NAME

atotim - Convert an ASCII time string to seconds.

SYNOPSIS

```
atotim (tptr,tsec)
     char *tpti;
     long *tsec;
```

DESCRIPTION

This routine takes an ASCII representation of time with format "mmddhhnnyy" from tptr and converts it to seconds since 00:00:00 GMT, Jan. 1, 1970 and stores it in tsec.

```
mm - 2 digit month    01 to 12
dd - 2 digit day      01 to 31
hh - 2 digit hour     00 to 23
nn - 2 digit minutes  00 to 59
yy - 2 digit year.
```

If successful a zero is returned, otherwise, -1 is returned.

LIBRARY

/lib/lib1.a

SEE ALSO

time(2),ctime(3),timtoa(3L)

DIAGNOSTICS

A -1 is returned if an error is found.

NAME

binary -- convert to binary

SYNOPSIS

```
binary(radix)
int radix;
```

DESCRIPTION

This subroutine converts data consisting of ASCII characters to its binary value and stores the binary value in a two-word external array, WORD. WORD[1] contains the low-order bits. The address of WORD is returned to the calling program, unless an error is encountered. In this case, appropriate error information is returned in the external variables, E_SPCL, E_TYPE, E_CODE, E_NUM, and E_MSG, and a 0 is returned by this subroutine.

BINARY has one argument, radix, which specifies the radix or base of the data to be converted from ASCII to binary. A radix of 32 uses base 16 but ASCII conversion is done for a 101 ESS. The data to be converted is passed to this subroutine via the external variable, VALSTR.

The global variables used are:

```
char *E_SPCL;
char *E_TYPE;
char *E_CODE;
char *E_NUM;
char *E_MSG;
char VALSTR[33];
int  WORD[2];
```

The error information returned is:

```
E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "002";
E_MSG=   "INVALID BASE.";
```

LIBRARY

/lib/lib1.a

SEE ALSO

dp_add(3), dp_mul(3), binasc(3)

DIAGNOSTICS

A 0 is returned if @A 0 is returned if radix is not between one and eleven, or sixteen, or thirty-two

NAME

banner - prepare the requested banner and copy to the indicated buffer.

SYNOPSIS

```
#include <banner.h>

banner(bufnam, lnlen, charsiz, banstr)
char *bufnam;
int lnlen;
int charsiz;
char *banstr;
```

DESCRIPTION

Banner prepares the requested banner string, centers it in a line of the requested length, and copies it to the specified output buffer. Line lengths up to LL_132 (132 character line) are supported and two character sizes are supported by this subroutine. The supported character sizes are DOT55 (5x5 character matrices) and DOT59 (5x9 character matrices).

The character size DOT55 only supports the upper case alphabetic characters A - Z and the numerals 0 - 9. Lower case alphabetic characters are mapped to the corresponding upper case character, while all other characters except spaces and newlines are ignored. The character size DOT59 supports all ASCII characters between space (040) and ~ (0176); newlines are also supported.

If an error is detected, banner returns the value ERR_RTN; otherwise, the value NORM_RTN is returned.

The argument bufnam is the address of a character output buffer into which the banner is to be copied. Banner assumes that this buffer is large enough to hold the entire banner. The user should be aware that each banner character requires eight bytes per line (banner character plus surrounding white space) and either five or nine lines depending upon the character size selected. These things should be taken into consideration when determining the size of the output buffer.

The argument lnlen identifies the maximum length of the printed line. Line lengths greater than zero and less than or equal to LL_132 are permitted.

The argument charsiz identifies the character size that is to be generated. Two character sizes are presently supported, DOT55 and DOT59.

The argument banstr is the address of a string containing the banner that is to be generated. This string may contain newlines. Examples of acceptable banner strings are:

```
"abc"
"\n\nabc\n\n"
"\n\nThis is a banner string\n\n"
"\n\nSPA\n\nANALYSIS\n\nREPORT\n\n"
```

FILES

/usr/include/banner.h which contains the define variables DOT55, DOT59, LL_80, LL_132, ERR_RTN, and NORM_RTN.

LIBRARY

/lib/libl.a

SEE ALSO

e_output(3L)

DIAGNOSTICS

If this subroutine detects an error, an Output Message (OM) is generated by one of the standard OM generation subroutines, but not printed. The value ERR_RTN is returned to the calling routine. If the calling routine wishes to print the stored OM, it may call one of the standard OM outputting subroutines, such as e_output(3L).

BUGS

NAME

binasc -- binary to ascii conversion

SYNOPSIS

```
binasc(radix)
int radix;
```

DESCRIPTION

This subroutine converts a binary number contained in the external variable, WORD, to an ASCII string. The ASCII string is terminated with a null byte and stored right-justified in the external variable, STRING. The starting address of the ASCII string within STRING is returned by this routine, unless an error is detected. In this case, appropriate error information is returned in the external variables, E_SPCL, E_TYPE, E_CODE, E_NUM, and E_MSG, and a 0 is returned by this subroutine.

