns the IORESULT from the printer driver. The printer driver data comm port is set to the specified port. The data comm port code is one of the following:

Code Identifier	Value	Description
Port1	0	Data comm port 1
Port2	1	Data comm port 2

An example of this function is:

```
var IOst,port: integer;
....
port := Port2;
IOst := PrtDataCom (port); { set data comm port to 2 }
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
```

PrtTblStatus Function

PrtTblStatus returns printer driver status information. The definition of this function is:

```
FUNCTION PrtTblStatus (var CPI,LPI: integer): integer;
```

Parameter	Data Type	Description
CPI	integer	Current printer chars/inch
LPI	integer	Current printer lines/inch

The function returns the IORESULT from the data comm driver. CPI is set to either 10 or 12. LPI is set to either 6 or 8.

An example of this function is:

```
var IOst,cpi,lpi: integer;
....
IOst := PrtTblStatus (cpi,lpi);
if IOst <> 0
    then writeln ('DataComm error: ',iost:1);
writeln ('Printer driver is set to ',cpi:1,
         ' chars/inch and ',lpi:1,' lines/inch');
```

The Volume Directory Unit

ccDIRio

The Volume Directory Unit is used to read and write volume directories. Directories may contain either MSB first integers (CCOS format) or LSB first integers (UCSD format). The unit converts the directory integers to true integers if needed.

****** NOTE ******

WRITING VOLUME DIRECTORIES IS NOT RECOMMENDED.

****

The ccDIRio unit USES no other units.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDIRio;
```

ccDIRio Unit Constants

Constants defined in ccDIRio are:

Identifier	Value	Description
BlockSize	512	Number of bytes in data block
VIDlength	7	Volume ID string length
TIDlength	15	File ID string length
MaxDirEnt	77	Maximum number of directory entries

ccDIRio Unit Types

Data types defined in ccDIRio are:

Data Type	Description		
DirRange	Directory record index range		
	0..MaxDirEnt;		
VID	Volume ID string		
	string[VIDlength];		
TID	File ID string		
	string[TIDlength];		
FileKind	File type		
	UntypedFile	0	Directory header
	XDskFile	1	...unused
	CodeFile	2	UCSD p-System code file
	TextFile	3	Text file
	InfoFile	4	...unused
	DataFile	5	Data file
	GrafFile	6	...unused
	FotoFile	7	...unused
	SecurDir	8	Directory header

Data Type	Description
DateRec	System date record

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

DirEntry	Volume directory record

```
packed record
firstblock: integer;
nextblock:  integer;
MarkBit:    boolean;
filler:     0..2047;
case fkind: FileKind of

  SECURDIR,UNTYPEDFILE:

  (dvid:        VID;          { Disk volume name       }
   deovblock:   integer;      { Last block of volume   }
   dnumfiles:   integer;      { Number of files        }
   dloadtime:   integer;      { Time of last access    }
   dlastboot:   DateRec;      { Most recent date       }
   MemFlipped:  boolean;      { TRUE if flpd in memory }
   DskFlipped:  boolean);     { TRUE if flpd on disk   }

  XDSKFILE,CODEFILE,TEXTFILE,INFOFILE,
  DATAFILE,GRAFFILE,FOTOFILE:

  (dtid:        TID;          { Title of file          }
   dlastbyte:   1..BlockSize; { Bytes in last block    }
   daccess:     DateRec);     { Last modification date }
```

Directory	Volume directory

```
array [DirRange] of DirEntry;
```


ccDIRio Unit Variables

Variables defined in ccDIRio are:

None.

ccDIRio Unit Functions and Procedures

Procedures defined in ccDIRio are:

Procedure/Subroutine		
Pascal	FORTRAN	Description
ccDIRioInit	...none	Unit initialization
GetVolDir	...none	Read volume directory
PutVolDir	...none	Write volume directory

Functions defined in ccDIRio are:

None.

ccDIRioInit Procedure

ccDIRioInit initializes the ccDIRio unit. This procedure may be called before any other procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccDIRioInit;
```

Currently, this procedure does nothing.

An example of this procedure is:

```
ccDIRioInit;
```

GetVolDir Procedure

GetVolDir reads the directory of the specified volume. The definition of this procedure is:

```
PROCEDURE GetVolDir (    VolID:       VID;
                      var VolDir:      directory;
                      var VolBlocked:  boolean;
                      var VolDevNo:    integer;
                      var VolValid:    boolean);
```

Parameter	Data Type	Description
VolID	VID	Volume name
VolDir	directory	Volume directory
VolBlocked	boolean	Volume blocked flag
VolDevNo	integer	Volume unit number
VolValid	boolean	Volume directory valid flag

This procedure reads the directory of the volume specified by VolID into the VolDir data area. VolBlocked is TRUE if the volume is blocked (a disk volume), FALSE otherwise. VolDevNo is the system unit number of the specified volume. VolValid is TRUE if the volume directory is valid, FALSE otherwise.

A simple program to display the file names contained in a specified volume illustrates the use of this procedure.

```
program dirlist;
uses {$u /ccutil/cclib} ccDIRio;
var d: directory;
    VolName: VID;
    BlkFlg,ValFlg: boolean;
    DevNbr,i: integer;
begin
ccDIRioInit;
writeln; write ('Enter volume name: ');
readln (VolName);
GetVolDir (VolName,d,BlkFlg,DevNbr,ValFlg);
if not ValFlg then begin
    writeln ('Invalid volume name specified ....');
    exit (dirlist);
    end;
writeln;
writeln ('Volume is mounted on system unit ',DevNbr:1);
if not BlkFlg then begin
    writeln ('Volume is not blocked ....');
    exit (dirlist);
    end;
writeln ('Volume ',d[0].dvid,' contains ',
                   d[0].dnumfiles:1,' files ');
write ('Directory type = ');
if d[0].DskFlipped
    then write ('UCSD ') else write ('CCOS ');
case d[0].fkind of
    untypedfile: writeln ('untyped');
       securdir: writeln ('secure directory');
            end;
for i := 1 to d[0].dnumfiles do begin
    writeln ('    ',d[i].dtid);
    end;
end.
```

EXIT

PutVolDir Procedure

PutVolDir writes a volume directory. Use of this procedure is NOT recommended. The definition of this procedure is:

```
PROCEDURE PutVolDir (var VolDir:   directory;
                         VolDevNo: integer);
```

Parameter	Data Type	Description
VolDir	directory	Volume directory
VolDevNo	integer	Volume unit number

This procedure writes the directory, VolDir, to the volume mounted on system unit VolDevNo.

An example of this procedure is:

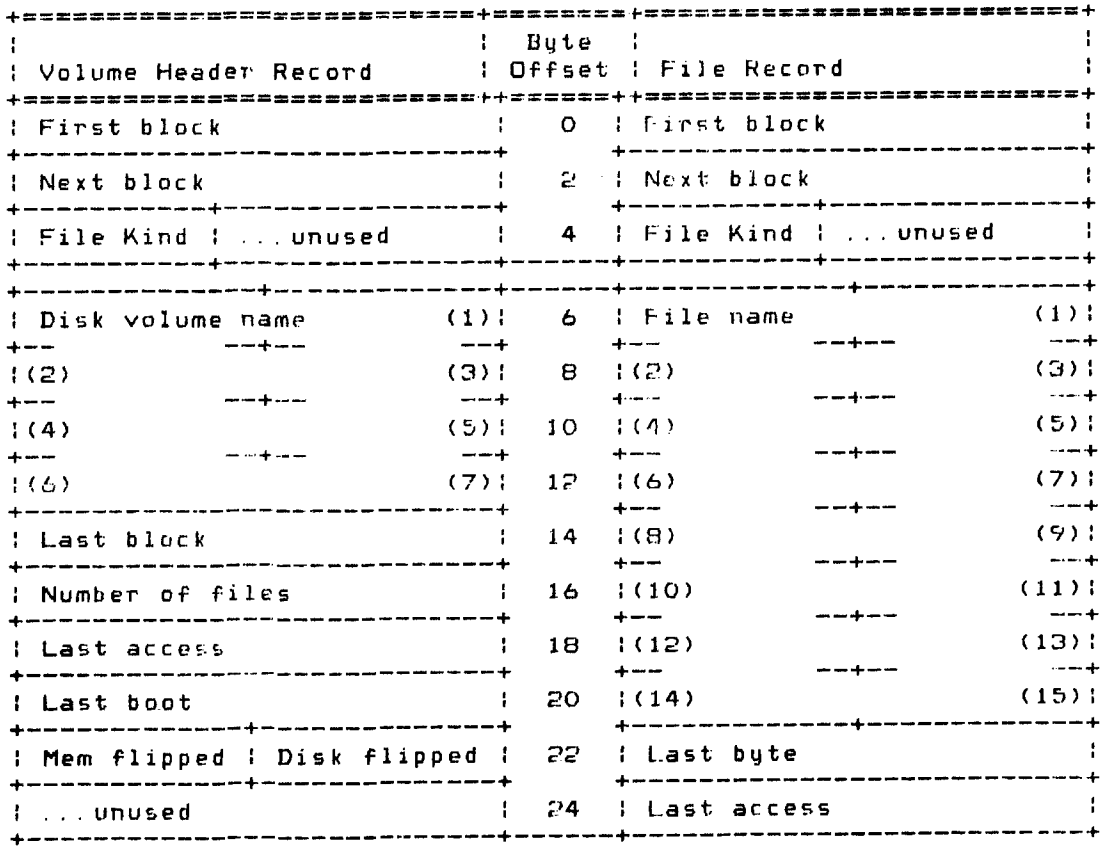
```
var NewUnit: integer;
    NewDir:  directory;
....
PutVolDir (NewDir,NewUnit);
```


Device Directory Description

Volumes reside on a blocked device, a disk. Each volume has a device (volume) directory which contains information about the volume and the files within that volume. A complete directory is an array of 78 directory entries, the first record being the header record which describes the specific volume. The other 77 entries are for the files.

The diagram below illustrates the layout of a single directory record. The upper section is common to all directory entries. In the lower section, the entries on the left side correspond to a volume header record and those on the right side correspond to a file record.

Directory Record Format



The elements in a directory record are:

FIRST BLOCK

is a word quantity which is the block number of the first block for the volume or file. This field is zero in the volume header record and the first block number of the file in file records.

NEXT BLOCK

is a word quantity which is the block number of the next available block for the volume or file. For the volume header record, this is the first block number after the volume directory which is normally block 6. For file records, this is the last block number of the file plus one.

FILE KIND

is a four-bit quantity which is the kind of file that the record describes. The values of file kind that are of interest are:

0 - a volume directory header,

2 - a UCSD p-System code file,

3 - a text file,

5 - a data file,

8 - is also a volume directory header.

The file kind field is followed by 12 bits of unused space to fill up the word. The following sections describe the different layouts of a directory record depending on the file kind field.

Directory Volume Header Record ----------

If the FILE KIND field in the directory record indicates that this record is a volume header record, the following fields are valid:

VOLUME NAME

is a 8-byte field containing the volume name. The first byte contains the length of the volume name. The remaining 7 bytes are the characters of the volume name, ie, a seven character string.

LAST BLOCK

is a word quantity which is the number of the last available block on this volume.

NUMBER OF FILES

is a word quantity which is the number of files on this volume.

LAST ACCESS

is a word quantity which is not used - it is set to zero.

LAST BOOT

is a word quantity which contains the most recent setting of the date. This word is in fact a date record.

MEMORY FLIPPED

is a boolean quantity used only by the system.

DISK FLIPPED

is a boolean quantity used only by the system.

There are two unused bytes at the end of the volume header record.

Directory File Record

If the FILE KIND field in the directory record indicates that this record is a file record, the following fields are valid:

FILE NAME

is a 16-byte field containing the file name. The first byte contains the length of the file name. The remaining 15 bytes are the characters of the file name, ie, a 15 character string.

LAST BYTE

is a word quantity which is the number of bytes in the last block of the file.

LAST MODIFICATION DATE

is a word quantity containing a date record representing the last time that this file was changed.

The Graphics Display Unit

ccGRFio

The Graphics Display Unit is used to interface with the graphics functions of the display driver.

The ccGRFio unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccGRFio;
```

ccGRFio Unit Constants

The following tables describe the Mode and Qualifier values for the SetOrigin, PlotPoint, and DrawLine procedures. Constants defined in ccGRFio are:

Mode Values for PlotPoint and DrawLine Procedures

Identifier	Value	Description
GrfMwhite	1	Draws white pixels
GrfMblack	0	Draws black pixels
GrfMflip	-1	Draws inverse of screen pixels

Qualifier Values for SetOrigin Procedure

Identifier	Value	Description
GrfQgrRel	1	Set origin relative to current graphics origin
GrfQgrAbs	2	Set origin to absolute graphics coordinates
GrfQchRel	3	Set origin relative to current character cursor position
GrfQchAbs	4	Set origin to character cursor position (x,y)

ccGRFio Unit Types

Data types defined in ccGRFio are:

None.

ccGRFio Unit Variables

Variables defined in ccGRFio are:

Variable	Data Type	Description
GrfPicBuf	pWords	Graphics picture buffer pointer
GrfPicPtr	pBytes	Graphics picture data pointer
GrfPicWd	integer	Graphics picture width (pixels)
GrfPicHt	integer	Graphics picture height (pixels)
GrfPicLn	longint	Graphics picture data size (bytes)

ccGRFio Unit Functions and Procedures

Procedures defined in ccGRFio are:

Procedure	Description
ccGRFioInit	Unit initialization
ccGRFioTerm	Unit termination
SetOrigin	Set graphics origin
PlotPoint	Plot display screen point
DrawLine	Draw display screen line
FillBox	Fill display screen area
CopyBox	Move display screen data
ReadBytes	Move data from display screen (one line)
WriteBytes	Move data to display screen (one line)
RelGrfPic	Release graphics picture buffer

Functions defined in ccGRFio are:

Procedure	Description
AloGrfPic	Allocate graphics picture buffer
RdDisp	Move data from display screen (rectangle)
WrDisp	Move data to display screen (rectangle)
DspToDsk	Move data from display screen to disk file
DskToDsp	Move data from disk file to display screen

ccGRFioInit Procedure

ccGRFioInit initializes the ccGRFio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccGRFioInit;
```

An example of this procedure is:

```
ccGRFioInit;
```

ccGRFioTerm Procedure

ccGRFioTerm terminates the ccGRFio unit. This procedure is called prior to program termination. The definition of this procedure is:

```
PROCEDURE ccGRFioTerm;
```

The procedure releases the graphics picture buffer if one is allocated. An example of this procedure is:

```
ccGRFioTerm;
```

SetOrigin Procedure

SetOrigin sets the graphics origin. The definition of this procedure is:

```
PROCEDURE SetOrigin (NewX,NewY,Qual: integer);
```

Parameter	Data Type	Description
NewX	integer	New graphics origin (graphics coordinates)
NewY	integer	
Qual	integer	Qualifier code from table below

The procedure sets the current graphics origin to the specified coordinates. Qual is one of the qualifier values described below:

Identifier	Value	Description
GrfQgrRel	1	Set origin relative to current graphics origin
GrfQgrAbs	2	Set origin to absolute graphics coordinates
GrfQchRel	3	Set origin relative to current character cursor position
GrfQchAbs	4	Set origin to character cursor position (x,y)

For a more detailed explanation of these values, please refer to the "Corvus Concept Operating System Manual."

An example of this procedure is:

```
SetOrigin (0,0,GrfQgrAbs); { set origin to graphics 0,0 }
```

PlotPoint Procedure

PlotPoint displays one point on the display screen. The definition of this procedure is:

```
PROCEDURE PlotPoint (Xcoord,Ycoord,Mode: integer);
```

Parameter	Data Type	Description
Xcoord	integer	Point at which to plot point (graphics coordinates)
Ycoord	integer	
Mode	integer	Mode code from table below

The procedure sets the indicated pixel based on Mode. Mode is one of the mode values described below:

Identifier	Value	Description
GrfMwhite	1	Draws white pixels
GrfMblack	0	Draws black pixels
GrfMflip	-1	Draws inverse of screen pixels

An example of this procedure is:

```
PlotPoint (100,100,GrfMwhite); { plot white point at 100,100 }
```

DrawLine Procedure

DrawLine draws a line on the display screen. The definition of this procedure is:

```
PROCEDURE DrawLine (StartX,StartY,
                    EndX,EndY,Mode: integer);
```

Parameter	Data Type	Description
StartX	integer	Starting point of line relative to graphics origin (graphics coordinates)
StartY	integer	
EndX	integer	Ending point of line relative to graphics origin (graphics coordinates)
EndY	integer	
Mode	integer	Mode code from table below

The procedure draws a line from the starting graphics coordinate to the ending graphics coordinate. Mode is one of the mode values described below:

Identifier	Value	Description
GrfMwhite	1	Draws white pixels
GrfMblack	0	Draws black pixels
GrfMflip	-1	Draws inverse of screen pixels

An example of this procedure is:

```
DrawLine (0,0,100,100,GrfMwhite); { draw white line          }
                                  { from coordinate 0,0      }
                                  { to coordinate 100,100    }
```


FillBox Procedure

FillBox fills a rectangular area on the display screen. The definition of this procedure is:

```
PROCEDURE FillBox (StartX,StartY,
                   Width,Height,Density: integer);
```

Parameter	Data Type	Description
StartX	integer	Lower left corner coordinate of the area being filled (graphics coordinates)
StartY	integer	
Width	integer	Pixel width of area
Height	integer	Pixel height of area
Density	integer	Pixel density of filled area

The procedure fills the specified rectangle with pixels of the specified density. A density of 1 completely fills the rectangle. A density of 2 displays every other pixel. A density of 3 displays every third pixel, and so forth.