BINASC has one argument, radix, which specifies the radix or base to which the binary number is to be converted. Radix must be a number between one and eleven, sixteen or thirty-two (thirty-two means use a radix of 16 for a 101 ESS).

The global variables used are:

```
char *E_SPCL;
char *E_TYPE;
char *E_CODE;
char *E_NUM;
char *E_MSG;
char STRING[33];
int  WORD[2];
```

The error information returned is:

```
E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "002";
E_MSG=   "INVALID BASE.";
```

LIBRARY

/lib/lib1.a

SEE ALSO

binary(3)

DIAGNOSTICS

A 0 is returned if radix is not between one and eleven, or sixteen, or thirty-two.

NAME

bit -- bit extraction

SYNOPSIS

```
bit(hi,lo)
int hi, lo;
```

DESCRIPTION

This subroutine returns a binary number that is the bit extraction of a specified field within a 32-bit binary number contained in the external variable, WORD. The bits in WORD are numbered 0 to 31 going from right to left.

This subroutine has two arguments, hi and lo, which specify the field to be extracted. Hi is the number of the left-most bit to be extracted, while lo is the number of the right-most bit to be extracted. Hi minus lo must be less than sixteen.

The global variables used are:

```
int  WORD[2];
```

LIBRARY

/lib/lib1.a

SEE ALSO

lbit(3L)

BUGS

No checks are made on the reasonableness of the hi and lo and no error conditions are returned. If (hi - lo) is greater than 15, then 16 bits are returned; if (hi - lo) is less than 0, then 0 is returned.

NAME

bopnclos - Buffered open and close.

SYNOPSIS

```
bopnclos (filename,mode)        /*Open file with given mode.*/
     char *filename;
     int mode;

bopnclos (fildes,-1)            /*File descriptor may be closed
     int fildes;                   if needed.*/

bopnclos (fildes,-2)            /*File descriptor may be truly
     int fildes;                   closed if previously marked as
                                   not needed.*/

bopnclos (-2,-2)                /*All file descriptors not needed
                                  should be closed.*/
```

DESCRIPTION

NOTE: When dealing with new programs consider Standard I/O first.

This routine maintains a list of up to 15 open files. If the external variable BOCLOS_MAX is set to a number between 1 and 15 then BOCLOS_MAX descriptors will be kept open. Otherwise 10 descriptors are kept open.

To "open" a file a bopnclos (filename,mode) call is made. If a file is already opened with the name exactly matching filename and with a matching mode the descriptor is returned and the file is marked in use. Otherwise, if the routine has not used up all the descriptors allowed to it by BOCLOS_MAX the file is truly opened and its name and mode recorded. If the file is not already opened and there are no spare descriptors then a file "closed" by bopnclos (fildes,-1) is truly closed to free up the descriptor. Of all the "closed" files the one least recently "opened" is chosen.

When the routine is called with bopnclos (fildes,-1) to "close" a file, its entry is found in the local table and it is marked as available for closing if its descriptor is needed.

When the routine is called with bopnclos (fildes,-2) the fildes, if it is marked available, is truly closed. Otherwise a -1 is returned.

Bopnclos (-2,-2) forces a close of all the file descriptors marked available. If no file was marked available a -1 is returned and errno is clear. Otherwise the logical or of all the returns for the close system calls is returned and errno is as the system leaves it. This is useful when you know you no longer need those files or when you unlink the original file and want to open another file by the same name.

This routine can be used to save disk accesses for opening and closing files frequently accessed by a running process. Note that it must be used with extreme caution since it can cause the user to run out of descriptors if proper care was not exercised. Its use should also be coordinated with the system gnomes since it potentially makes use of a large percentage of the total number of descriptors that the system may have open at one time.

LIBRARY

/lib/lib1.a

SEE ALSO

bread(3L),bwrite(3L),open(2),close(2),fopen(3)

DIAGNOSTICS

A -1 is returned when asked to truly close a file descriptor not marked available for closing or when a file name of more than 29 characters is passed to it. Otherwise the return of the corresponding system call (open or close) is returned.

BUGS

File names of more than 29 characters do not fit in its internal buffers.

NAME

bread - Buffered reads.

SYNOPSIS

```
#include <bread.h>

bread (brbuf,ubuf,n)
     char *ubuf;            /*Buffer where user wants to read into.*/
     struct BREAD *brbuf;   /*Buffer used by bread.*/
     int n;                 /*Number of bytes to be read.*/

bropen (filename,brbuf,size)
     char *filename;        /*File to be opened.*/
     struct BREAD *brbuf;   /*Buffer set up by bropen.*/
     int size;              /*Size of area in brbuf actually
                             used for character buffering.*/

brsetup (brbuf,fdes,size)
     struct BREAD *brbuf;   /*Buffer to be set up.*/
     int fdes;              /*File descriptor of file to be
                             buffer read.*/
     int size;              /*Size of area in brbuf actually
                             used for character buffering.*/

brlseek (brbuf,offset,ptrname)
     struct BREAD *brbuf;   /*Buffer describing file on which
                             seek is to be done.*/
     long offset;           /*Same as in lseek sys call.*/
     int ptrname;           /*Same as in lseek sys call.*/

long brtell (brbuf)
     struct BREAD *brbuf;

brclose (brbuf)
     struct BREAD *brbuf;
```

DESCRIPTION

NOTE: When dealing with new programs consider Standard I/O first.

Bread performs buffered reads on the file described by brbuf. Brbuf has the following format:

```
struct BREAD{                       /*See bread.h file*/
 int br_fildes;                     Read descriptor of file.
 char *br_next;                     Next valid character in buffer.
 char *br_last;                     Last valid character in buffer.
 int br_bufsize;                    Size of br_buffer passed by
                                    user in bytes.
 char br_buffer[BR_BUFFER_SIZ];     Actual character buffer.
                                    (size defined by user)
```

};
When bread is called, if there are less than n characters in br_buffer, these characters are passed to the user then a read of a buffer full is done. If the read is not successful the characters passed to the user are retrieved back to br_buffer and -1 is returned. This is useful when a read (say from a pipe) may be interrupted by a signal. In order to do the retrieval n is restricted to be less than the size of br_buffer as specified by the user. If it is not, a -2 is returned. If successful, bread returns the number of characters actually passed to the user's buffer. This may be less than n if an end of file is reached.

Bropen opens filename for reading and saves its descriptor in brbuf. It also saves the size of the area in brbuf actually used for buffered characters. This allows the user to specify the size most suitable for the application (usually 512). Other variables in brbuf are set up properly for use with bread. Bropen returns the return from the open system call.

Brsetup sets up. Brbuf the same as bropen but instead of opening the file it gets passed the descriptor of a file that is already opened for reading or reading and writing. It returns the descriptor.

Brlseek permits the user to do a lseek on a file that is being read with bread. It discards any characters that may be buffered in br_buffer and then does the requested seek. Note that the seek is done to the actual place requested and therefore some efficiency may be lost when seeking on a disk file to a non multiple of 512 when the size of br_buffer has been specified to be 512; when the next bread occurs 512 bytes will be read into br_buffer but they will overlap two physical blocks and therefore causes two disk accesses. Brlseek returns the result of the lseek.

Brtell returns to the user the position in the file described by brbuf of the next character that can be read using bread. The actual character may be still in the file (if br_buffer is empty) or in br_buffer. A -1 is returned when the tell system call returns -1.

Brclose closes the file described by descriptor in br_fildes, loads -1 into br_fildes and discards any characters in br_buffer. It returns the result of the close system call.

FILES

/usr/include/bread.h

LIBRARY

/lib/lib1.a

SEE ALSO

open(2),close(2),lseek(2),bopnclos(3L),bwrite(3L),fread(3)

DIAGNOSTICS

bread - returns -1 if read sys call returns -1.
returns -2 if trying to read more than the number of characters that fit into br_buffer.
bropen - returns result of open sys call.
brlseek - returns result of lseek sys call.
brtell - returns -1 if tell sys call fails.
brclose - returns result of close sys call.

NAME

breaks -- look for first char in pattern

SYNOPSIS

```
breaks(s1,s2)
char *s1, *s2;
```

DESCRIPTION

breaks returns an integer indicating the success or failure of the pattern match. If the value returned is an array index the match was a success. If the value returned is -1 the match was a failure. This function indicates success if a character in the pattern string is found in the searched string.

s1 the searched character string.

s2 a string of characters used as a pattern.

The pattern, s2, can be any null terminated string of characters. Repeated characters in s2 are ignored. The pattern string "Mississippi" is equivalent to the pattern string "iMps".

This function is implemented with a table driven pattern matcher. The empty string is defined as a string whose first character is the null character.

If either or both of the strings is empty the value returned will be -1.

If a character of the string s2 is found in the string s1, the array index of the character position in s1 will be returned.

If the entire string s1 is searched and no character in s2 is found in s1, the value returned will be -1.

LIBRARY

/lib/lib3.a

NAME

bwrite - Buffered writes.

SYNOPSIS

```
#include <bwrite.h>

bwrite (bwbuf,ubuf,n)
     struct BWRITE *bwbuf;   /*Buffer maintained by bwrite.*/
     char *ubuf;             /*Pointer to point byte user
                               wants to write.*/
     int n;                  /*Number of bytes to be written.*/

bwopen (filename,bwbuf,size)
     char *filenm;           /*File to be opened.*/
     struct BWRITE *bwbuf;   /*Buffer set up by bwopen.*/
     int size;               /*Size of area in bwbuf actually
                               used for character buffering.*/

bwsetup (bwbuf,outdes,size)
     struct BWRITE *bwbuf;   /*Buffer to be used by bwrite.*/
     int outdes;             /*Descriptor of file to be written.*/
     int size;               /*Size of area in bwbuf actually
                               used for character buffering.*/

bwflush (bwbuf)
     struct BWRITE *bwbuf;   /*Buffer in use by bwrite.*/

bwclose (bwbuf)
     struct BWRITE *bwbuf;   /*Buffer used by bwrite.*/
```

DESCRIPTION

NOTE: When dealing with new programs consider standard I/O first.