An example of this procedure is:

```
FillBox (0,0,100,100,3);        { fill every third pixel in   }
                                { 100 x 100 pixel area        }
```

CopyBox Procedure

CopyBox copies a rectangular area from one area to another on the display screen. The definition of this procedure is:

```
PROCEDURE CopyBox (StartX,StartY,Width,Height,
                   NewX,NewY: integer);
```

Parameter	Data Type	Description
StartX	integer	Lower left corner coordinate of the area being copied from (graphics coordinates)
StartY	integer	
Width	integer	Pixel width of area
Height	integer	Pixel height of area
NewX	integer	Lower left corner coordinate of the area being copied to (graphics coordinates)
NewY	integer	

The procedure moves screen pixel data from one area to another on the display screen.

An example of this procedure is:

```
CopyBox (0,0,100,100,200,200); { move a 100 x 100 pixel area }
                               { from coordinates 0,0         }
                               { to coordinates 200,200       }
```


ReadBytes Procedure

ReadBytes reads a series of pixels from the display screen. The definition of this procedure is:

```
PROCEDURE ReadBytes (Count: integer; pBuff: pBytes);
```

Parameter	Data Type	Description
Count	integer	Number of bytes to move
pBuff	pBytes	Screen data buffer pointer

The procedure moves screen pixel data starting at the current graphics cursor position to the buffer pointed to by pBuff. Bytes are assembled from the pixels to the right (X direction) of the current graphics cursor position.

Each byte represents eight pixels. Data type pBytes is defined in the ccDEFN unit.

An example of the ReadBytes procedures is included in programming example for the WriteBytes procedure (next section).

WriteBytes Procedure

WriteBytes writes a series of pixels to the display screen. The definition of this procedure is:

```
PROCEDURE WriteBytes (Count: integer; pBuff: pBytes);
```

Parameter	Data Type	Description
Count	integer	Number of bytes to move
pBuff	pBytes	Screen data buffer pointer

The procedure moves pixel data from the buffer pointed to by pBuff to the display screen starting at the current graphics cursor position. Bytes are moved to the right (X direction) of the current graphics cursor position.

Each byte represents eight pixels. Data type pBytes is defined in the ccDEFN unit.

An example of the ReadBytes and WriteBytes procedures is illustrated in the following trivial program:

```
program RdWrScreen;

{ This program reads a small area of the display screen  }
{ into memory and then rewrites the screen with the data }
{ inverted (bottom lines at the top, etc.)               }

uses {$u /ccutil/cclib} ccDEFN, ccGRFio;

const maxlin = 100;  { define 100 lines          }
      maxcol = 40;   {    of 320 pixels (40 x 8) }

var ScrBuf:  array [0..maxlin,0..maxcol] of byte;
    pScrBuf: pBytes; curline: integer;

begin
ccGRFioInit;
for curline := maxlin downto 0 do begin
    SetOrigin (0,curline,GrfQgrAbs);           { set origin }
    pScrBuf := @ScrBuf[curline,0];     { get buffer pointer }
    ReadBytes (maxcol,pScrBuf);       { get data from screen }
    end;
for curline := maxlin downto 0 do begin
    SetOrigin (0,maxlin-curline,GrfQgrAbs);    { set origin }
    pScrBuf := @ScrBuf[curline,0];     { get buffer pointer }
    WriteBytes (maxcol,pScrBuf);      { get data from screen }
    end;
end.
```

AloGrfPic Function

AloGrfPic allocates on the heap a buffer to contain an image of the specified size.

Function AloGrfBuf is essentially a "MARK" using variable GrfPicBuf and a "NEW" with a variable allocation size. Function RelGrfPic is essentially a "RELEASE" using variable GrfPicBuf. Keep this in mind when doing other heap related operations while using display screen images.

The definition of this function is:

```
FUNCTION AloGrfPic (Width,Height,OvhdLen: integer): boolean;
```

Parameter	Data Type	Description
Width	integer	Pixel width of area
Height	integer	Pixel height of area
OvhdLen	integer	Buffer overhead length (bytes)

The function result is TRUE if the buffer is successfully allocated or FALSE otherwise. The following variables are set:

Variable	Data Type	Description
GrfPicBuf	pWords	Graphics picture buffer pointer
GrfPicPtr	pBytes	Graphics picture data pointer (GrfPicBuf + OvhdLen)
GrfPicWd	integer	Graphics picture width (pixels)
GrfPicHt	integer	Graphics picture height (pixels)
GrfPicLn	longint	Graphics picture data size (bytes)

The buffer is available until the RelGrfPic procedure is called. If a buffer is already allocated, procedure RelGrfPic is called before allocating a new buffer. The DspToDsk and DskToDsp functions use this function when allocating buffer space for display screen images.

An example of this procedure is:

```
var i,wd,ht: integer;
....
wd := 100; ht := 100;
if AloGrfPic (wd,ht,0)
    then begin
        i := RdDisp (GrfPicPtr,0,0,wd,ht);
        i := WrDisp (GrfPicPtr,2,2,wd,ht,1);
        RelGrfPic;
        end
    else .... { heap space not available }
```

RelGrfPic Procedure

RelGrfPic releases the display screen image pointed to by variable GrfPicBuf.

Function AloGrfBuf is essentially a "MARK" using variable GrfPicBuf and a "NEW" with a variable allocation size. Function RelGrfPic is essentially a "RELEASE" using variable GrfPicBuf. Keep this in mind when doing other heap related operations while using display screen images.

The definition of this procedure is:

```
PROCEDURE RelGrfPic;
```

The DspToDsk and DskToDsp functions use this procedure when deallocating buffer space for display screen images. An example of this procedure is:

```
var i,wd,ht: integer;
....
wd := 100; ht := 100;
if AloGrfPic (wd,ht,0)
    then begin
        i := RdDisp (GrfPicPtr,0,0,wd,ht);
        i := WrDisp (GrfPicPtr,2,2,wd,ht,1);
        RelGrfPic;
        end
    else .... { heap space not available }
```

RdDisp Function ----------

RdDisp moves data (pixels) from the display screen to the specified buffer. The definition of this function is:

```
FUNCTION RdDisp (DstBufPtr: pBytes;
                 Xcoord,Ycoord,
                 Width,Height: integer): integer;
```

Parameter	Data Type	Description
DstBufPtr	pBytes	Destination buffer pointer
Xcoord	integer	Lower left coordinate of area to move (graphics coordinates in current window)
Ycoord	integer	
Width	integer	Pixel width of area
Height	integer	Pixel height of area

The function returns the status of the move. Valid function results are:

Identifier	Value	Description
....	0	Successful operation
....	14	Specified area not entirely in current window

The destination buffer is assumed to be large enough to contain the pixel data in the specified area.

Xcoord and Ycoord are graphics coordinates relative to the current window graphics origin of the lower left corner of the area to be moved. The area specified must be entirely in the current window.

If the command is successful, the specified buffer contains pixel data from the display screen.

HINT: If the function result is 14, ensure the graphics origin of the current window is correct.

An example of this function is:

```
var i,wd,ht: integer;
....
wd := 100; ht := 100;
if AloGrfPic (wd,ht,0)
    then begin
        i := RdDisp (GrfPicPtr,0,0,wd,ht);
        if i = 0 then ....      { pixels moved successfully  }
                 else ....      { area not entirely in       }
                                {    current window          }
        i := WrDisp (GrfPicPtr,2,2,wd,ht,1);
        if i = 0 then ....      { pixels moved successfully  }
                 else ....      { area not entirely in       }
                                {    current window          }
        RelGrfPic;
        end
    else .... { heap space not available }
```

WrDisp Function

WrDisp moves data (pixels) from the specified buffer to the display screen. The definition of this function is:

```
FUNCTION WrDisp (SrcBufPtr: pBytes;
                 Xcoord,Ycoord,
                 Width,Height,Mode: integer): integer;
```

Parameter	Data Type	Description
SrcBufPtr	pBytes	Source buffer pointer
Xcoord	integer	Lower left coordinate of area to move (graphics coordinates in current window)
Ycoord	integer	
Width	integer	Pixel width of area
Height	integer	Pixel height of area
Mode	integer	Move mode -1 - XOR data 0 - OR data 1 - AND data

The function returns the status of the move. Valid function results are:

Identifier	Value	Description
....	0	Successful operation
....	14	Specified area not entirely in current window

If SrcBufPtr is NIL (by specifying POINTER(0)), the specified area is filled with every other pixel on. This may be used to generate a shaded background.

Xcoord and Ycoord are graphics coordinates relative to the current window graphics origin of the lower left corner of the area to be moved. The area specified must be entirely in the current window.

If the command is successful, the display screen contains pixel data from the specified buffer.

HINT: If the function result is 14, ensure the graphics origin of the current window is correct.

An example of this function is:

```
var i,wd,ht: integer;
....
wd := 100; ht := 100;
if AloGrfPic (wd,ht,0)
    then begin
        i := RdDisp (GrfPicPtr,0,0,wd,ht);
        if i = 0 then ....      { pixels moved successfully  }
                 else ....      { area not entirely in       }
                                {    current window          }
        i := WrDisp (GrfPicPtr,2,2,wd,ht,1);
        if i = 0 then ....      { pixels moved successfully  }
                 else ....      { area not entirely in       }
                                {    current window          }
        RelGrfPic;
        end
    else .... { heap space not available }
```

DspToDsk Function

DspToDsk moves display screen data (pixels) to the specified disk file. The definition of this function is:

```
FUNCTION DspToDsk (FileID: string80;
                   Xcoord,Ycoord,
                   Width,Height: integer): integer;
```

Parameter	Data Type	Description
FileID	string80	Destination file name
Xcoord	integer	Lower left coordinate of area to move (graphics coordinates in current window)
Ycoord	integer	
Width	integer	Pixel width of area
Height	integer	Pixel height of area

The function returns the status of the move which is the IOresult returned from the disk access. Valid function results are:

Identifier	Value	Description
....	0	Successful operation
....	7	Invalid file name
....	8	No room on volume
....	9	Volume not found
....	14	Specified area not entirely in current window
....	15	No buffer space available
....	16	Volume write protected

Xcoord and Ycoord are graphics coordinates relative to the current window graphics origin of the lower left corner of the area to be moved. The area specified must be entirely in the current window.

If the command is successful, the specified buffer contains pixel data from the display screen.

HINT: If the function result is 14, ensure the graphics origin of the current window is correct.

After writing display data to a disk file, the following variables are set until procedure RelGrfPic is called:

Variable	Data Type	Description
GrfPicBuf	pWords	Graphics picture buffer pointer
GrfPicPtr	pBytes	Graphics picture data pointer
GrfPicWd	integer	Graphics picture width (pixels)
GrfPicHt	integer	Graphics picture height (pixels)
GrfPicLn	longint	Graphics picture data size (bytes)

An example of this function is:

```
var i,wd,ht: integer;
....
wd := 100; ht := 100;
i := DspToDsk ('DISPLAY',0,0,wd,ht);
if i = 0 then ....      { pixels moved successfully  }
         else ....      { error processing           }
RelGrfPic;
```

DskToDsp Function

DskToDsp moves data (pixels) from the specified disk file to the display screen. The definition of this function is:

```
FUNCTION DskToDsp (FileID: string80;
                   Xcoord,Ycoord,
                   Mode: integer;
                   DispFlg: boolean): integer;
```

Parameter	Data Type	Description
FileID	string80	Destination file name
Xcoord	integer	Lower left coordinate of area to move (graphics coordinates in current window)
Ycoord	integer	
Mode	integer	Move mode -1 - XOR data 0 - OR data 1 - AND data
DispFlg	boolean	Display data if TRUE

The function returns the status of the move which is the IOresult returned from the disk access. Valid function results are:

Identifier	Value	Description
....	0	Successful operation
....	7	Invalid file name
....	9	Volume not found
....	10	File not found
....	14	Specified area not entirely in current window
....	15	No buffer space available

Xcoord and Ycoord are graphics coordinates relative to the current window graphics origin of the lower left corner of the area to be moved. The area specified must be entirely in the current window.

If DispFlg is TRUE, data from the specified file is moved to the display screen. If DispFlg is FALSE, data from the specified file is moved to the allocated buffer, but not moved to the display screen. DispFlg FALSE may be used when the size of the image is not known. After reading the image from disk, GrfPicWd and GrfPicHt may be used in positioning the image on the display screen with the WrDisp function.

If the command is successful, the display screen contains pixel data from the specified disk file.

HINT: If the function result is 14, ensure the graphics origin of the current window is correct.

After reading display data from a disk file, the following variables are set until procedure RelGrfPic is called:

Variable	Data Type	Description
GrfPicBuf	pWords	Graphics picture buffer pointer
GrfPicPtr	pBytes	Graphics picture data pointer
GrfPicWd	integer	Graphics picture width (pixels)
GrfPicHt	integer	Graphics picture height (pixels)
GrfPicLn	longint	Graphics picture data size (bytes)

An example of this function is:

```
var i: integer;
....
i := DskToDsp ('DISPLAY', 0, 0, TRUE);
if i = 0 then ....      { pixels moved successfully  }
         else ....      { error processing           }
RelGrfPic;
```

The Function Key Label Unit

ccLBLio

The Function Key Label Unit is used to manage the function key labels. Function key labels are displayed below the command window on the display screen. When the labels are displayed, pressing a function key generates a software-defined character sequence in the keyboard buffer. This character sequence is returned when characters are read from the label manager software in the CONSOLE/SYSTERM driver.

In general, function key labels are initialized by:

    LblsInit to initialize all label definitions
    LblSet to define label contents (one call for each label)
    LblsOn to display labels and return defined strings
    LblsOff to clear function key label display

The ccLBLio unit USES no other units.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccLBLio;
```

ccLBLio Unit Constants

Constants defined in ccLBLio are:

Identifier	Value	Description
LblKeyLen	8	Label key text string length
LblRtnLen	16	Label return text string length

ccLBLio Unit Types

Data types defined in ccLBLio are:

Data Type	Description
LblKeyStr	Label key text string
	string[LblKeyLen];
LblRtnStr	Label return text string
	string[LblRtnLen];

ccLBLio Unit Variables

Variables defined in ccLBLio are:

None.

ccLBLio Unit Functions and Procedures

Procedures defined in ccLBLio are:

Procedure	Description
ccLBLioInit	Unit initialization
ccLBLioTerm	Unit termination
LblsInit	Initialize labels to blank
LblsOn	Turn on function key labels
LblsOff	Turn off function key labels

Functions defined in ccLBLio are:

Function	Description
LblSet	Set label display/return strings

ccLBLioInit Procedure

ccLBLioInit initializes the ccLBLio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccLBLioInit;
```

This procedure initializes the function key labels to blanks and return strings to null. This procedure does not turn on the function key labels.

An example of this procedure is:

```
ccLBLioInit;
```

ccLBLioTerm Procedure

ccLBLioTerm terminates function key label processing. This procedure is called after all function key label processing is complete. The definition of this procedure is:

```
PROCEDURE ccLBLioTerm;
```

This procedure sets the function key labels to blanks and return strings to null. This procedure does not turn on the function key labels.

An example of this procedure is:

```
ccLBLioTerm;
```

LblsInit Procedure

LblsInit initializes all function key label information. The definition of this procedure is:

```
PROCEDURE LblsInit;
```

This procedure turns off the currently displayed function key labels, if any. All function key labels are set to blanks. All return strings are set to null. Function key labels are not displayed.

An example of this procedure is:

```
LblsInit;
```

LblSet Function

LblSet places text to be displayed in the function key label. It also defines the text to be returned when the function key is pressed. The definition of this function is:

```
FUNCTION LblSet (KeyNbr: integer;
                 LblStr: LblKeyStr;
                 RetStr: LblRtnStr): integer;
```

Parameter	Data Type	Description
KeyNbr	integer	Key number code from table
LblStr	LblKeyStr	Function key label text
RetStr	LblRtnStr	Function key return data

This function returns the I/O result code from the label manager in the SYSTERM driver. KeyNbr is the key number code for the label from the table below.

key	F1	F2	F3	F4	F5	F6	F7	F8	F9
CS	30	31	32	33	34	35	36	37	38	39
CU	20	21	22	23	24	25	26	27	28	29
S	10	11	12	13	14	15	16	17	18	19
U	00	01	02	03	04	05	06	07	08	09

CS - Command shifted
CU - Command unshifted
S - Shifted
U - Unshifted

LblStr is the string (up to 8 characters) to be displayed in function key label when the labels are turned on. RetStr is the string (up to 16 characters) to generate in the keyboard buffer when the corresponding function key is pressed.

An example of this function is:

```
Rslt := LblSet (9, 'Exit', 'Q');
```

Rslt is the I/O result code from the label manager in the SYSTERM driver. This example sets the function label marked F10 to display Exit. When function key F10 with no qualifiers is pressed after turning on labels, a Q is placed in the keyboar buffer.

LblsOn Procedure

LblsOn turns the function key labels on once they have been set (see LblSet function). The definition of this procedure is:

```
PROCEDURE LblsOn;
```

This procedure displays the current function key labels below the command window on the display screen. Return strings are placed in the keyboard buffer when the function keys are pressed.

An example of this procedure is:

```
LblsOn;
```

LblsOff Procedure

LblsOff turns the function key labels off. The definition of this procedure is:

```
PROCEDURE LblsOff;
```

This procedure clears the display of current function key labels below the command window on the display screen. Afterwards, pressing function keys have no effect in that no return strings are placed in the keyboard buffer when the function key labels are not displayed. The function key definitions are retained. LblsOn may be used to redisplay the currently defined function key labels.

An example of this procedure is:

```
LblsOff;
```

The Omninet Interface Unit

ccOMNio

The Omninet Interface Unit is used to interface with the Corvus Omninet local area network.

This document does not define the various Omninet operations, but details the use of the unit procedures available for interacting with the Omninet network. See the "Omninet Programmer's Guide" for a detailed description of the Omninet operations.

The ccOMNio unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccOMNio;
```

ccOMNio Unit Constants

Constants defined in ccOMNio are:

Transporter Return Codes (OCresult)

Identifier	Value	Description
OkCode	0	Successful operation
GaveUp	128	Aborted a send command after MaxRetries tries
TooLong	129	Last message sent was too long for the receiver
NoSockt	130	Sent to an uninitialized socket

(continued on next page)

Transporter Return Codes (OCresult)
(continued)

Identifier	Value	Description
HdrErr	131	Sender's header length did not match receiver's header length
BadSock	132	Invalid socket number
Inuse	133	Tried to set up a receive on an active socket
BadDest	134	Sent to an invalid host number
Echoed	192	Echo command was successful
CmdAcpt	254	Command accepted
NoTrans	255	Unable to communicate with Transporter

Transporter OPcodes

Identifier	Value	Description
RecvOp	$F0	SETUPRECV opcode
SendOp	$40	SENDMSG opcode
InitOp	$20	INIT opcode
EndOp	$10	ENDRECV opcode
DebOp	$08	PEEK/POKE opcode
EchoOp	$02	ECHOCMD opcode
WhoOp	$01	WHOAMI opcode

ccOMNio Unit Types

Data types defined in ccOMNio are:

Data Type	Description
pOCrsltRcd	Result record pointer
OCrsltRcd	Result record

```
RCode:  byte;     { command result code  }
Sourc:  byte;     { source host number   }
Len:    integer;  { received data length }
UCdta:  array [0..255] of byte;
                  { user control data    }
```

ccOMNio Unit Variables

Variables defined in ccOMNio are:

Variable	Data Type	Description
OCresult	integer	similar to UCSD Pascal IORESULT, may be checked after each command
OCrslt	OCrsltRcd	Result record which is used for all commands except OCsndMesg and OCsetRCcv
OCcurBP	pBytes	Current buffer pointer
OCcurRP	pOCrsltRcd	Current result pointer

ccOMNio Unit Functions and Procedures

Procedures defined in ccOMNio are:

Procedure	Description
ccOMNioInit	Unit initialization
OCsndMesg	Send message
OCsetRecv	Set up receive
OCendRecv	End receive
OCinitTrans	Initialize Transporter
OCwhoAmI	Get Transporter number
OCechoTrans	Echo to specified Transporter
OCpokeTrans	Write to Transporter memory

Functions defined in ccOMNio are:

Function	Description
OCpeekTrans	Read from Transporter memory

ccOMNioInit Procedure

ccOMNioInit initializes the ccOMNio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccOMNioInit;
```

An example of this procedure is:

```
ccOMNioInit;
```

OCsndMesg Procedure

OCsndMesg sends a message to the specified host and socket. The definition of this procedure is:

```
PROCEDURE OCsndMesg (pMesgBuf: pBytes; pRsltBuf: pOCrsltRcd;
                     ScktNbr,DtaLen,HdrLen,DstHost: integer);
```

Parameter	Data Type	Description
pMesgBuf	pBytes	Send data buffer
pRsltRcd	pOCrsltRcd	Result record buffer
ScktNbr	integer	Destination socket number in (1 to 4 sockets $80..$80)
DtaLen	integer	Send data buffer length
HdrLen	integer	User control data length
DstHost	integer	Destination host number

The procedure attempts to send a message to the destination host. After executing the procedure the Rcode field of the specified result buffer contains a one byte signed equivalent of the command result. Valid command results are:

Identifier	Value	Description
	0..127	Successful operation (retries) Rcode = 0..127
GaveUp	128	Aborted a send command after MaxRetries tries Rcode = -128
TooLong	129	Last message sent was too long for the receiver Rcode = -127
NoSockt	130	Sent to an uninitialized socket Rcode = -126

(continued on next page)

Identifier	Value	Description	
HdrErr	131	Sender's header length did not match receiver's header length	Rcode = -125
BadSock	132	Invalid socket number	Rcode = -124
BadDest	134	Sent to an invalid host number	Rcode = -122

An example of this procedure is:

```
var Sbuff: array [1..512] of byte;      { send message buffer }
    Srslt: OCrsltRcd;                   { send result buffer  }
....
OCsndMesg (@Sbuff,@Srslt,1,512,0,63);
                                        { send to host 63, socket 1  }
if Rrslt.Rcode >= 0 then ....           { mesg send successfully     }
                    else ....           { error processing           }
```

OCsetRecv Procedure

OCsetRecv prepares the specified socket to receive a single message. The definition of this procedure is:

```
PROCEDURE OCsetRecv (pMesgBuf: pBytes; pRsltBuf: pOCrsltRcd;
                     ScktNbr,DtaLen,HdrLen: integer);
```

Parameter	Data Type	Description
pMesgBuf	pBytes	Receive data buffer
pRsltRcd	pOCrsltRcd	Result record buffer
ScktNbr	integer	Receive socket number in the range of 1 to 4 (1..4 = sockets $80..$B0)
DtaLen	integer	~~Send~~ Recv data buffer length
HdrLen	integer	User control data length

The procedure activates the socket to receive a message. After executing the procedure the Rcode field of the specified result buffer contains a one byte signed equivalent of the command result. Valid command results are:

Identifier	Value	Description
....	0	Successful operation Rcode = 0
BadSock	132	Invalid socket number Rcode = -124
Inuse	133	Tried to set up a receive on an active socket Rcode = -123
CmdAcpt	254	Command accepted Rcode = -2

If the command is accepted successfully, the Rcode field retains the value CmdAcpt until a message is received for the socket.

An example of this procedure is:

```
var Rbuff: array [1..512] of byte;      { recv message buffer }
    Rrslt: OCrsltRcd;                   { recv result buffer  }
....
OCsetRecv (@Rbuff,@Rrslt,1,512,0);
                                        { set receive on socket 1     }
while Rrslt.Rcode = CmdAcpt do;         { wait until mesg received    }
if Rrslt.Rcode = 0 then ....            { mesg received successfully }
                   else ....            { error processing            }
```

OCendRecv Procedure

OCendRecv releases the specified socket number. The definition of this procedure is:

```
PROCEDURE OCendRecv (ScktNbr: integer);
```

Parameter	Data Type	Description
ScktNbr	integer	Receive socket number in the range of 1 to 4 (1..4 = sockets $80..$B0)

The procedure disables reception of any more messages for the socket until another OCsetRecv command is issued. After executing the procedure OCresult contains the command result. Valid command results are:

Identifier	Value	Description
OkCode	0	Successful operation
BadSock	132	Invalid socket number specified

An example of this procedure is:

```
OCendRecv (1);              { end receiving on socket 1 ($80) }
if OCresult <> 0 then ...   { command error                   }
```

OCinitTrans Procedure

OCinitTrans initializes the Transporter as in a hardware reset or a power-up. The definition of this procedure is:

```
PROCEDURE OCinitTrans;
```

The procedure sets all parameters to their default values. Event counters are set to zero. After executing the procedure OCresult contains the Transporter number of the host computer.

An example of this procedure is:

```
var TransNbr: integer;
....
OCinitTrans;
TransNbr := OCresult  { save Transporter number }
```

OCwhoAmI Procedure

OCwhoAmI returns the Transporter number of the host computer. The definition of this procedure is:

```
PROCEDURE OCwhoAmI;
```

After executing the procedure OCresult contains the Transporter number of the host computer.

An example of this procedure is:

```
OCwhoAmI;
writeln ('The host Transporter number is ',OCresult:2);
```

OCechoTrans Procedure

OCechoTrans requests the Transporter to send an echo packet to the specified host. The echo packet is used to verify the presence of another network device without disturbing that device. The definition of this procedure is:

```
PROCEDURE OCechoTrans (DstHost: integer);
```

Parameter	Data Type	Description
DstHost	integer	Destination host number

The procedure sends an echo packet to Transporter DstHost. Transporter DstHost receives the packet and acknowledges without informing the attached host computer. After executing the procedure OCresult contains the command result. Valid command results are:

Identifier	Value	Description
GaveUp	128	Aborted a send command after MaxRetries tries
BadDest	134	Sent to an invalid host number
Echoed	192	Echo command was successful

An example of this procedure is:

```
OCechoTrans (1);                { is Transporter 1 active?    }
if OCresult = Echoed then ...   { Transporter responded       }
                  else ...      { Transporter did not respond }
```

OCpeekTrans Function

OCpeekTrans is used to examine internal memory of the Transporter. See the "Omninet Programmer's Guide" for more information on the Transporter peek command. The definition of this function is:

```
FUNCTION OCpeekTrans (Addr: integer): byte;
```

Parameter	Data Type	Description
Addr	integer	Transporter memory address

The function returns a byte of data from location Addr in the internal memory of the Transporter.

An example of this function is:

```
var tbyte: byte;
....
tbyte := OCpeekTrans ($00E1);  { get number of retries }
```

OCpokeTrans Procedure

OCpokeTrans is used to alter internal memory of the Transporter. See the "Omninet Programmer's Guide" for more information on the Transporter poke command. The definition of this procedure is:

```
PROCEDURE OCpokeTrans (Addr: integer; Value: byte);
```

Parameter	Data Type	Description
Addr	integer	Transporter memory address
Value	byte	Data byte to move to the Transporter memory

The function moves the byte of Value to location Addr in the internal memory of the Transporter.

An example of this procedure is:

```
OCpokeTrans ($00E1,10);  { set number of retries }
```

The Omninet Transporter Interface Unit

ccOTCio

The Omninet Transporter Interface Unit is used to interface with the Corvus Omninet local area network. This unit functionally replaces unit ccOMNio.

This document does not define the various Omninet operations, but details the use of the unit functions and procedures available for interacting with the Omninet network. See the "Omninet Programmer's Guide" for a detailed description of the Omninet operations.

The ccOTCio unit USES no other units.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccOTCio;
```

If the Omninet Transporter driver is not loaded, the Transporter is used directly and no interrupt processing is performed. This is similar to the processing in unit ccOMNio. Currently, the Omninet Transporter driver is automatically loaded during system initialization on 512k systems.

ccOTCio Unit Constants

Constants defined in ccOTCio are:

Omninet Transporter Driver Return Codes

Identifier	Value	Description
TCnotRdy	21	Transporter not ready
TCqueued	30	Command queued warning
TCentUse	52	Entry in use error
TCinvFnc	56	Invalid function code error

Transporter Return Codes

Identifier	Value	Description
OkCode	0	Successful operation
GaveUp	128	Aborted a send command after maximum retries
TooLong	129	Last message sent was too long for the receiver
NoSockt	130	Sent to an uninitialized socket
HdrErr	131	Sender's header length did not match receiver's header length
BadSock	132	Invalid socket number
Inuse	133	Tried to set up a receive on an active socket
BadDest	134	Sent to an invalid host number

(continued on next page)

Transporter Return Codes (continued)

Identifier	Value	Description
Echoed	192	Echo command was successful
CmdAcpt	254	Command accepted
NoTrans	255	Unable to communicate with Transporter

Index into Transporter Counters

Identifier	Value	Description
TCCmiss	1	Missed packets. Number of ADLC address present interrupts
TCCcoll	2	Number of collision AVOIDANCE interrupts
TCCintErr	3	Number of unknown interrupts inside Transporter
TCCrcvErr	4	Number of ADLC receive errors (CRC, overrun, etc.)
TCCmaxCnt	4	Number of Transporter counters

Miscellaneous Values

Identifier	Value	Description
TCvers	n.n	Unit version number string
TCvrs64	100	Transporter version $64 number
TCvrs8A	138	Transporter version $8A number

ccOTCio Unit Types

Data types defined in ccOTCio are:

Data Type	Description
pTCbuffer	Omninet data buffer pointer
TCbuffer	Omninet data buffer
	array [0..32765] of -128..127;
pTCrsltRcd	Result record pointer
TCrsltRcd	Result record
	Rcode: byte; { command result code }
	Sourc: byte; { source host number }
	Len: integer; { received data length }
	UCdta: array [0..255] of -128..127;
	{ user control data }
pTComniCmd	Omninet command record pointer
TComniCmd	Omninet command record
	case integer of
	1: (p: record
	RP: pTCrsltRcd; { result record pointer }
	DP: pTCbuffer; { data buffer pointer }
	LN: integer; { data length }
	HL: integer; { header length }
	end);
	2: (a: array [1..12] of -128..127);
pTCparmBlk	Request parameter block record pointer
TCparmBlk	Request parameter block record
	pComd: pTComniCmd; { Omninet command pointer unit }
	pProc: pTCbuffer; { interrupt procedure ptr user }
	pPblk: pTCparmBlk; { parameter block pointer unit }
	pBuff: pTCbuffer; { data buffer pointer user }
	pRslt: pTCrsltRcd; { result record pointer user }
	oComd: TComniCmd; { Omninet command unit }
	rDone: boolean; { request complete if TRUE intr }
	rStat: integer; { request status intr }
	rRslt: integer; { request result code intr }

Data Type	Description
TChosts	Valid Omninet host numbers
	0..63;
TChostSet	Set of Omninet host numbers
	set of TChosts;
TCcntBuf	Transporter counters buffer
	array [1..TCCmaxCnt] of longint;

ccOTCio Unit Variables

Variables defined in ccOTCio are:

Variable	Data Type	Description
TCtrnVrsn	integer	Transporter version number
TChaveDrv	boolean	TRUE if using Omninet driver
TCcounts	TCcntBuf	Transporter counters
TCadlc	integer	Status of ADLC at last recv error

ccOTCio Unit Functions and Procedures

Procedures defined in ccOTCio are:

Procedure	Description
ccOTCioInit	Unit initialization
ccOTCioTerm	Unit termination
TCinitBlk	Initialize request control block
TCinterrupt	Basic interrupt processing
TCgetCounts	Update unit Transporter counts

Functions defined in ccOTCio are:

Function	Description
TCsetRecv	Set up receive
TCsndMesg	Send message
TCendRecv	End receive
TCwhoAmI	Get Transporter number
TCechoTrans	Echo to specified Transporter
TCpeekTrans	Read from Transporter memory
TCpokeTrans	Write to Transporter memory
TCsetRetry	Set Transporter retry count
TCnetMap	Get set of active Transporter numbers

ccOTCioInit Procedure

ccOTCioInit initializes the ccOTCio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccOTCioInit;
```

The procedure performs the following:

* Determines if the Omninet Transporter driver is loaded
* Sets unit Omninet event counters to 0
* Gets the Transporter version number

An example of this procedure is:

```
ccOTCioInit;
```

ccOTCioTerm Procedure

ccOTCioTerm terminates the ccOTCio unit. This procedure is called prior to program termination. The definition of this procedure is:

```
PROCEDURE ccOTCioTerm;
```

The procedure cancels all outstanding receives set by the current program. An example of this procedure is:

```
ccOTCioTerm;
```


TCgetCounts Procedure

TCgetCounts updates the Transporter counters maintained in this unit and resets the internal Transporter counters. The definition of this procedure is:

```
PROCEDURE TCgetCounts;
```

Counters maintained in array TCcounts are updated with current Transporter values. TCcounts is defined in the global variable section of this unit. Offsets within TCcounts are:

Identifier	Value	Description
TCCmiss	1	Missed packets. Number of ADLC address present interrupts
TCCcoll	2	Number of collision AVOIDANCE interrupts
TCCintErr	3	Number of unknown interrupts inside Transporter
TCCrcvErr	4	Number of ADLC receive errors (CRC, overrun, etc.)