Bwrite performs buffered writes on the file described by bwbuf. Bwbuf has the following format:

```
struct BWRITE{                  /*See bwrite.h file*/
 int bw_des;                     Write descriptor of file.
 char *bw_nxtc;                  Position at which next character
                                 will be stored.
 char *bw_lstc;                  Points to end of bw_buf.
 char  bw_buf[BW_BUFFER_SIZ];    Actual character buffer
                                 (defined by user).

};
```

When bwrite is called it copies n characters from ubuf to the appropriate location in bw_buf one by one. If bw_buf is filled at any time bwrite writes out a buffer full to the file specified by bw_des, resets the internal buffer pointer and continues copying characters from ubuf to bw_buf. If successful bwrite returns n. If on coming in it finds an obviously wrong bwbuf it clears errno (see INTRO 2) and returns a -1. If an attempted write of a

buffer full fails a -1 is returned and errno is as left by the write system call. Note that if a write fails it is not obvious to the user what data got actually written and what data is still in bw_buf. Also note that a bwflush or bwclose must always be done at the end of all the bwrites for a given bwbuf.

Bwopen opens filename for writing and saves its descriptor in bwbuf. It also saves in bwbuf the size of the area actually used for buffered characters. This allows the user to specify the size most suitable for the application (usually 512). The other variables in bwbuf are set up properly for use with bwrite. Bwrite returns the return of the open system call.

Bwsetup sets up bwbuf the same as bwopen but instead of opening the file it gets passed the descriptor of a file that is already opened for writing or reading and writing. It returns the descriptor.

Bwflush may be called at any time to force a write of any characters buffered in bwbuf. If successful bwflush returns 0. If it finds an obviously wrong bwbuf it clears errno (see INTRO 2) and returns -1. If the write of residual characters fails it returns -1 and errno is as left by the write system call.

Bwclose writes out any characters that may be left in bw_buf and closes the file descriptor. If bwbuf is obviously wrong it clears errno (see INTRO 2) and returns -1. If the write of residual characters fails it returns -1 and errno is as left by write system call. Otherwise it returns the return of the close system call.

FILES

/usr/include/bwrite.h

LIBRARY

/lib/lib1.a

SEE ALSO

open(2),close(2),write(2),intro(2),bopnclos(3L),bread(3L),fwrite(3)

DIAGNOSTICS

NAME

ca_funcs -- Initialize array of supported Common Analysis functions

SYNOPSIS

```
#include <ca_funcs.h>
```

DESCRIPTION

List of standard Common Analyses functions referenced by common header file ca_funcs.h

```
char *ca_funcs[] {
        "spa",
        "eca",
        "trk",
        "sda",
        "nca",
        "ppa",
        0
};
```

LIBRARY

/lib/lib1.a

NAME

chlnams -- array of channel names

SYNOPSIS

```
#include <chlnams.h>
```

DESCRIPTION

source:

```
char *chlnams[] {
        "mtc",
        "smtc",
        "trk",
        "misc",
        0
};
```

LIBRARY

/lib/lib1.a

NAME

clliofc - translate CLLI code to office name.

SYNOPSIS

```
#include "ofcclli.h"

ofcid = clliofc(ofcp, cllip, flag)
     char *ofcp;          /* office name string */
     char *cllip;         /* CLLI name string */
     int flag;            /* keep file open flag */
     int ofcid;           /* office id */
```

DESCRIPTION

Clliofc translates a standard Common Language Location Identification (CLLI) into a valid office name on the system. It assumes the CLLI is pointed to by cllip and is always CLLINAMSIZ characters long. The CLLI need not be null-terminated. Clliofc looks up the CLLI in the file /sccetc/ofcclli and copies the corresponding office name to the string pointed to by ofcp, then null-terminates it. The office name string must be at least OFCNAMSIZ+1 characters long. In addition, the subroutine returns the office's office id (values 0 to MAXOID-1) as the returned value (but see DIAGNOSTICS below).

The flag argument is used to tell the subroutine whether to leave the file descriptor open for the /sccetc/ofcclli file before returning. Flag = 0 means close the file descriptor; flag = 1 means leave it open. Programs which expect to call clliofc a lot may want to leave the file descriptor open, so that the subroutine does not have to re-open the file each time; other programs which use the subroutine only once or who are more concerned about file descriptor usage may want to close the file descriptor.