An example of this procedure is:

```
TCgetCounts;
writeln ('TCCmiss    = ',TCcounts[TCCmiss]:1);
writeln ('TCCcoll    = ',TCcounts[TCCcoll]:1);
writeln ('TCCintErr = ',TCcounts[TCCintErr]:1);
writeln ('TCCrcvErr = ',TCcounts[TCCrcvErr]:1);
```

TCinitBlk Procedure

TCinitBlk initializes the specified request parameter block with default values. The definition of this procedure is:

```
PROCEDURE TCinitBlk (var ReqBlk: TCparmBlk;
                         pResRcd: pTCrsltRcd;
                         pDtaRcd: pTCbuffer;
                         pIntPro: pTCbuffer);
```

Parameter	Data Type	Description
ReqBlk	TCparmBlk	Request parameter block
pResRcd	pTCrsltRcd	Result record pointer
pDtaRcd	pTCbuffer	Data buffer pointer
pIntPro	pTCbuffer	Interrupt processing pointer

The procedure initializes the specified request parameter block as follows:

TCparmBlk	Request parameter block record
pComd	Pointer to oComd in this record
pProc	Specified interrupt processing pointer
pPblk	Specified request parameter block pointer
pBuff	Specified data buffer pointer
pRslt	Specified result record pointer
oComd	Omninet command (12 bytes of 0)
rDone	FALSE
rStat	0
rRslt	255 ($FF)

An example of this procedure is:

```
var pblk: TCparmBlk;
    rslt: TCrsltRcd;
    buff: array [1..512] of -128..127;
    Dsta: integer;
....
TCinitBlk (pblk, @rslt, @buff, @TCinterrupt);
                                        { init parm block }
Dsta := TCsetRecv (pblk, 1, 512, 0);    { set recv on $80 }
if Dsta <> 0 then ....          { Transporter driver error }
```

TCinterrupt Procedure

TCinterrupt updates the request parameter block with request completion information. The definition of this procedure is:

```
PROCEDURE TCinterrupt (QueFlg: integer;
                       DrvSta: integer;
                      pResRcd: pTCrsltRcd;
                      pDtaRcd: pTCbuffer;
                      pReqBlk: pTCparmBlk);
```

Parameter	Data Type	Description
QueFlg	integer	Request queued flag
DrvSta	integer	Driver status
pResRcd	pTCrsltRcd	Result record pointer
pDtaRcd	pTCbuffer	Data buffer pointer
pReqBlk	pTCparmBlk	Request parameter block pointer

The procedure sets the following fields in the request parameter block when a request is complete (QueFlg = 0) or when a request is terminated with an error (DrvSta <> 0):

TCparmBlk	Request parameter block record (partial)
rDone	TRUE
rStat	Returned driver status
rRslt	Result code from Omninet result record

This procedure is used by this unit for all functions except TCsndMesg and TCsetRecv. TCinterrupt may also be used for simple TCsndMesg and TCsetRecv completion processing.

TCsetRecv Function

TCsetRecv prepares the specified socket to receive a single message. The definition of this function is:

```
FUNCTION TCsetRecv (var ReqBlk: TCparmBlk;
                    ScktNbr, DtaLen, HdrLen: integer): integer;
```

Parameter	Data Type	Description
ReqBlk	TCparmBlk	Request parameter block
ScktNbr	integer	Receive socket number in the range of 1 to 4 (1..4 = sockets $80..$80)
DtaLen	integer	~~Send~~ Recv data buffer length
HdrLen	integer	User control data length

The function result is the Transporter driver request status. After executing the function, the rRslt field of the specified request parameter block contains the Omninet command result. Valid command results are:

Identifier	Value	Description
....	0	Successful operation
BadSock	132	Invalid socket number
Inuse	133	Tried to set up a receive on an active socket
CmdAcpt	254	Command accepted

If the command is successful, the rRslt field of the specified request parameter block contains the the value CmdAcpt until a message is received for the socket.

If the rRslt field contains CmdAcpt, the rDone field is FALSE, otherwise, rDone is TRUE.

The rDone field of the request parameter block is set to TRUE when a message is received for the socket.

The user's interrupt procedure is responsible for updating the following fields in the request parameter block:

TCparmBlk	Request parameter block record (partial)
rDone	TRUE
rStat	Returned driver status
rRslt	Result code from Omninet result record

If only these fields need to be updated in the interrupt procedure, the TCinterrupt procedure in this unit may be used as the user's interrupt procedure.

An example of this function is:

```
var pblk:  TCparmBlk;
    Rbuff: array [1..512] of -128..127; { recv message buffer }
    Rrslt: TCrsltRcd;                   { recv result buffer  }
    Dsta:  integer;
....
TCinitBlk (pblk,@Rrslt,@Rbuff,@TCinterrupt);
Dsta := TCsetRecv (pblk,1,512,0);
                                  { set receive on socket 1    }
if Dsta <> 0 then ....            { Transporter driver error   }
while NOT pblk.rDone do;          { wait until mesg received   }
if pblk.rRslt < 127 then ....     { mesg received successfully }
                    else ....     { error processing           }
```

TCsndMesg Function

TCsndMesg sends a message to the specified host and socket. The definition of this function is:

```
FUNCTION TCsndMesg (var ReqBlk: TCparmBlk;
                        ScktNbr, DtaLen, HdrLen,
                    DestHost: integer): integer;
```

Parameter	Data Type	Description
ReqBlk	TCparmBlk	Request parameter block
ScktNbr	integer	Destination socket number in the range of 1 to 4 (1..4 = sockets $B0..$B0)
DtaLen	integer	Send data buffer length
HdrLen	integer	User control data length
DestHost	integer	Destination host number

The function result is the Transporter driver request status. If the interrupt procedure pointer in the request parameter block (ReqBlk.pProc) is NIL, the function waits for command completion before returning. After executing the function and waiting for command completion, the rRslt field of the specified request parameter block contains the Omninet command result. Valid command results are:

Identifier	Value	Description
....	0..127	Successful operation (retries)
GaveUp	128	Aborted a send command after maximum retries
TooLong	129	Last message sent was too long for the receiver
NoSockt	130	Sent to an uninitialized socket

(continued on next page)

Identifier	Value	Description
HdrErr	131	Sender's header length did not match receiver's header length
BadSock	132	Invalid socket number
BadDest	134	Sent to an invalid host number

The user's interrupt procedure is responsible for updating the following fields in the request parameter block:

TCparmBlk	Request parameter block record (partial)
rDone	TRUE
rStat	Returned driver status
rRslt	Result code from Omninet result record

If only these fields need to be updated in the interrupt procedure, the TCinterrupt procedure in this unit may be used as the user's interrupt procedure.

An example of this function is:

```
var pblk:  TCparmBlk;
    Sbuff: array [1..512] of -128..127; { send message buffer }
    Srslt: TCrsltRcd;                   { send result buffer  }
    Dsta:  integer;
....
TCinitBlk (pblk,@Srslt,@Sbuff,@TCinterrupt);
Dsta := TCsndMesg (pblk,1,512,0,63);
                                  { send to host 63, socket 1  }
if Dsta <> 0 then ....            { Transporter driver error   }
while NOT pblk.rDone do;          { wait until mesg sent       }
if pblk.rRslt = 0 then ....       { mesg sent successfully     }
                  else ....       { error processing           }
.... or ....

TCinitBlk (pblk,@Srslt,@Sbuff,NIL);
Dsta := TCsndMesg (pblk,1,512,0,63);
                                  { send to host 63, socket 1  }
if Dsta <> 0 then ....            { Transporter driver error   }
if pblk.rRslt = 0 then ....       { mesg sent successfully     }
                  else ....       { error processing           }
```


TCendRecv Function

TCendRecv disables reception of any more messages for the specified socket until another TCsetRecv command is issued for the socket. The definition of this function is:

```
FUNCTION TCendRecv (ScktNbr: integer;
                    var CmdRslt: integer): integer;
```

Parameter	Data Type	Description
ScktNbr	integer	Receive socket number in the range of 1 to 4 (1..4 = sockets $80..$80)
CmdRslt	integer	Omninet command result

The function result is the Transporter driver request status. After executing the function, CmdRslt contains the Omninet command result. Valid command results are:

Identifier	Value	Description
OkCode	0	Successful operation
BadSock	132	Invalid socket number specified

An example of this function is:

```
var Dsta,Osta: integer;
....
Dsta := TCendRecv (1,status); { end receiving on socket 1 }
if Dsta <> 0 then ....        { Transporter driver error  }
if Osta <> 0 then ....        { Omninet command error     }
```

TCwhoAmI Function

TCwhoAmI returns the Transporter number of the host computer. The definition of this function is:

```
FUNCTION TCwhoAmI (var HostNmbr: integer): integer;
```

Parameter	Data Type	Description
HostNmbr	integer	Host Transporter number

The function result is the Transporter driver request status. After executing the function, HostNmbr contains the Transporter number of the host computer.

An example of this function is:

```
var Dsta,TransNbr: integer;
....
Dsta := TCwhoAmI (TransNbr);
if Dsta <> 0 then .... { Transporter driver error }
writeln ('Our Transporter number is ',TransNbr:1);
```


TCechoTrans Function

TCechoTrans requests the Transporter to send an echo packet to the specified host. The echo packet is used to verify the presence of another network device without disturbing that device. The definition of this function is:

```
FUNCTION TCechoTrans (DestHost: integer;
                      var CmdRslt: integer): integer;
```

Parameter	Data Type	Description
DestHost	integer	Destination host number
CmdRslt	integer	Omninet command result

The function result is the Transporter driver request status. The function sends an echo packet to Transporter DestHost. Transporter DestHost receives the packet and acknowledges without informing the attached host computer. After executing the function, CmdRslt contains the Omninet command result. Valid command results are:

Identifier	Value	Description
GaveUp	128	Aborted a send command after maximum retries
BadDest	134	Sent to an invalid host number
Echoed	192	Echo command was successful

An example of this function is:

```
var Dsta,Osta: integer;
....
Dsta := TCechoTrans (1,Osta); { is Transporter 1 active?    }
if Dsta <> 0 then ....        { Transporter driver error    }
if Osta = Echoed then ....    { Transporter responded       }
                  else ....   { Transporter did not respond }
```

TCpeekTrans Function

TCpeekTrans is used to examine internal memory of the Transporter. See the "Omninet Programmer's Guide" for more information on the Transporter peek command. The definition of this function is:

```
FUNCTION TCpeekTrans (Addr: integer;
                      var Value: integer): integer;
```

Parameter	Data Type	Description
Addr	integer	Transporter memory address
Value	integer	Data byte value moved from Transporter memory

The function result is the Transporter driver request status. The function returns the unsigned byte value of data from location Addr in the internal memory of the Transporter.

An example of this function is:

```
var Dsta,Tvalue: integer;
....
Dsta := TCpeekTrans ($E1,Tvalue);     { get nmbr of retries }
if Dsta <> 0 then ....                { Transporter driver error }
```


TCpokeTrans Function

TCpokeTrans is used to alter internal memory of the Transporter. See the "Omninet Programmer's Guide" for more information on the Transporter poke command. The definition of this function is:

```
FUNCTION TCpokeTrans (Addr, Value: integer;
                      var CmdRslt: integer): integer;
```

Parameter	Data Type	Description
Addr	integer	Transporter memory address
Value	integer	Data byte value to move to Transporter memory
CmdRslt	integer	Omninet command result

The function result is the Transporter driver request status. The function moves the unsigned byte value of data to location Addr in the internal memory of the Transporter.

An example of this function is:

```
var Dsta,Osta: integer;
....
Dsta := TCpokeTrans ($E1,10,Osta);    { set nmbr of retries }
if Dsta <> 0 then ....                { Transporter driver error }
if Osta <> 0 then ....                { Omninet command error    }
```

TCsetRetry Function

TCsetRetry is used to set the number of Transporter retries. The definition of this function is:

```
FUNCTION TCsetRetry (Retries: integer): integer;
```

Parameter	Data Type	Description
Retries	integer	Number of Transporter retries

The function result is the Transporter driver request status. The function sets the specified number of retries into the internal memory of the Transporter.

An example of this function is:

```
var Dsta: integer;
....
Dsta := TCsetRetry (3);          { set number of retries to 3 }
if Dsta <> 0 then ....           { Transporter driver error   }
```

TCnetMap Function

TCnetMap is used to define a set of active network hosts. The definition of this function is:

```
FUNCTION TCnetMap (var NetMap: TChostSet); integer;
```

Parameter	Data Type	Description
NetMap	TChostSet	Set of active host numbers

The function result is the Transporter driver request status.

An example of this function is:

```
var Dsta,tn: integer; map: TChostSet;
....
Dsta := TCnetMap (map);
if Dsta <> 0 then ....          { Transporter driver error   }
for tn := 0 to 63 do begin
    if tn in map then
        writeln ('   Transporter ',tn:1,' is active');
    end;
writeln;
```

Omninet Transporter Unit Example Program

The following simple program illustrates using the Omninet Transporter driver unit.

```
program ot;

uses {$u /ccutil/cclib} ccDEFN, ccCRTio, ccOTCio;

procedure RunTest;
      var i,r,tn: integer;
          map:  TChostSet;
          tcp:  TCparmBlk;
          rslt: TCrsltRcd;
          buff: array [1..512] of -128..127;
      begin
      writeln; writeln ('TChaveDrv = ',TChaveDrv);
               writeln ('TCtrnVrsn = ',TCtrnVrsn:1); writeln;

      writeln ('TCwhoAmI test');
      r := TCwhoAmI (i);
      writeln ('   result = ',r:1,'  transporter number = ',i:1);
      writeln;

      writeln ('TCechoTrans test');
      for tn := 0 to 63 do begin
          r := TCechoTrans (tn,i);
          if (r <> 0) or (i = echoed) then
              writeln ('   result = ',r:1,
                       '   transporter number = ',tn:1);
          end;
      writeln;

      writeln ('TCnetMap test (TCpokeTrans & TCsetRetry)');
      r := TCnetMap (map);
      writeln ('   result = ',r:1);
      for tn := 0 to 63 do begin
          if tn in map then
              writeln ('   transporter number = ',tn:1);
          end;
      writeln;

      writeln ('TCsetRetry test (TCpeekTrans & TCpokeTrans)');
      r := TCsetRetry (1);
      writeln ('   result = ',r:1);
      r := TCpeekTrans ($E1,i);
      writeln ('   result = ',r:1,'   retries = ',i:1);
      r := TCsetRetry (10);
      writeln ('   result = ',r:1);
      r := TCpeekTrans ($E1,i);
```

```
    writeln ('   result = ',r:1,'   retries = ',i:1);
    writeln;

    writeln ('TCgetCounts test (TCpeekTrans & TCpokeTrans)');
    TCgetCounts;
    writeln ('   TCCmiss    = ',TCcounts[TCCmiss]:1);
    writeln ('   TCCcoll    = ',TCcounts[TCCcoll]:1);
    writeln ('   TCCintErr  = ',TCcounts[TCCintErr]:1);
    writeln ('   TCCrcvErr  = ',TCcounts[TCCrcvErr]:1);
    writeln ('   TCadlc     = ',TCadlc:1);
    writeln;

    writeln ('TCsetRecv test');
    for i := 0 to 5 do begin
        TCinitBlk (tcp, @rslt, @buff, NIL);
        r := TCsetRecv (tcp, i, 512, 0);
        writeln ('   result = ',r:1,
                 '   socket = ',i:1,
                 '   transporter result = ',tcp.rRslt:1);
        end;
    for i := 4 downto 1 do begin
        TCinitBlk (tcp, @rslt, @buff, NIL);
        r := TCsetRecv (tcp, i, 512, 0);
        writeln ('   result = ',r:1,
                 '   socket = ',i:1,
                 '   transporter result = ',tcp.rRslt:1);
        end;
    writeln;

    writeln ('TCendRecv test');
    for i := 0 to 5 do begin
        r := TCendRecv (i, tn);
        writeln ('   result = ',r:1,
                 '   socket = ',i:1,
                 '   transporter result = ',tn:1);
        end;
    for i := 4 downto 1 do begin
        r := TCendRecv (i, tn);
        writeln ('   result = ',r:1,
                 '   socket = ',i:1,
                 '   transporter result = ',tn:1);
        end;
    writeln;

    end;


    begin
    ccCRTioInit;
    CrtAction (EraseALL);
    writeln ('ccOTCio unit test'); writeln;
```

```
      ccOTCioInit;
      RunTest;
      ccOTCioTerm;
      end.
```

The output generated by this program is:

```
ccOTCio unit test


TChaveDrv =  TRUE
TCtrnVrsn = 100

TCwhoAmI test
   result = 0  transporter number = 11

TCechoTrans test
   result = 0   transporter number = 0

TCnetMap test (TCpokeTrans & TCsetRetry)
   result = 0
   transporter number = 0

TCsetRetry test (TCpeekTrans & TCpokeTrans)
   result = 0
   result = 0   retries = 1
   result = 0
   result = 0   retries = 10

TCgetCounts test (TCpeekTrans & TCpokeTrans)
   TCCmiss   = 0
   TCCcoll   = 0
   TCCintErr = 0
   TCCrcvErr = 0
   TCadlc    = 0

TCsetRecv test
   result = 0   socket = 0   transporter result = 255
   result = 0   socket = 1   transporter result = 254
   result = 0   socket = 2   transporter result = 254
   result = 0   socket = 3   transporter result = 254
   result = 0   socket = 4   transporter result = 254
   result = 0   socket = 5   transporter result = 255
   result = 0   socket = 4   transporter result = 133
   result = 0   socket = 3   transporter result = 133
   result = 0   socket = 2   transporter result = 133
   result = 0   socket = 1   transporter result = 133

TCendRecv test
```



```
result = 0   socket = 0   transporter result = 132
result = 0   socket = 1   transporter result = 0
result = 0   socket = 2   transporter result = 0
result = 0   socket = 3   transporter result = 0
result = 0   socket = 4   transporter result = 0
result = 0   socket = 5   transporter result = 132
result = 0   socket = 4   transporter result = 0
result = 0   socket = 3   transporter result = 0
result = 0   socket = 2   transporter result = 0
result = 0   socket = 1   transporter result = 0
```

Omninet Transporter Driver Background Information

The following sections give a brief description of the Omninet Transporter driver. Topics discussed in the background information sections do not have to be understood in order to use the Transporter commands unit.

The Transporter driver has three main functions:

1. "strobe in" Transporter commands,
2. handle Transporter generated interrupts,
3. ensure only one command and one receive on each socket is attempted at the same time.

Access to the driver is through the UnitStatus mechanism of the Corvus Concept Operating System. The Pascal defined UnitStatus call is:

```
UnitStatus (UnitNmbr, ParmBlock, FuncCode);
```

where UnitNmbr is the unit number of the Omninet Transporter driver, ParmBlock is the Omninet Transporter driver parameter block, and FuncCode is one of the valid function codes for the driver.

ParmBlock has the form:

```
    record CommandPointer:    pBytes;
           ProcedurePointer:  pBytes;
           UserData:          LongWord;
           end;
```

where CommandPointer is a pointer to the Transporter Control Block to be "strobed" into the Transporter. ProcedurePointer is a pointer to the global level procedure the Transporter driver calls when the Transporter interrupt occurs. UserData is a four byte (long word) data field which is user-defined.

The CommandPointer must point to a valid Transporter Control Block above the address $80000, except for the "Clear Receive Socket" function in which case the pointer can be NIL.

The ProcedurePointer must point to a valid user interrupt service routine or may be NIL. A NIL ProcedurePointer indicates NO user interrupt service routine to call when the operation is dequeued or complete. The interrupt service routine must accept five parameters: a dequeue flag, a status code, the result and buffer pointers from the Transporter Control Block, and the UserData parameter from the ParmBlock (see description in interrupt routine discussion).

The dequeue flag is non-zero when the interrupt service routine is called for an operation start attempt. The dequeue operation start call to the interrupt service routine is made only if the Transporter Control Block was queued by the driver. The status code describes the IORESULT code for the operation start attempt. If the status code is non-zero, the control block specified failed to be "strobed in."

The dequeue flag is zero when the interrupt routine is called after a complete operation.

The user interrupt procedure interface is:

```
Procedure Done (DequeFlag: integer;   {dequeue flag                }
                DrvStatus: integer;   {driver status               }
                ResultPtr: pBytes;    {user may use any pointer..  }
                BufferPtr: pBytes;    {..type for these pointers   }
                UserData:  longint);  {can be any long word type   }
```

Failure to comply with these rules results in catastrophic consequences.

The UserData parameter is available for any purpose the user determines. It is not examined or used for any purpose by the Transporter driver. It is returned to the user's interrupt service routine. For example, it may be a pointer to an operation control block, a transaction code, or an index into an array.

The ccOTCio unit uses this field to point to a request parameter block which contains all information needed to process an Omninet Transporter command.

The Omninet Transporter driver functions are:

0	Current Command.
1 2 3 4	Setup Receive Socket. The function code specifies which socket, where 1 is socket $80, 2 is socket $90 and so on.
129 130 131 132	Clear Receive Socket. The function code specifies which socket, where 129 is socket $80, 130 is socket $90 and so on.

Each of the functions use the same parameter block. If any other function codes are used, the driver returns a status code indicating the source of the error (see section on error and warning codes).

Driver Functions

The driver functions use an internal table to control the operations. This table has five entries, one for the Current Command and four for each of the receive sockets.

The Current Command entry is used for send, peek, poke, init, echo, and other immediate Transporter commands sent to the driver via the Current Command function interface.

The "Clear Receive Socket" function uses the Current Command entry if the command address is not NIL. This is used to send an "End Receive" command to the Transporter.

The "Setup Receive Socket" function also uses the Current Command entry to transmit the "Setup Receive" command to the Transporter. It uses the receive socket entry if the command is successfully strobed into the Transporter.

Each entry in the table can be in use for only one command at a time. The four receives sockets may each have only one receive pending at a time. If a "Setup Receive Socket" function is requested on a socket with a receive pending, an "In Use" status code is returned to the caller and the driver takes no action.

The current command entry may have only one command pending on it. However, if a current command request is made while one is currently pending, the driver queues the new request if there is room in the queue. The driver returns a warning code if it queues a request (see error and warning code section). The "Setup Receive Socket" function does not setup the socket entry if the current command is queued. When the request is dequeued, the receive will be setup, if possible. The user's interrupt procedure is called with a non-zero dequeue flag. If the driver cannot setup the receive, the status parameter is non-zero defining the I/O error. Otherwise, the status parameter is zero, indicating no error on dequeue.

The "Setup Receive Socket" function does special processing on the result code of the current command if it is successfully transmitted to the Transporter. The function waits for a change of the result code instead of depending on the driver's interrupt service routine to release the current command entry. If the result code does not change within a certain time, the driver returns the "Transporter Not Ready" status code. It also clears both the current command and receive socket entries. However, if the Transporter changes the result code to an error state, the function does NOT release the socket entry. It assumes the receive is setup. Therefore, the caller must do a "Clear Receive Socket" function with the command address NIL to free up the receive socket entry.

For the peek command, the Transporter returns the data as the result code. Since the driver determines which command and, therefore, which entry caused the interrupt by the result code, a peek response of $FF causes the driver to miss the interrupt for the peek completion. Consequently, the driver would not release the current command entry. To prevent this, the driver waits on the completion of the peek command. If the result code does not change within a certain time, the driver assumes the peek command returned an $FF and releases the current command entry. Unfortunately, this means a user should not wait on an "In Use" error response from the driver in an interrupt routine if it is possible that a peek command is pending in the current command entry. For this last case, if the peek response is $FF, the current command entry will not be released.

Interrupt Service Routine

If the user specifies an interrupt service routine, this routine must be resident in memory the entire time the Transporter command is active. Furthermore, if the code is Pascal, the routine must be a global level procedure. A global level procedure is nested only under the MAIN level program.

The interrupt service routine must support two types of processing. The first occurs when the dequeue flag is zero indicating the operation is complete. The interrupt service routine should do normal operation complete processing, which may entail calling the Transporter driver again to initiate another command. The second occurs when the dequeue flag is non-zero indicating a previously queued command has been dequeued and processed by the driver to start the Transporter operation. The interrupt service routine must check the status code parameter which describes any error condition found when initiating the command as it was removed from the queue. If the status code is zero, the set up was successful, otherwise, the set up failed. The status code is usually a "Transporter Not Ready" error. This indicates a good probability of a hardware malfunction with the Transporter. However, it can be the "In Use" error if the operation requested is a "Setup Receive Socket" function. If the setup succeeded, the interrupt service routine is called again when the operation is complete. If it failed, the interrupt service routine is NOT be called for an operation completion.

Normally, the interrupt service routine is called after the command has been performed and the entry has been released. Prior to the interrupt service routine call, the driver restores the interrupt level to the state when the Transporter interrupt occurred. Upon return, the driver resets the interrupt level to disable Transporter interrupts. For the dequeue call, the command is not performed, the current command entry is not released, and the entry is not removed from the queue, therefore, the user's normal processing may cause problems. After the interrupt service routine returns from the dequeue call, the current command entry is released and the entry is removed from the queue.

The user's interrupt service routine must be very careful about reentrancy problems, such as changing global variables from within the interrupt service routine or calling non-reentrant system functions. This interrupt service routine may call other procedures and functions within the user's program. It should not call the display or keyboard drivers because of timing and reentrancy problems. The driver's interrupt routine saves and restores IORESULT, a potential source of reentrancy problems in the system. This protects the user's interrupt routines which call the driver from damaging I/O error reporting from the system after the interrupt call is completed.

The interrupt service routine is never called for a "Setup Receive Socket" function. This function forces the current command entry's procedure pointer to NIL when it calls the Current Command function.

Error and Warning Codes

The following is a summary of the error and warning codes returned by the driver to the calling routine.

3	Invalid I/O request error
	The driver returns this code if the command code is invalid. The command codes defined by the Concept OS are in the range of 0 to 6, inclusive. The driver supports only: UnitInstall (command code = 0), UnitStatus (command code = 5), UnitUnmount (command code = 6). All other commands are invalid.
21	Transporter not ready error
	The driver returns this code if the Transporter fails to respond ready in time when trying to "strobe in" the command. The driver also responds with this code when it times out waiting for the "Setup Receive" response from the Transporter.
30	Queued request warning
	The driver returns this code whenever it queues a request. This can occur for "Current Command" and "Setup Receive Socket" function calls.
52	Entry in use error
	The driver returns this code whenever the specified entry or implied entry has a command pending on it. The "Clear Receive Socket" function returns this error if the current command entry is in use. The "Setup Receive Socket" function returns this error if the specified receive socket entry is in use or the current command entry is in use and the queue is full. The "Current Command" function returns this error only if the current command entry is in use and the queue is full.
56	Invalid function code error
	The driver returns this code if the user passes a function code to the UnitStatus command that is not 0, 1, 2, 3, 4, 129, 130, 131 or 132.

The Window Control Unit

ccWNDio

The Window Control Unit is used to interface with the display driver window functions.

The ccWNDio unit USES unit ccDEFN.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccWNDio;
```

ccWNDio Unit Constants

Constants defined in ccWNDio are:

WinCreate Function Flag Values

Identifier	Value	Description
WfgGraf	2	Graphics mode
WfgCursOn	4	Cursor on
WfgInvCur	8	Inverse cursor
WfgWrap	16	Line wrap
WfgScrOff	32	Scroll off
WfgClrPg	64	Clear page

WinSystem Function System Window Select Codes

Identifier	Value	Description
WsysCurr	1	Current process window
WsysCmd	2	Cmd/msg window
WsysRoot	3	Root user window (full screen)

ccWNDio Unit Types

Data types defined in ccWNDio are:

Data Type	Description
pCharSet	Character set record pointer
CharSet	Character set record

```
    tblloc:  pBytes;    { character set data pointer       }
    lpch:    integer;   { scanlines per character         }
    bpch:    integer;   { bits per character              }
    frstch:  integer;   { first character code - ascii    }
    lastch:  integer;   { last character code - ascii     }
    mask:    longint;   { mask used in positioning cells  }
    attr1:   byte;      { attributes                      }
                        { bit 0 = 1 - vertical orientation }
    attr2:   byte;      { currently unused (always = 0 )  }
```

Data Type	Description
pWndRcd	Window record pointer

WndRcd	Window record

```
  charpt:  pCharSet;  { character set record pointer      }
  homept:  pBytes;    { home (upper left) pointer         }
  curadr:  pBytes;    { current location pointer          }
  homeof:  integer;   { bit offset of home location       }
  basex:   integer;   { home x value, rel to root window  }
  basey:   integer;   { home y value, rel to root window  }
  lngthx:  integer;   { maximum x value, bits rel to wnd  }
  lngthy:  integer;   { maximum y value, bits rel to wnd  }
  cursx:   integer;   { current x value, bits rel to wnd  }
  cursy:   integer;   { current y value, bits rel to wnd  }
  bitofs:  integer;   { bit offset of current address     }
  grorgx:  integer;   { graphics origin x, bits home rel  }
  grorgy:  integer;   { graphics origin y, bits home rel  }
  attr1:   byte;      { inverse, underscore, insert       }
  attr2:   byte;      { v/h, graph/char, cursor on/off,   }
                      { cursor inv/underline              }
  state:   byte;      { for decoding escape sequences     }
  rcdlen:  byte;      { window description record length  }
  attr3:   byte;      { enhanced character set attributes }
  fill1:   byte;      { currently unused                  }
  fill2:   byte;      { currently unused                  }
  fill3:   byte;      { currently unused                  }
  fill4:   longint;   { currently unused                  }
  wwsptr:  pBytes;    { window working storage pointer    }
```

ccWNDio Unit Variables

Variables defined in ccWNDio are:

None.

ccWNDio Unit Functions and Procedures

Procedures defined in ccWNDio are:

Procedure	Description
ccWNDioInit	Unit initialization

Functions defined in ccWNDio are:

Function	Description
WinSystem	Select a system defined window
WinCreate	Create a user defined window
WinSelect	Select a user defined window
WinDelete	Delete a user defined window
WinClear	Clear a user defined window
WinStatus	Get status of current window
WinLoadCh	Load character set for window

ccWNDioInit Procedure

ccWNDioInit initializes the ccWNDio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccWNDioInit;
```

An example of this procedure is:

```
ccWNDioInit;
```

WinSystem Function

WinSystem selects a system defined window. The definition of this function is:

```
FUNCTION WinSystem (WN: integer): integer;
```

Parameter	Data Type	Description
WN	integer	System window selection code

The function returns the IORESULT from the display driver select window function. A value of 0 indicates a successful operation. WN is one of the following:

Identifier	Value	Description
WsysCurr	1	Current process window
WsysCmd	2	Cmd/msg window
WsysRoot	3	Root user window (full screen)
	4..20	Dynamic system windows created using the CreWndow key

An example of this function is:

```
var Wstatus: integer;

Wstatus := WinSystem (WsysCmd);
writeln ('This appears in the command window');
Wstatus := WinSystem (WsysCurr);
writeln ('This appears in the user window');
```

Wstatus is the status of the display driver select window function. This example selects the command window and outputs text. The user window (current window when the program was loaded) is then selected.

WinCreate Function

WinCreate creates a user defined window. The definition of this function is:

```
FUNCTION WinCreate (var WR: WndRcd; HomeX,HomeY,
                    Width,Lngth,Flags: integer): integer;
```

Parameter	Data Type	Description
WR	WndRcd	Window record of window to create
HomeX	integer	New window home coordinates relative to current window
HomeY	integer	
Width	integer	New window width (X size)
Lngth	integer	New window length (Y size)
Flags	integer	Flag codes from table below

The function returns the IORESULT from the display driver create window function. A value of 0 indicates a successful operation.

For text windows HomeX, HomeY, Width, and Lngth contain character position values. For graphics windows (WfgGraf in Flags) these variables contain pixel position values. HomeX and HomeY contain the home (upper left) position of the new window relative to the current window, either characters or pixels.

Width contains the number of positions in the X direction of the new window, either characters or pixels. Lngth contains the number of positions in the Y direction of the new window, either characters or pixels.

Flags contains the sum of window attributes from:

Identifier	Value	Description
WfgGraf	2	Graphics mode
WfgCursOn	4	Cursor on
WfgInvCur	8	Inverse cursor
WfgWrap	16	Line wrap
WfgScrOff	32	Scroll off
WfgClrPg	64	Clear page

An example of this function is:

```
var Wstatus: integer;
    homex,homey,width,lngth,curx,cury: integer;
    BaseX,BaseY,LngthX,LngthY,Wflags: integer;
    UserWindow: WndRcd;
....
Wstatus := WinStatus (homex,homey,width,lngth,curx,cury);
BaseX  := 0;      BaseY  := 0;
LngthX := width;  LngthY := 3;
Wflags := WfgCursOn + WfgInvCur;
Wstatus := WinCreate (UserWindow,
                      BaseX,BaseY,LngthX,LngthY,Wflags);
```

Wstatus is the status of the display driver create window function. UserWindow is the user window record for the new window. This example creates a three line text window at the top of the current window.