The /sccetc/ofcclli file is maintained by the commands RC:CLLI and VRFY:CLLI.

FILES

/sccetc/ofcclli

LIBRARY

/lib/lib1.a

SEE ALSO

dltclli(3L), ofcclli(3L)

DIAGNOSTICS

Ofcid will have the following values in error cases:

-1 if system error occurred (open or read failure); standard SCCS errors are printed in this case.

-2 if subroutine could not find CLLI in the /sccetc/ofcclli file.

NAME

clrbuf -- clear and audit pattern buffer

SYNOPSIS

clrbuf()

DESCRIPTION

clrbuf returns an integer with a value of zero or -1. If the buffer contains only null characters clrbuf returns a zero. If the buffer contains one or more character which are not null, clrbuf returns a -1.

clrbuf has no arguments. It operates on the buffer used by the pattern building character routines. The name of the buffer is _B_U_F_. _B_U_F_ is a 256 byte character array which should normally contain only null characters.

clrbuf provides a means of clearing any garbage that might be in the pattern buffer and provides an audit on the pattern buffer. If clrbuf ever returns a -1, there has been a program error. Either one of the character programs has failed or a user program has written into the pattern buffer. This subroutine is normally used only during debugging a program.

LIBRARY

/lib/lib3.a

NAME

cmd -- analize syntax of SCC command line

SYNOPSIS

cmd(cmdstr, ptrlst) char *cmdstr; char **ptrlst;

DESCRIPTION

The input to cmd is the sccsh command line pointed to by cmdstr, and the output is a list of pointers, ptrlst, to each argument in the command string. How cmd parses any given command is easily tested by the utility args(1L).

LIBRARY

/lib/lib1.a

DIAGNOSTICS

NAME

compar -- compare two character strings

SYNOPSIS

```
compar(str1,str2,len) char *str1,*str2;    int len;
```

DESCRIPTION

This subroutine compares two given strings, character by character, for a specified length and returns the following values:

if str1 < str2 -- return negative integer
if str1 = str2 -- return zero
if str1 > str2 -- return positive integer

Compar has three arguments, str1, str2, and len. The addresses of the two strings are passed as str1 and str2, while len is the number of bytes to be compared.

LIBRARY

/lib/lib1.a

SEE ALSO

strncmp(3)

NAME

compb -- compare input character string with a constant character string

SYNOPSIS

jsr r5,compb;name

DESCRIPTION

Enter with r0 set to point to input string.

Name is the address of the constant data--this string must be terminated with null byte.

On exit, c bit is set for failure, clear for success.

LIBRARY

/lib/lib1.a

NAME

cputm, cputm60 - return total child cpu time

SYNOPSIS

```
cputm()

long cputm60()
```

DESCRIPTION

cputm — return total child cpu time (sys + usr) since last call (in seconds); the first call returns garbage.

cputm60 — return total child cpu time (sys + usr) since last call (in 1/60 seconds); the first call returns garbage.

LIBRARY

/lib/lib1.a

NAME

cpyfld -- locate a specified field within a specified line of ASCII data and copy to a specified character buffer

SYNOPSIS

```
cpyfld(line,field,skip,length,from,to)
int line, field, skip, length;
struct GMBUF *from;
char *to;
```

DESCRIPTION

Cpyfld extracts a specified field or subfield from a data buffer, copies the contents of this field to a specified character buffer, and appends a null byte to the end of the character buffer. If an error is detected, cpyfld returns a value that is less than zero, as discussed below; otherwise, it returns the number of characters that have been copied to the character buffer, but not including the null byte. Calling programs should always check the number of characters copied to the character buffer to insure that a valid field or subfield has been extracted.

Cpyfld calls the library subroutine getfld(3L) to locate the specified field. Getfld breaks the specified line of input data into its respective fields, starting with field 0. Once the specified field has been located, the field, or a subfield within the field, is copied to the character buffer. Field separation characters are one or more tabs and/or blanks, a newline, an octal 212, or a null byte.

The ASCII data buffer, which is a structure of type GMBUF, is declared and allocated by the calling routine. Before calling this subroutine, the calling routine must fill the ASCII data buffer via the subroutine gtmsg(3L) or some other routine which performs a similar function.

For the argument descriptions that follow, the value nchar represents the total number of characters contained in the specified field in the data buffer.

The argument line is the number of the line in which the requested field is located. The range of values for line are:

0 <= line < GM_MAX_LNS

where GM_MAX_LNS is defined in the header file, gtmhdr.h.

The argument field is the number of the field that is to be located. The range of values for field are:

0 <= field < max. fields for specified line

The argument skip is the number of characters at the beginning of the field that are to be skipped before the specified subfield is copied to the specified character buffer. If the value of skip is 0, then no characters are skipped and copying begins with the first character in the field. The range of values for skip are:

0 <= skip < nchar

The argument length is the number of characters that are to be copied from the field to the character buffer. If the value of length is less than 0, then an error value is returned, as described below. If the value of length is 0, then all characters from skip to the end of the field are copied to the character buffer. If the value of length is greater than 0, then the number of characters copied to the character buffer is length or (nchar - skip), whichever is smaller.