WinSelect Function

WinSelect selects a user defined window to be the current window. After the window is selected, all display driver activity affects only the newly selected window. The definition of this function is:

```
FUNCTION WinSelect (var WR: WndRcd): integer;
```

Parameter	Data Type	Description
WR	WndRcd	Window record of window to select

The function returns the IORESULT from the display driver select window function. A value of 0 indicates a successful operation. WR, the window record, must be created with the WinCreate function before selecting the user window.

- - - I M P O R T A N T - - -

Window records can not be local to a procedure. Place window records with the global program variables. When specifying window records in parameter strings, always specify them as a VAR parameter. Also, select the current system window before exiting a program by specifying:

```
Wstatus := WinSystem (WsysCurr);
```

- - - - - - - - - - - - - - - -

An example of this function is:

```
var Wstatus:     integer;
    UserWindow:  WndRcd;
....
Wstatus := WinSelect (UserWindow);
```

Wstatus is the status of the display driver select window function. UserWindow is the user window record for the window to be selected.

WinDelete Function

WinDelete deletes a user defined window. The definition of this function is:

```
FUNCTION WinDelete (var WR: WndRcd): integer;
```

Parameter	Data Type	Description
WR	WndRcd	Window record of window to delete

The function returns the IORESULT from the display driver delete window function. A value of 0 indicates a successful operation. WR: the window record, must be created with the WinCreate function before deleting the user window.

An example of this function is:

```
var Wstatus:     integer;
    UserWindow:  WndRcd;
    . . .
Wstatus := WinDelete (UserWindow);
```

Wstatus is the status of the display driver delete window function. UserWindow is the user window record for the window to be deleted.

WinClear Function

WinClear clears the window defined by the specified window record. The definition of this function is:

```
FUNCTION WinClear (var WR: WndRcd): integer;
```

Parameter	Data Type	Description
WR	WndRcd	Window record of window to clear

The function returns the IORESULT from the display driver clear window function. A value of 0 indicates a successful operation. WR, the window record, must be created with the WinCreate function before clearing the user window.

An example of this function is:

```
var Wstatus:    integer;
    UserWindow: WndRcd;

...
Wstatus := WinClear (UserWindow);
```

Wstatus is the status of the display driver clear window function. UserWindow is the user window record for the window to be cleared.

WinStatus Function

WinStatus sets six integer variables which define the status of the current window. The definition of this function is:

```
FUNCTION WinStatus (var HomeX,HomeY,Width,Lngth,
                        CurX,CurY: integer): integer;
```

Parameter	Data Type	Description
HomeX	integer	Current window absolute home coordinates (relative to entire screen)
HomeY	integer	
Width	integer	Current window width (X size)
Lngth	integer	Current window length (Y size)
CurX	integer	Current cursor position relative to current window
CurY	integer	

The function returns the IORESULT from the display driver window status function. A value of 0 indicates a successful operation.

For text windows all variables contain character positions values. For graphics windows (see WinCreate) all variables contain pixel positions. HomeX and HomeY contain the absolute home (upper left corner of full screen) position of the current window, either characters or pixels.

Width contains the number of positions in the X direction of the current window, either characters or pixels. Lngth contains the number of positions in the Y direction of the current window, either characters or pixels.

CurX and CurY contain the current cursor position in the current window, either characters or pixels.

An example of this function is:

```
var Wstatus: integer;
    AbsHomeX,AbsHomeY,LngthX,LngthY,CursorX,CursorY: integer;
....
Wstatus := WinStatus (AbsHomeX,AbsHomeY,
                      LngthX,LngthY,CursorX,CursorY);
```

Wstatus is the status of the display driver window status function.

WinLoadCh Function

WinLoadCh loads the specified character set for the current window. The definition of this function is:

```
FUNCTION WinLoadCh (CSname: string80): integer;
```

Parameter	Data Type	Description
CSname	string80	File name of character set

The Window Manager program is used to load the specified character set. The function returns the result from calling the Window Manager program. A value of 0 indicates a successful character set load.

An example of this function is:

```
var Wstatus: integer;
....
Wstatus := WinLoadCh ('/CCUTIL/CSD.07.11.ALT');
```

Wstatus is the status of the character set load returned by the Window Manager program.

The Graphics Control Unit

TurtleGraphics

The TurtleGraphics Unit is used to interface with the display driver graphics functions. The Corvus Concept implementation is a subset of other implementations of TurtleGraphics.

The TurtleGraphics unit USES no other units.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} TurtleGraphics;
```

TurtleGraphics Unit Constants

Constants defined in TurtleGraphics are:

Identifier	Description
TurtleVersion	Current version number string

TurtleGraphics Unit Types

Data types defined in TurtleGraphics are:

Data Type	Description		
ScreenColor	Pen colors and screen colors		
	None	0	No color

(continued on next page)

Data Type	Description		
ScreenColor	Pen colors and screen colors (continued)		
	White	1	White
	Black	2	Black
	Reverse	3	White --> Black Black --> White Green --> Violet Violet --> Green Orange --> Blue Blue --> Orange
	Radar	4	Not used
	Black1	5	Black
	Green	6	Move - Reverse pixel Fill - Density = 2
	Violet	7	Move - Reverse pixel Fill - Density = 2
	White1	8	White
	Black2	9	Black
	Orange	10	Move - Reverse pixel Fill - Density = 3
	Blue	11	Move - Reverse pixel Fill - Density = 3
	White2	12	White

TurtleGraphics Unit Variables

Variables defined in TurtleGraphics are:

None.

TurtleGraphics Unit Functions and Procedures

Procedures defined in TurtleGraphics are:

Procedure	Description
InitTurtle	Unit initialization
GrafMode	Set graphics mode
TextMode	Set text mode
ViewPort	Set view port
PenColor	Set pen color
FillScreen	Fill view port with color
Turn	Turn turtle (relative to current)
TurnTo	Turn turtle (absolute)
Move	Move turtle (relative to current)
MoveTo	Move turtle (absolute)

Functions defined in TurtleGraphics are:

Function	Description
TurtleX	Get current turtle X coordinate
TurtleY	Get current turtle Y coordinate
TurtleAng	Get current turtle angle
ScreenBit	Test for displayed pixel

InitTurtle Procedure

InitTurtle initializes the TurtleGraphics unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE InitTurtle;
```

The current window is cleared. The turtle placed in the center of the window with an angle of 0 degrees (facing the right side of the screen).

To obtain the current window size, use the TurtleX and TurtleY functions and multiply each value by two.

An example of this procedure is:

```
var maxX,maxY: integer;
....
InitTurtle;
maxX := TurtleX * 2;
maxY := TurtleY * 2;
```

GrafMode Procedure

GrafMode sets the graphics mode. The definition of this procedure is:

```
PROCEDURE GrafMode;
```

The procedure does nothing in this implementation. It is included for compatibility with other implementations.

An example of this procedure is:

```
GrafMode;
```

TextMode Procedure

TextMode sets the text mode. The definition of this procedure is:

```
PROCEDURE TextMode;
```

The procedure does nothing in this implementation. It is included for compatibility with other implementations.

An example of this procedure is:

```
TextMode;
```


ViewPort Procedure

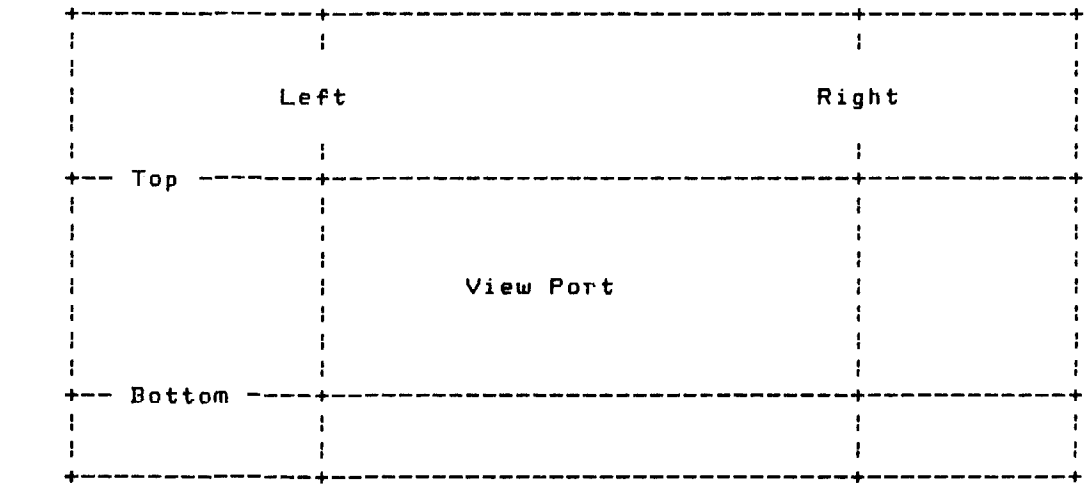
ViewPort sets the limits of the current plotting "window". As lines are drawn only line segments contained in the current view port are displayed. Line segments outside of the current view port are not displayed, ie, "clipped" at the edge of the view port. The definition of this procedure is:

```
PROCEDURE ViewPort (Left,Right,Bottom,Top: integer);
```

Parameter	Data Type	Description
Left	integer	View port left edge (X coord)
Right	integer	View port right edge (X coord)
Bottom	integer	View port bottom edge (Y coord)
Top	integer	View port top edge (Y coord)

The procedure sets the limits for the current view port. View port limits are relative to the current window. Initially, the view port limits describe the entire current window.

View Port Relationship to Current Window



An example of this procedure is:

```
ViewPort (100,200,200,300);
```

PenColor Procedure

PenColor sets the current pen color. The definition of this procedure is:

```
PROCEDURE PenColor (Color: ScreenColor);
```

Parameter	Data Type	Description
Color	ScreenColor	Pen color (none, white, black)

The procedure sets the pen color to the specified color. All lines are drawn (with Move or MoveTo) with this color. Color None is specified to not draw lines when moving the turtle.

An example of this procedure is:

```
PenColor (white);
```

FillScreen Procedure

FillScreen fills the current view port with the specified color. The definition of this procedure is:

```
PROCEDURE FillScreen (Color: ScreenColor);
```

Parameter	Data Type	Description
Color	ScreenColor	Background color

An example of this procedure is:

```
ViewPort (100,200,200,300);
FillScreen (white);
```

Turn Procedure

Turn rotates the turtle the specified number of degrees. The definition of this procedure is:

```
PROCEDURE Turn (Degrees: integer);
```

Parameter	Data Type	Description
Degrees	integer	Angle to turn

The procedure rotates the turtle counter clockwise if the angle is positive or clockwise if the angle is negative. The angle of rotation has a range of from -359 degrees to 359 degrees. Rotation is relative to the current turtle angle.

An example of this procedure is:

```
Turn (-90);
```

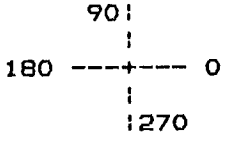
TurnTo Procedure

TurnTo turns the turtle to the specified heading. The definition of this procedure is:

```
PROCEDURE TurnTo (Degrees: integer);
```

Parameter	Data Type	Description
Degrees	integer	Absolute turtle heading

The procedure turns the turtle to the specified angle. Turtle headings are:



An example of this procedure is:

```
TurnTo (90);
```

Move Procedure

Move draws a line of the specified length from the current turtle position. The definition of this procedure is:

```
PROCEDURE Move (Distance: integer);
```

Parameter	Data Type	Description
Distance	integer	Length of line

The procedure moves the turtle a specified distance from its current position, drawing a line of the current pen color. The direction to move is defined by the current turtle angle. If the current pen color is None, the turtle is moved the specified distance with no line being drawn.

An example of this procedure is:

```
Move (100);
```

MoveTo Procedure

MoveTo draws a line from the current turtle position to the specified coordinates. The definition of this procedure is:

```
PROCEDURE MoveTo (NxtX,NxtY: integer);
```

Parameter	Data Type	Description
NxtX	integer	New absolute X coordinate
NxtY	integer	New absolute Y coordinate

The procedure moves the turtle from its current position to the specified absolute coordinates, drawing a line of the current pen color. If the current pen color is None, the turtle is placed at the specified coordinates with no line being drawn. The current turtle heading is not changed.

An example of this procedure is:

```
MoveTo (100,200);
```

TurtleX Function

TurtleX is used to ascertain the current turtle X coordinate. The definition of this function is:

```
FUNCTION TurtleX: integer;
```

The function returns the current turtle X coordinate. The coordinate is relative to the current window and is not related to the view port.

An example of this function is:

```
var CurX: integer;
....
curX := TurtleX;
writeln ('The current turtle X coordinate is ',CurX:1);
```

TurtleY Function

TurtleY is used to ascertain the current turtle Y coordinate. The definition of this function is:

```
FUNCTION TurtleY: integer;
```

The function returns the current turtle Y coordinate. The coordinate is relative to the current window and is not related to the view port.

An example of this function is:

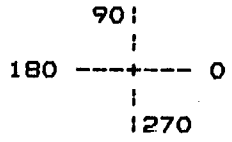
```
var CurY: integer;
....
curY := TurtleY;
writeln ('The current turtle Y coordinate is ',CurY:1);
```

TurtleAng Function

TurtleAng is used to find the current angle of the turtle. The definition of this function is:

```
FUNCTION TurtleAng: integer;
```

The function returns the current angle of the turtle in degrees. The angle has a range of from 0 to 359 degrees. Turtle headings are:



An example of this function is:

```
var CurAngle: integer;
....
CurAngle := TurtleAng;
writeln ('The current turtle angle is ',CurAngle:1,' degrees');
```

ScreenBit Function

ScreenBit tests the status of the pixel at the current turtle position. The definition of this function is:

```
FUNCTION ScreenBit: boolean;
```

The function returns the status of the pixel at the current turtle position. TRUE is returned if the pixel is on (white) or FALSE if the pixel is off (black).

An example of this function is:

```
MoveTo (100,200);
if ScreenBit then write ('Pixel  on at 100,200')
             else write ('Pixel off at 100,200');
```

Miscellaneous Functions and Procedures

CCLIB contains several assembly language functions and procedures. To use functions and procedures in this section, declare the function or procedure as EXTERNAL. The linker resolves the external declarations when CCLIB is linked with a program.

The functions and procedures assume no static link on the stack when called. This implies defining the external functions and procedures at the global level.

Miscellaneous Functions and Procedures

Miscellaneous procedures defined in CCLIB are:

Procedure	Description
xGetDir	Read volume directory
xPutDir	Write volume directory

Miscellaneous functions defined in CCLIB are:

Function	Description
OSactSlt	Get active slot
OSactSrv	Get active server
OSsltType	Get device type for slot
OSdevType	Get device type for given unit number
OSmaxDev	Get maximum device number

(continued on next page)

Function	Description
OSdcm1Dv	Get DTACOM1 driver device number
OSdcm2Dv	Get DTACOM2 driver device number
OSdispDv	Get DISPLAY driver device number
OSkybdDv	Get KYBD driver device number
OSomniDv	Get OMNINET driver device number
OSprtrDv	Get PRINTER driver device number
OSsltDv	Get SLOTIO driver device number
OSstrmDv	Get SYSTERM driver device number
OStimDv	Get TIMER driver device number
OSsysSize	Get system size
OScurSP	Get current system SP
OSvrtCrt	Returns TRUE if vertical orientation
pOScurKbd	Get current keyboard record pointer
pOScurWnd	Get current window record pointer
pOSsysWnd	Get system window record pointer
pOSdevNam	Get device name string pointer
pOSdate	Get system date pointer
pOSsysVol	Get system volume name string pointer
pOScurVol	Get current volume name string pointer
pOSsysVrs	Get OS version number string pointer
pOSsysDat	Get OS version date string pointer

(continued on next page)

Function	Description
KeyPress	Returns TRUE if any key is pressed
BrkPress	Returns TRUE if BREAK key is pressed
BitFlip	Change state of bit in integer
BitSet	Set bit in integer
BitClear	Clear bit in integer
BitTest	Test state of bit in integer
ShiftRt	Shift integer right one bit
ShiftLt	Shift integer left one bit
MakeByte	Convert integer to byte

xGetDir Procedure

xGetDir reads the directory of a volume. The definition of this procedure is:

```
PROCEDURE xGetDir (    VolID:       VID;
                   var VolDir:      directory;
                   var VolBlocked:  boolean;
                   var VolDevNo:    integer;
                   var VolValid:    boolean);        EXTERNAL;
```

Parameter	Data Type	Description
VolID	VID	Volume name
VolDir	directory	Volume directory
VolBlocked	boolean	Volume blocked flag
VolDevNo	integer	Volume unit number
VolValid	boolean	Volume directory valid flag

xPutDir Procedure

xPutDir writes a volume directory. Use of this procedure is NOT recommended. The definition of this procedure is:

```
PROCEDURE xPutDir (var VolDir:   directory;
                       VolDevNo: integer);          EXTERNAL;
```

Parameter	Data Type	Description
VolDir	directory	Volume directory
VolDevNo	integer	Volume unit number

OSactSlt Function

OSactSlt returns the active disk slot number. The definition of this function is:

```
FUNCTION OSactSlt: integer;                                    EXTERNAL;
```

OSactSrv Function

OSactSrv returns the active disk server number. The definition of this function is:

```
FUNCTION OSactSrv: integer;                                    EXTERNAL;
```

OSsltType Function

OSsltType returns the slot type of the specified slot. The definition of this function is:

```
FUNCTION OSsltType (slot: integer): slottype;                  EXTERNAL;
```

Parameter	Data Type	Description
Slot	integer	Slot number

OSdevType Function

OSdevType returns the slot type of the specified unit number. The definition of this function is:

```
FUNCTION OSdevType (Unt: integer): slottype;                   EXTERNAL;
```

Parameter	Data Type	Description
Unt	integer	Unit number

OSmaxDev Function

OSmaxDev returns the maximum device number. The definition of this function is:

```
FUNCTION OSmaxDev: integer;                    EXTERNAL;
```

OSdcm1Dv Function

OSdcm1Dv returns the device number of DTACOM1. The definition of this function is:

```
FUNCTION OSdcm1Dv: integer;                    EXTERNAL;
```

OSdcm2Dv Function

OSdcm2Dv returns the device number of DTACOM2. The definition of this function is:

```
FUNCTION OSdcm2Dv: integer;                    EXTERNAL;
```

OSdispDv Function

OSdispDv returns the device number of DISPLAY. The definition of this function is:

```
FUNCTION OSdispDv: integer;                    EXTERNAL;
```

OSkybdDv Function

OSkybdDv returns the device number of KYBD. The definition of this function is:

```
FUNCTION OSkybdDv: integer;                    EXTERNAL;
```


OSomniDv Function

OSomniDv returns the device number of OMNINET. The definition of this function is:

```
FUNCTION OSomniDv: integer;                    EXTERNAL;
```

OSprtrDv Function

OSprtrDv returns the device number of PRINTER. The definition of this function is:

```
FUNCTION OSprtrDv: integer;                    EXTERNAL;
```

OSsltDv Function

OSsltDv returns the device number of SLOTIO. The definition of this function is:

```
FUNCTION OSsltDv: integer;                     EXTERNAL;
```

OSstrmDv Function

OSstrmDv returns the device number of SYSTERM. The definition of this function is:

```
FUNCTION OSstrmDv: integer;                    EXTERNAL;
```

OStimDv Function

OStimDv returns the device number of TIMER. The definition of this function is:

```
FUNCTION OStimDv: integer;                     EXTERNAL;
```


OSsysSize Function

OSsysSize returns either 256 or 512 to indicate system memory size. The definition of this function is:

```
FUNCTION OSsysSize: integer;                    EXTERNAL;
```

OScurSP Function

OScurSP returns the current system stack pointer value. The definition of this function is:

```
FUNCTION OScurSP: longint;                      EXTERNAL;
```

OSvrtCrt Function

OSvrtCrt returns TRUE if the display is in the vertical orientation or FALSE if the display is in the horizontal orientation. The definition of this function is:

```
FUNCTION OSvrtCrt: boolean;                     EXTERNAL;
```

pOScurKbd Function

pOScurWnd returns a pointer to the current keyboard record. The definition of this function is:

```
FUNCTION pOScurKbd: pointer;                    EXTERNAL;
```

pOScurWnd Function

pOScurWnd returns a pointer to the current window record. The definition of this function is:

```
FUNCTION pOScurWnd: pointer;                    EXTERNAL;
```


pOSsysWnd Function

pOSsysWnd returns a pointer to the specified system window record. The definition of this function is:

```
FUNCTION pOSsysWnd (WndNbr: integer): pointer;      EXTERNAL;
```

Parameter	Data Type	Description
WndNbr	integer	System window number

pOSdevNam Function

pOSdevNam returns a pointer to the device name of the specified device. The definition of this function is:

```
FUNCTION pOSdevNam (UntNbr: integer): pointer;      EXTERNAL;
```

Parameter	Data Type	Description
UntNbr	integer	Unit number of device

pOSdate Function

pOSdate returns a pointer to the system date. The definition of this function is:

```
FUNCTION pOSdate: pointer;                          EXTERNAL;
```

pOSsysVol Function

pOSsysVol returns a pointer to the system volume name string. The definition of this function is:

```
FUNCTION pOSsysVol: pointer;                        EXTERNAL;
```


pOScurVol Function

pOScurVol returns a pointer to the current volume name string. The definition of this function is:

```
FUNCTION pOScurVol: pointer;                    EXTERNAL;
```

pOSsysVrs Function

pOSsysVrs returns a pointer to the OS version number string. The definition of this function is:

```
FUNCTION pOSsysVrs: pointer;                    EXTERNAL;
```

pOSsysDat Function

pOSsysDat returns a pointer to the OS version date string. The definition of this function is:

```
FUNCTION pOSsysDat: pointer;                    EXTERNAL;
```

KeyPress Function

KeyPress returns TRUE if any key is pressed and not yet read. The definition of this function is:

```
FUNCTION KeyPress: boolean;                     EXTERNAL;
```

BrkPress Function

BrkPress returns TRUE if the BREAK key has been pressed. The function clears the "BREAK key pressed" flag in the keyboard driver. The definition of this function is:

```
FUNCTION BrkPress: boolean;                     EXTERNAL;
```

BitFlip Function

BitFlip returns an integer with the specified bit changed. The definition of this function is:

```
FUNCTION BitFlip (Data,BitNum: integer): integer;   EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data
BitNum	integer	Bit number to change

BitSet Function

BitSet returns an integer with the specified bit set. The definition of this function is:

```
FUNCTION BitSet (Data,BitNum: integer): integer;    EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data
BitNum	integer	Bit number to set

BitClear Function

BitClear returns an integer with the specified bit clear. The definition of this function is:

```
FUNCTION BitClear (Data,BitNum: integer): integer;  EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data
BitNum	integer	Bit number to clear

BitTest Function

BitTest returns TRUE if the specified bit is set, FALSE if clear. The definition of this function is:

```
FUNCTION BitTest (Data,BitNum: integer): boolean;   EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data
BitNum	integer	Bit number to test

ShiftRt Function

ShiftRt returns an integer with the specified data shifted one bit to the right. The definition of this function is:

```
FUNCTION ShiftRt (Data: integer): integer;          EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data

ShiftLt Function

ShiftLt returns an integer with the specified data shifted one bit to the left. The definition of this function is:

```
FUNCTION ShiftLt (Data: integer): integer;          EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data

MakeByte Function

MakeByte returns a byte of data from the specified data. The definition of this function is:

```
FUNCTION MakeByte (Data: integer): byte;              EXTERNAL;
```

Parameter	Data Type	Description
Data	integer	Integer data

The Corvus Disk Interface Unit

ccDRVio

The Corvus Disk Interface Unit is used to interface with the Corvus disk controller. This unit is used by all of the Corvus utilities which communicate directly with the Corvus disk controller. It is used for both Omninet disks and local disks. It can access any slot and any server.

The ccDRVio unit USES units ccDEFN and ccLNGINT from CCLIB.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccLNGINT,
     {$U /CCUTIL/C2LIB} ccDRVio;
```

ccDRVio Unit Constants

Constants defined in ccDRVio are:

Identifier	Value	Description
DrvIOversion	n.n	Current unit version number
CDbuf_Max	1023	Corvus disk buffer length
DrvBlkSize	512	Disk block length
SndRcvMax	530	Send/receive string length
Low_Slot	1	Minimum slot number
High_Slot	5	Maximum slot number
Low_Server	0	Minimum server number
High_Server	63	Maximum server number

(continued on next page)

Identifier	Value	Description
MUX	64	Server number for MUX (High_Server + 1)
DrMax	7	Maximum number of drives on disk server or MUX

ccDRVio Unit Types

Data types defined in ccDRVio are:

Data Type	Description
SndRcvStr	Disk controller command string record

```
    sln: integer; {send length}
    rln: integer; {recv length}
    case integer of
        1: (c: packed array [1..SndRcvMax] of char);
        2: (b:        array [1..SndRcvMax] of byte);
```

CDaddr	Network address record

```
    SlotNo:     byte;       { Slot number                  }
    Kind:       SlotType;   { Type of interface in slot    }
    NetNo:      byte;       { Network number (UNUSED)      }
    StationNo:  byte;       { Omninet station address      }
    DriveNo:    byte;       { Disk drive number            }
    BlkNo:      LongInt;    { Disk block number            }
```

PhysDrInfo	Physical disk drive information record

```
    spt:        integer;    { Sectors/track                }
    tpc:        integer;    { Tracks/Sector                }
    cpd:        integer;    { Cylinders/Drive              }
    Capacity:   LongInt;    { Total nmbr of blocks         }
    DrSize:     DrSizes;    { Drive size                   }
    DrType:     DrRev;      { Drive controller revision    }
    PhysDr:     boolean;    { TRUE if a physical drive     }
    ROMvers:    integer;    { ROM version                  }
    FirmMsg:    string[8];  { Firmware message             }
    FirmVers:   integer;    { Firmware version number      }
```

Data Type	Description
DrvBlk	Disk block record
	case integer of 1: (c: packed array [1..DrvBlkSize] of char); 2: (b: array [1..DrvBlkSize] of byte);
CD_Buf	Disk command block record
	array [0..CDbuf_Max] of byte;
Valid_Slot	Valid slots
	Low_Slot..High_Slot;
Valid_Server	Valid disk servers
	Low_Server..High_Server;
PDrArray	Physical disk drive information table
	array [1..DrMax] of PhysDrInfo;
SprTrks	Spare tracks table
	array [1..DrMax] of integer;
Host_Type	Host device types (not currently used)
	User_Station = 0 ; User station
	File_Server = 1 ; File server
	Printer_Server = 2 ; Printer server
	Name_Server = 3 ; Name server
	Modem_Server = 4 ; Modem server
	DB_Server = 5 ; Data base server
	ON_Interconnect = 6 ; Omninet interconnection
	X25_Gateway = 7 ; X.25 gateway
	SNA_Gateway = 8 ; SNA gateway

Data Type	Description		
DrRev	Disk controller revision number		
	NoDrv	0	No controller
	RevA	1	Rev A controller
	RevB	2	Rev B controller
	RevH	3	Rev H controller
DrSizes	Disk drive size		
	OldTenMB	0	Rev A 10 megabyte disk
	FiveMB	1	5 or 6 megabyte disk
	TenMB	2	10 or 12 megabyte disk
	TwentyMB	3	18 or 20 megabyte disk
	FortyMB	4	.... reserved
	SixtyMB	5	.... reserved
	HundredMB	6	.... reserved

ccDRVio Unit Variables

Variables defined in ccDRVio are:

Variable	Data Type	Description
Spares	SprTrks	Spare tracks table

ccDRVio Unit Functions and Procedures

Procedures defined in ccDRVio are:

Procedure	Description
ccDRVioInit	Unit initialization
InitSlot	Initialize network address record
DrvInit	Get number of drives and physical disk information table
CDsend	Send disk command to controller
CDrecv	Receive data from controller

Functions defined in ccDRVio are:

Function	Description
CDslotInfo	Get slot type
CDbootInfo	Get boot slot number, type, and disk server number
CDslot	Verify that Corvus disk is in slot
CDserver	Verify disk server number
CDread	Read data from disk
CDwrite	Write data to disk

ccDRVioInit Procedure

ccDRVioInit initializes the ccDRVio unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccDRVioInit;
```

An example of this procedure is:

```
ccDRVioInit;
```

InitSlot Procedure

InitSlot initializes a network address record with the values for the boot slot and boot disk server. The definition of this procedure is:

```
PROCEDURE InitSlot (VAR NetLoc: CDaddr);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive

The procedure initializes the specified network address record with the following values:

```
SlotNo    - boot slot number
Kind      - boot slot type
StationNo - boot disk server number
NetNo     - 0
DriveNo   - 1
BlkNo     - 0
```

An example of this procedure is:

```
var curAddr: CDaddr;
....
InitSlot (curAddr);
```

DrvInit Procedure

DrvInit returns the number of drives and physical disk drive information for the specified disk drive controller. The definition of this procedure is:

```
PROCEDURE DrvInit (NetLoc: CDaddr;
                   VAR NumDrives: integer;
                   VAR PhysDrives: PDrArray);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive
NumDrives	integer	Number of drives on controller
PhysDrives	PDrArray	Physical disk drive info table

The procedure sets integer variable NumDrives to the number of disk drives at network address NetLoc. The physical disk drive information table, PhysDrives, is initialized for the number of drives on the disk controller.

An example of this procedure is:

```
var curAddr: CDaddr; curDrvs: integer; PDtable: PDrArray;
    i: integer;
....
DrvInit (curDrvs,PDtable);
for i := 1 to curDrvs do begin
    { .... do something for each drive on controller }
    end; {for}
```

CDsend Procedure

CDsend sends the specified disk command to the disk drive. The definition of this procedure is:

```
PROCEDURE CDsend (NetLoc: CDaddr; VAR DtaStr: SndRcvStr);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive
DtaStr	SndRcvStr	Disk command buffer

The procedure sends data in DtaStr to the disk drive at location NetLoc. DtaStr.sln bytes are sent to the disk drive. Normally, procedure CDrecv is executed directly after a CDsend in order to receive data back from the disk drive.

An example of this procedure is:

```
var curAddr: CDaddr; curCmd: SndRcvStr;
....
with curCmd do begin
    sln := nn;    { set send length       }
    rln := nn;    { set recv length       }
    { ...set disk controller command      }
    { ...move data to buffer, if needed   }
    CDsend (curAddr,curCmd);
    CDrecv (curAddr,curCmd);
    { ...check for successful operation   }
    end; {with curCmd do}
```

CDrecv Procedure

CDrecv receives disk data from the disk drive. The definition of this procedure is:

```
PROCEDURE CDrecv (NetLoc: CDaddr; VAR DtaStr: SndRcvStr);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive
DtaStr	SndRcvStr	Disk command buffer

The procedure receives data in DtaStr from the disk drive at location NetLoc. DtaStr.rln bytes are received from the disk drive. Normally, procedure CDrecv is executed directly after a CDsend in order to receive data back from the disk drive.

An example of this procedure is:

```
var curAddr: CDaddr; curCmd: SndRcvStr;
....
with curCmd do begin
    sln := nn;    { set send length      }
    rln := nn;    { set recv length      }
    { ...set disk controller command     }
    { ...move data to buffer, if needed }
    CDsend (curAddr,curCmd);
    CDrecv (curAddr,curCmd);
    { ...check for successful operation }
    end; {with curCmd do}
```


CDslotInfo Function

CDslotInfo returns the slot type for the specified slot number. The definition of this function is:

```
FUNCTION CDslotInfo (SlotNum: integer): SlotType;
```

Parameter	Data Type	Description
SlotNum	integer	Slot number

This function returns the slot type for the specified slot number. Slot types are:

Identifier	Value	Description
NoDisk	0	No disk
LocalDisk	1	Corvus local disk
OmninetDisk	2	Corvus Omninet disk server
FlpyC8Disk	3	Corvus 8" SSSD floppy disk
FlpyC5Disk	4	...reserved
FlpyA5Disk	5	Apple 5" floppy disk
BankDisk	6	...reserved
FlpyF8Disk	7	Corvus 8" DSDD floppy disk
FlpyF5Disk	8	Corvus 5" DSDD floppy disk
FlpyF3Disk	9	...reserved

An example of this function is:

```
var i: integer; Stype: SlotType;
....
for i := Low_Slot to High_Slot do begin
    Stype := CDslotInfo (i);
    write ('Slot ',n:1,' contains ');
    case Stype of
              NoDisk: writeln ('no disk');
           LocalDisk: writeln ('a Corvus local disk');
         OmninetDisk: writeln ('a Corvus Omninet disk');
          FlpyC8Disk: writeln ('a Corvus 8" SSSD floppy disk');
          FlpyC5Disk: writeln ('no disk');
          FlpyA5Disk: writeln ('an Apple 5" floppy disk');
          FlpyF8Disk: writeln ('a Corvus 8" DSDD floppy disk');
          FlpyF5Disk: writeln ('a Corvus 5" DSDD floppy disk');
          FlpyF3Disk: writeln ('no disk');
            BankDisk: writeln ('no disk');
                end; {case Stype of}
    end; {for}
```


CDbootInfo Function

CDbootInfo returns the boot slot number, boot disk server number, and the boot slot type. The definition of this function is:

```
FUNCTION CDbootInfo (VAR SlotNum,SrvrNum: integer): SlotType;
```

Parameter	Data Type	Description
SlotNum	integer	Slot number
SrvrNum	integer	Disk server number

This function returns the boot slot type. Integer variable SlotNum is set to the boot slot number. Integer variable SrvrNum is set to the boot disk server number. Boot slot types are:

Identifier	Value	Description
NoDisk	0	No disk
LocalDisk	1	Corvus local disk
OmninetDisk	2	Corvus Omninet disk server
FlpyC8Disk	3	Corvus 8" SSSD floppy disk
FlpyC5Disk	4	...reserved
FlpyA5Disk	5	Apple 5" floppy disk
BankDisk	6	...reserved
FlpyF8Disk	7	Corvus 8" DSDD floppy disk
FlpyF5Disk	8	Corvus 5" DSDD floppy disk
FlpyF3Disk	9	...reserved

An example of this function is:

```
var Bslot,Bsrvr: integer; Btype: SlotType;
....
Btype := CDbootInfo (Bslot,Bsrvr);
```

CDslot Function

CDslot returns a boolean value indicating if the specified slot has access to a Corvus disk. The definition of this function is:

```
FUNCTION CDslot (SlotNum: integer): boolean;
```

Parameter	Data Type	Description
SlotNum	integer	Slot number

This function returns TRUE if the specifed slot contains a local disk drive or an Omninet disk drive. FALSE is returned neither is present in the slot.