The argument from is the address of a data buffer in which the requested line and field can be found. The data buffer must have a format that is identical to that required by the gtmsg(3L) subroutine.

The argument to is the address of the character buffer to which the specified field or subfield is to be copied. This address must be nonzero.

FILES

/usr/include/gtmhdr.h which contains the definitions for GMBUF, GM_MAX_LNS, CFR_LEN, CFR_SKIP, CFR_TO, CFR_FLD, CFR_LN.

LIBRARY

/lib/lib1.a

SEE ALSO

getfld(3L), gtmsg(3L)

DIAGNOSTICS

The error codes returned by this subroutine are:

CFR_LEN The argument length is less than zero.

CFR_SKIP The argument skip is less than zero or greater than the number of characters in the field.

CFR_TO The argument to contains an invalid address for the character buffer.

CFR_FLD The argument field is out of range.

CFR_LN The argument line is out of range.

NAME

crcbuf -- get CRC16 checksum

SYNOPSIS

```
unsigned crcbuf(bufp, nchar, oldcrc)
char *buff;
unsigned nchar, oldcrc;
```

DESCRIPTION

Crcbuf returns the CRC16 checksum of the nchar characters pointed to by bufp. Oldcrc represents the checksum of some other array of characters which should be taken into account in the calculation of the checksum for these characters. Oldcrc should be 0 if no such array exists.

LIBRARY

/lib/lib1.a

NAME

datchk -- check data validity

SYNOPSIS

datchk(nchar,base) int nchar,base;

DESCRIPTION

This subroutine checks the validity of the data that is passed to it in the external variable INPUT. If the data is valid, it is returned in the external variable, VALSTR, and the subroutine returns a 1. The data in VALSTR is right-justified in a field having a specified length, padded on the left with blanks, and terminated with a null byte. If the data is not valid, appropriate error information is returned in the external variables, E_SPCL, E_TYPE, E_CODE, E_NUM, and E_MSG, and a 0 is returned by this subroutine.

DATCHK has two arguments, nchar and base. Nchar specifies the length of the string in which the validated data is placed. Base specifies the base or radix of the input data contained in INPUT. These arguments are used to determine the validity of the input data. The validity checks made by this program are:

a. determine if base is between one and eleven, or sixteen, or thirty_two (means use base sixteen for a 101 ESS).

b. determine if INPUT contains more than nchar characters.

c. determine if INPUT contains a character that is not valid for the base specified.

The global variables used are:

```
char *E_SPCL;
char *E_TYPE;
char *E_CODE;
char *E_NUM;
char *E_MSG;
char *INPUT;
char VALSTR[33];
```

The error information returned is:

```
E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "002";
E_MSG=   "INVALID BASE.";      or

E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "003";
E_MSG=   "TOO MANY CHARACTERS IN DATA.";     or
```

```
E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "004";
E_MSG=   "INVALID CHARACTER IN DATA.";
```

LIBRARY

/lib/lib1.a

SEE ALSO

DIAGNOSTICS

Value for base must between one and eleven, or sixteen, or thirty-two for a 101 ESS.

NAME

dd_check -- check process ID

SYNOPSIS

```
dd_check(procid)
int procid;
```

DESCRIPTION

This subroutine checks existence of a process corresponding to a given process ID. It does this by attempting to send a SIGFPT signal to the process with the PID in question (this subroutine is intended to be used with programs which use the dd_ifree and dd_wnfree subroutines. These programs ignore SIGFPT). This subroutine actually executes another program which runs as root, because only programs which run as root can send signals to unrelated processes.

This subroutine returns one of the following values to the calling program:

-2 system error occurred.

-1 process with the given PID does not exist.

0 process with the given PID does exist.

Arguments:

procid is the process id of a process whose existence is to be checked.

LIBRARY

/lib/lib1.a

NAME

dd_freup - release access to data distributor control file

SYNOPSIS

```
dd_freup(oid)
int oid;

dd_wakup(ptr)
char *ptr;
```

DESCRIPTION

dd freup

This subroutine is intended to be used by programs which access the .ddcntl files. This subroutine, when called, will check a file (protect file) which contains an access slot for each office. Each access slot is either empty, in which case no process is currently accessing the .ddcntl file for that office, or the access slot contains the process ID of the process currently having access to the ddcntl file for that office.

The argument oid is the office ID of the office to be acessed.

This subroutine, when called, will check to see if the process ID in the access slot of the office of interest matches the process ID of the calling process. If it does, it will remove the process ID from the access slot and zero the access slot, indicating that the calling process no longer has access to that particular .ddcntl file. It will then post a semaphore, causing any process which has been waiting for access to be awakened. This subroutine should thus be called only after a program has finished with the .ddcntl file for an office.