An example of this function is:

```
if CDslot (5) then writeln ('Omninet disk available')
              else writeln ('Omninet disk not available');
```

CDserver Function

CDserver return a boolean value indicating if the specified server is valid. The definition of this function is:

```
FUNCTION CDserver (Server: integer): boolean;
```

Parameter	Data Type	Description
Server	integer	Disk server number

This function returns TRUE if the specifed server is valid. FALSE is returned if the specified server is not valid. Currently, this function is not operational.

CDread Function

CDread reads data from the disk drive. The definition of this function is:

```
FUNCTION CDread (NetLoc: CDaddr;
                 VAR Buf: CD_Buf; Len: integer): integer;
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive
Buf	CD_Buf	Data that is read
Len	integer	Number of bytes to read

This function returns the disk status code. NetLoc contains the disk drive number and starting block number along with the other network address information. Len bytes of data is placed in Buf.

An example of this function is:

```
var IOst: integer; curAddr: NetLoc; curBuff: CD_Buf;
....
IOst := CDread (curAddr,curBuff,512);
```

CDwrite Function

CDwrite writes data to the disk. The definition of this function is:

```
FUNCTION CDwrite (NetLoc: CDaddr;
                  VAR Buf: CD_Buf; Len: integer): integer;
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network address of disk drive
Buf	CD_Buf	Data to be written
Len	integer	Number of bytes to write

This function returns the disk status code. NetLoc contains the disk drive number and starting block number along with the other network address information. Len bytes of data is written to the disk from Buf.

An example of this function is:

```
var IOst: integer; curAddr: NetLoc; curBuff: CD_Buf;
....
IOst := CDwrite (curAddr,curBuff,512);
```


The Corvus Disk Pipes Interface Unit

ccPIPES

The Corvus Disk Pipes Interface Unit is used to interface with the Corvus disk controller pipe functions.

The ccPIPES unit USES units ccDEFN and ccLNGINT from CCLIB. It also USES unit ccDRVio from C2LIB.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN, ccLNGINT,
     {$U /CCUTIL/C2LIB} ccDRVio, ccPIPES;
```

ccPIPES Unit Constants

Constants defined in ccPIPES are:

Pipe Command Status Codes

Identifier	Value	Description
PipeOk	0	Successful pipes command
PipeEmpty	-8	Tried to read an empty pipe
PipeNotOpen	-9	Pipe not open for read or write
PipeFull	-10	Tried to write to a full pipe
PipeOpErr	-11	Tried to open an open pipe
PipeNotThere	-12	Pipe does not exist
PipeNoRoom	-13	Pipe data structures are full
PipeBadCmd	-14	Invalid pipes command

(continued on next page)

Pipe Command Return Codes (continued)

Identifier	Value	Description
PipesNotInitted	-15	Pipes area not initialized
PipeDskErr	-255	
An error code less than -127 is a fatal disk error		

Identifier	Value	Description
PipesVersion	n.n	Current unit version number
PnameLen	8	Pipe name length

ccPIPES Unit Types

Data types defined in ccPIPES are:

Data Type	Description
PNameStr	Pipe name string
string[PnameLen];	

ccPIPES Unit Variables

Variables defined in ccPIPES are:

Variable	Data Type	Description
PipeDebug	boolean	Pipes debug switch

ccPIPES Unit Functions and Procedures

Procedures defined in ccPIPES are:

Procedure	Description
ccPIPEinit	Unit initialization

Functions defined in ccPIPES are:

Function	Description
PipeStatus	Get status of pipes area
PipeOpRd	Open pipe for reading
PipeOpWr	Open pipe for writing
PipeRead	Read data from pipe
PipeWrite	Write data to pipe
PipeClRd	Close pipe for reading
PipeClWr	Close pipe for writing
PipePurge	Purge pipe
PipesInit	Initialize pipes area on disk

ccPIPEinit Procedure

ccPIPEinit initializes the ccPIPES unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccPIPEinit (NetLoc: CDaddr);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network location record

An example of this procedure is:

```
var PnetLoc: CDaddr;
....
InitSlot (PnetLoc);
....
ccPIPEinit (PnetLoc);
```

PipeStatus Function

PipeStatus reads the pipe name table and pipe pointer table from disk. The definition of this function is:

```
FUNCTION PipeStatus (VAR Names,Ptrs: DrvBlk): integer;
```

Parameter	Data Type	Description
Names	DrvBlk	Pipe name table
Ptrs	DrvBlk	Pipe pointer table

This function returns the status of the pipe command.

An example of this function is:

```
var Pstat: integer; Pnames,Pptrs: DrvBlk;
....
Pstat := PipeStatus (Pnames,Pptrs);
```

Format of the Pipe name table is 8 bytes per Pipe (64 Pipes). The first name is WOOFWOOF and the last name is FOOWFOOW.

The Pipe pointer table has 64 entries, each 8 bytes long with the following format:

One byte Pipe number
Three bytes of starting (512 byte) block number
Three bytes of ending (512 byte) block number
One byte of Pipe status code

Pipe status codes are:

Dec	Hex	Description
1	01	Open for write, Pipe empty
2	02	Open for read, Pipe empty
128	80	Closed
129	81	Open for write
130	82	Open for read

PipeOpRd Function

PipeOpRd open a pipe for reading. The definition of this function is:

```
FUNCTION PipeOpRd (PName: PNameStr): integer;
```

Parameter	Data Type	Description
PName	PNameStr	Pipe name to open

This function returns the pipe number if the specified pipe exists and is not already open. Otherwise, a negative error code is returned.

An example of this function is:

```
var Pnmbr: integer, Pname: PNameStr;
....
Pname := 'KLLPIPE';
Pnmbr := PipeOpRd (Pname);
if Pnmbr > 0
    then writeln ('Pipe ',Pname,' [',Pnmbr:1,'] opened')
    else writeln ('Unable to open pipe ',Pname);
```

PipeOpWr Function

PipeOpWr opens a pipe for writing, assigns the pipe a name, and assigns a number to the pipe. The definition of this function is:

```
FUNCTION PipeOpWr (PName: PNameStr): integer;
```

Parameter	Data Type	Description
PName	PNameStr	Pipe name to open

This function returns the assigned pipe number if successful. Otherwise, a negative error code is returned.

An example of this function is:

```
var Pnmbr: integer; Pname: PNameStr;
....
Pname := 'KLLPIPE';
Pnmbr := PipeOpWr (Pname);
if Pnmbr > 0
    then writeln ('Pipe ',Pname,' [',Pnmbr:1,'] opened')
    else writeln ('Unable to open pipe ',Pname);
```

PipeRead Function

PipeRead reads a block of data from the specified pipe. The definition of this function is:

```
FUNCTION PipeRead (NPipe: integer; VAR Info: DrvBlk): integer;
```

Parameter	Data Type	Description
NPipe	integer	Pipe number for read
Info	DrvBlk	Data buffer for read

This function returns the number of bytes written if the read is successful. Otherwise, a negative error code is returned. PipeRead is repeated for each block to be read from the pipe.

An example of this function is:

```
var Pstat,Pnmbr: integer; Pdata: DrvBlk;
....
Pstat := PipeRead (Pnmbr,Pdata);
```

PipeWrite Function

PipeWrite writes a block of data to the specified pipe. The definition of this function is:

```
FUNCTION PipeWrite (NPipe,WLen: integer;
                    VAR Info: DrvBlk): integer;
```

Parameter	Data Type	Description
NPipe	integer	Pipe number for write
WLen	integer	Length of data to write
Info	DrvBlk	Data buffer for write

This function returns the number of bytes written if the write is successful. Otherwise, a negative error code is returned. PipeWrite is repeated for each block to be written to the pipe.

An example of this function is:

```
var Pstat,Pnmbr: integer; Pdata: DrvBlk;
....
Pstat := PipeWrite (Pnmbr,512,Pdata);
```

PipeClRd Function

PipeClRd closes the specified pipe for reading. The definition of this function is:

```
FUNCTION PipeClRd (NPipe: integer): integer;
```

Parameter	Data Type	Description
NPipe	integer	Pipe number to close

This function returns the status of the pipe command. If the pipe is empty, the pipe is deleted.

An example of this function is:

```
var Pstat,Pnmbr: integer;
....
Pstat := PipeClRd (Pnmbr);
```

PipeClWr Function

PipeClWr closes the specified pipe for writing. The definition of this function is:

```
FUNCTION PipeClWr (NPipe: integer): integer;
```

Parameter	Data Type	Description
NPipe	integer	Pipe number to close

This function returns the status of the pipe command. Once a pipe has been closed for writing, no additional data can be written to it.

An example of this function is:

```
var Pstat,Pnmbr: integer;
....
Pstat := PipeClWr (Pnmbr);
```

PipePurge Function

PipePurge purges the specified pipe. The definition of this function is:

```
FUNCTION PipePurge (NPipe: integer): integer;
```

Parameter	Data Type	Description
NPipe	integer	pipe number to purge

This function returns the status of the pipe command.

An example of this function is:

```
var Pstat,Pnmbr: integer;
....
Pstat := PipePurge (Pnmbr);
```

PipesInit Function

PipesInit initializes the pipes data structures on the disk. The definition of this function is:

```
FUNCTION PipesInit (Baddr,Bsize: LongInt): integer;
```

Parameter	Data Type	Description
Baddr	LongInt	Pipes area base block number
Bsize	LongInt	Pipes area number of blocks

This function returns the status of the pipe command.

An example of this function is:

```
var Pstat: integer; Paddr,Psize: LongInt;
....
Paddr := 10000;
Psize := 1024;
Pstat := PipesInit (Paddr,Psize);
```

The Corvus Disk Semaphores Interface Unit

ccSEMA4

The Corvus Disk Semaphores Interface Unit is used to interface with the Corvus disk controller semaphore functions.

The ccSEMA4 unit USES unit ccDEFN from CCLIB. It also USES unit ccDRVio from C2LIB.

The unit is included in user software by declaring:

```
USES {$U /CCUTIL/CCLIB} ccDEFN,
     {$U /CCUTIL/C2LIB} ccDRVio, ccSEMA4;
```

ccSEMA4 Unit Constants

Constants defined in ccSEMA4 are:

Semaphore Command Status Codes

Identifier	Value	Description
Sema4Rev	n.n	Current unit version number
SemWasSet	128	Prior state was locked
SemNotSet	0	Prior state was unlocked
SemFull	253	Semaphore table is full (32 active semaphores)
SemDskErr	-255	Disk error during write thru
An error code less than -127 is a fatal disk error		

ccSEMA4 Unit Types

Data types defined in ccSEMA4 are:

```
+============+=============================================+
| Data Type  | Description                                 |
+============+=============================================+
| SemStr     | Semaphore name string                       |
+---+--------+---------------------------------------------+
    | string[8];                                           |
+---+--------+---------------------------------------------+
| SemKeys    | Semaphore key                               |
+---+--------+---------------------------------------------+
    | packed array [1..8] of char;                         |
+---+--------+---------------------------------------------+
| SemKeyList | Semaphore key array record                  |
+---+--------+---------------------------------------------+
    | case integer of                                      |
    |     1: (skey: array [1..32]  of SemKeys);            |
    |     2: (sbyt: array [1..256] of byte);               |
    +------------------------------------------------------+
```

ccSEMA4 Unit Variables

Variables defined in ccSEMA4 are:

```
+============+============+================================+
| Variable   | Data Type  | Description                    |
+============+============+================================+
| Sema4debug | boolean    | Semaphores debug switch        |
+------------+------------+--------------------------------+
```


ccSEMA4 Unit Functions and Procedures

Procedures defined in ccSEMA4 are:

Procedure	Description
ccSEMA4init	Unit initialization

Functions defined in ccSEMA4 are:

Function	Description
SemLock	Lock semaphore
SemUnlock	Unlock semaphore
SemClear	Clear all semaphores
SemStatus	Get semaphore status

ccSEMA4init Procedure

ccSEMA4init initializes the ccSEMA4 unit. This procedure must be called before any other functions or procedures in this unit are called. The definition of this procedure is:

```
PROCEDURE ccSEMA4init (NetLoc: CDaddr);
```

Parameter	Data Type	Description
NetLoc	CDaddr	Network location record

An example of this procedure is:

```
var SnetLoc: CDaddr;
....
InitSlot (SnetLoc);
....
ccSEMA4init (SnetLoc);
```


SemLock Function

SemLock locks the specified semaphore. The definition of this function is:

```
FUNCTION SemLock (Key: SemStr): integer;
```

Parameter	Data Type	Description
Key	SemStr	Semaphore name to lock

This function returns one of the following codes:

Identifier	Value	Description
SemWasSet	128	Semaphore already locked
SemNotSet	0	Semaphore successfully locked
SemFull	253	Semaphore table is full, semaphore not locked
SemDskErr	-255	Disk error
An error code less than -127 is a fatal disk error		

An example of this function is:

```
var Sstat: integer; Sname: SemStr;
....
Sname := 'KLL';
Sstat := SemLock (Sname);
case Sstat of
    SemWasSet: writeln ('Semaphore already locked');
    SemNotSet: writeln ('Semaphore successfully locked');
      SemFull: writeln ('Semaphore table is full');
    otherwise: writeln ('Disk error');
          end; {case}
```


SemUnlock Function

SemUnlock unlocks the specified semaphore. The definition of this function is:

```
FUNCTION SemUnlock (Key: SemStr): integer;
```

Parameter	Data Type	Description
Key	SemStr	Semaphore name to unlock

This function returns one of the following codes:

Identifier	Value	Description
SemWasSet	128	Semaphore successfully unlocked
SemNotSet	0	Semaphore was not locked
SemDskErr	-255	Disk error
An error code less than -127 is a fatal disk error		

An example of this function is:

```
var Sstat: integer; Sname: SemStr;
.....
Sname := 'KILL';
Sstat := SemUnlock (Sname);
case Sstat of
    SemWasSet: writeln ('Semaphore successfully unlocked');
    SemNotSet: writeln ('Semaphore was not locked');
    otherwise: writeln ('Disk error');
          end; {case}
```


SemClear Function

SemClear clears the semaphore table.

```
FUNCTION SemClear: integer;
```

This function returns one of the following codes:

Identifier	Value	Description
....	0	Semaphore table cleared
An error code less than -127 is a fatal disk error		

An example of this function is:

```
var Sstat: integer;
....
Sstat := SemClear;
if Sstat = 0
    then writeln ('Semaphore table cleared')
    else writeln ('Semaphore table clear failed');
```


SemStatus Function

SemStatus returns the names of locked semaphores. The definition of this function is:

```
FUNCTION SemStatus (VAR KeyBuf: SemKeyList): integer;
```

Parameter	Data Type	Description
KeyBuf	SemKeyList	Semaphore name table

This function returns one of the following codes:

Identifier	Value	Description
....	0	Semaphore table read successful
....	-2	Unable to enter PREP mode (could not find DIAG.DATA)
An error code less than -127 is a fatal disk error		

An example of this function is:

```
var Sstat: integer; Skeys: SemKeyList;
....
Sstat := SemStatus (Skeys);
```


Corvus Concept Pascal System Library Index

Corvus Concept Pascal System Library

READER'S COMMENTS

By responding with your comments, Corvus Systems can develop better documentation to meet your needs.

Did you find errors in this manual? If so, please specify by page.

_____
_____
_____
_____

Did you find this manual understandable, useable, and well-organized? Please make suggestions for improvements.

_____
_____
_____
_____

Is there sufficient documentation for use of the products described in this manual. If not, what material is missing and where should it be placed?

_____
_____
_____
_____

Please indicate the type of user/reader that you most nearly represent.

___ Software engineer
___ Hardware engineer
___ System integration
___ Engineering management
___ Non-technical
___ _____

Name _____ Date _____
Organization _____
Street _____
City _____ State _____ Zip _____

Please return this form to: Corvus CONCEPT Support Group
Corvus Systems, Inc.
2029 O'Toole Avenue
San Jose, California 95131