Subroutine dd freup returns one of the following values to the calling program:

- **-5** Bad Protect file.
- **-4** Protection file semaphore permanently locked.
- **-3** System error occurred.
- **-2** Illegal office ID passed to this subroutine.
- **-1** This process ID was not in the office slot for the desired office; access slot not zeroed.
- **0** Free up worked O.K.

The Protect file is a temporary file and is rebuilt, by the first call to the subroutine dd wnfree or subroutine dd ifree, each time the system is rebooted.

dd_wakup

This routine will execute the program which sends a signal to a process. The program to be executed runs as super-user and can therefore send a signal to any process.

The argument ptr is a pointer to a string which is the process ID of the process to be awakened.

The return value indicates the success of the signal attempt:

```
 0 -- process sucessfully signalled
-1 -- process to be awakened did not exist
-2 -- another type of error ocurred
```

LIBRARY

/lib/lib1.a

SEE ALSO

dd_ifree(3L) dd_wnfree(3L)

NAME

dd_ifree - test for access to data distributor control file

SYNOPSIS

```
dd_ifree(oid)
int oid;
```

DESCRIPTION

This subroutine is intended to be used by programs which access the ddcntl files. This subroutine, when called, will check a file (protect file) which contains an access slot for each office. Each access slot is either empty, in which case no process is currently accessing the .ddcntl file for that office, or the access slot contains the process ID of the process currently having access to the .ddcntl file for that office.

If the access slot for the office of interest is empty, this subroutine places the process ID of the calling process in that access slot and returns. However, if the access slot, for the office of interest, is not empty then another process is currently accessing the desired ddcntl file. In that case, the subroutine will check to see if the process, corresponding to the process ID in the access slot, still exists. If it does not, this subroutine places the process ID of the calling process in the access slot of the office of interest and returns. However, if the process does exist, this subroutine would not go any further and return.

Any program which uses this subroutine should not use the SIGFPT signal (currently number 8). This signal is used by this subroutine to check the existence of processes which are currently accessing the ddcntl files. Signal handling for SIGFPT is set up automatically by this subroutine. Also, since this subroutine sets up and catches alarm system calls, the programs which use the Locking Mechanism should not use an alarm call which might interfere with this subroutine's operation.

Subroutine dd ifree returns one of the following values to the calling program:

-5 Bad Protection file.

-4 Protection file semaphore permanently locked.

-3 System error occurred.

-2 Illegal office ID passed to this subroutine.

-1 Access to .ddcntl file not gained.

0 Access to .ddcntl file granted.

The Protect file is a temporary file and is rebuilt, by the first call to subroutine dd_wnfree or subroutine dd_ifree, each time the system is rebooted.

LIBRARY

/lib/lib1.a

SEE ALSO

dd_wnfree(3L), dd_freup(3L), dd_check(3L)

NAME

dd_look -- send wakup signal to a process

SYNOPSIS

/prog/dd_look pid

DESCRIPTION

This program will send a wake up signal to the named process in order to check its existence. It is intended to be used by programs which access the .ddcntl file associated with Data Distributer program for Common Analysis. The argument pid is the process ID of the process ID of the process to be awakened.

This subroutine must run as root so it can send a signal to any other process.

The exit status of this program defines the success of the signalling attempt:

-2 some problem occurred.
-1 process to be awakened does not exist.
0 process to be awakened does exist.

LIBRARY

/lib/lib1.a

NAME

dd_wnfree - wait for access to data distributor control file

SYNOPSIS

```
dd_wnfree(oid)
int oid;

dd_snatch()
```

DESCRIPTION

dd_wnfree

This subroutine is intended to be used by programs which access the .ddcntl files. This subroutine, when called, will check a file (protect file) which contains an access slot for each office. Each access slot is either empty, in which case no process is currently accessing the .ddcntl file for that office, or the access slot contains the process ID of the process currently having access to the .ddcntl file for that office.

The argument oid is the office ID of the office to be accessed.

This subroutine will return only when it has gained access to the ddcntl file for the office of interest or when an interrupt or system error occurs. If necessary this subroutine causes the calling process to go to sleep until it gains access. If the subroutine does gain access to the desired .ddcntl file, this subroutine will place the process ID of the calling process in the access slot for the office of interest.

Any program which uses this subroutine whould not use the SIGFPT signal (currently number 8). This signal is used by this subroutine to check the existence of processes which are currently accessing the .ddcntl files. Signal handling for SIGFPT is set up automatically by this subroutine. Also, since this subroutine sets up and catches alarm system calls, the programs which use the Locking Mechanism should not use an alarm call which might interfere with this subroutine's operation.

Subroutine dd_wnfree returns one of the following values to the calling program:

-5 Bad Protection file.

-4 Protection file semaphore permanently locked.

-3 System error occurred.

-2 Illegal office ID passed to this subroutine.

-1 An interrupt occurred.

0 Access to .ddcntl file gained.

The Protect file is a temporary file and is rebuilt, by the first call to subroutine dd wnfree or subroutine dd ifree, each time the system is rebooted.

If a process cannot afford to go to sleep and wait for gaining access to a particular .ddcntl file, it should use subroutine dd ifree.

dd snatch

This routine will catch the SIGFPT signal. It should be used in any program which uses the dd wnfree routine. Note that, except for setting an alarm clock, it performes no function. The SIGFPT signal will thus simply awake the process from being asleep on a semaphore. The alarm clock is used in the unlikely event that the process receives the SIGFPT signal between the time that the signal handling is set up and the time that the process goes to sleep on the semaphore.

LIBRARY

/lib/lib1.a

SEE ALSO

dd_ifree(3L), dd_freup(3L), dd_check(3L)

NAME

dial -- construct DDD telephone number

SYNOPSIS

```
#include <dial.h>

dial(dn_ptr, number)
struct dntbl *dn_ptr;
char *number;
```

DESCRIPTION

The purpose of this subroutine is to construct a well formed DDD telephone number from one supplied by the caller and to command the specified dn11 to dial that number. The argument dn_ptr points to the structure returned by getds(3L) and number is the DDD telephone number supplied by the user.

Return values:

The file descriptor of the data multiplexor line

Errors are indicated as follows :

-1	No carrier, busy, or no answer.
-2	All equipment in use.
-3	Bad speed specification.
-4	Bad telephone number.
-5	Ioctl failure.
-6	Bad equipment resource table.
-7	No equipment exists - not specifyed in equipment resource table.
-8	Permission denied.
-9	Bad equipment.

FILES

/usr/include/dial.h /etc/d_dntable

LIBRARY

/lib/lib1.a

SEE ALSO

getds(3L)

NAME

diff -- locate first string difference

SYNOPSIS

```
diff(s1,s2)
char *s1, *s2;
```

DESCRIPTION

diff returns an integer indicating the position within the two strings that the first character mismatch was discovered. If the two strings are identical, the integer returned is -1.

s1 string to be used in comparison.

s2 string to be used in comparison.

An empty string is one whose first character is the null character. If both strings are empty, the integer returned has the value -1.

The two strings are compared character by character until one or both are terminated by the null character or until a mismatch is found. If the two strings are identical until one string is terminated by the null character while the other string is not terminated, the returned integer is the index of the null character.

If the two strings differ on a character other than the null character the index of the character position that differs is returned.

LIBRARY

/lib/lib3.a

SEE ALSO

strcmp(3)

NAME

dltclli - delete office's CLLI code translation.

SYNOPSIS

```
#include "ofcclli.h"

dltclli(ofcp)
    char *ofcp;          /* office name string */
```

DESCRIPTION

Dltclli deletes the Common Language Location Identification (CLLI) code for the specified office. It assumes the office name is pointed to by ofcp and is null-terminated. Dltclli scans the file /sccetc/ofcclli and blanks out every entry which has the specified office name in the entry's office name field.

The /sccetc/ofcclli file is maintained by the commands RC:CLLI and VRFY:CLLI.

FILES

/sccetc/ofcclli

LIBRARY

/lib/lib1.a

SEE ALSO

clliofc(3L), ofcclli(3L)

DIAGNOSTICS

None; system errors are reported in standard SCCS fashion but no indication is returned to the caller.

NAME

dp_add -- double precision add

SYNOPSIS

```
int *dp_add(num) int num;
```

DESCRIPTION

This subroutine adds a single-precision number, num, to a double-precision number contained in the external variable, WORD. The result of this addition is placed in WORD, and the address of WORD is returned to the calling program.

The global variables used are:

```
int WORD[2];
```

LIBRARY

/lib/lib1.a

SEE ALSO

binary(3L)

DIAGNOSTICS

NAME

dp_mul -- double precision multiply

SYNOPSIS

```
int *dp_mul(radix) int radix;
```

DESCRIPTION

This subroutine multiplies a double-precision number, stored in the external variable, WORD, by a single-precision number, radix. The address of WORD is returned by this function, unless an error is detected. In this case a 0 is returned by this subroutine and appropriate error information is returned in the external variables, E_SPCL, E_TYPE, E_CODE, E_NUM, and E_MSG.

DP_MUL has one argument, radix, which specifies the base of the number contained in WORD.

The global variables used are:

```
char *E_SPCL;
char *E_TYPE;
char *E_CODE;
char *E_NUM;
char *E_MSG;
int  WORD[2];
```

The error information returned:

```
E_SPCL=  "?D";
E_TYPE=  "  ";
E_CODE=  "LIB";
E_NUM=   "001";
E_MSG=   "PRODUCT LARGER THAN 32 BITS.";
```

LIBRARY

/lib/lib1.a

SEE ALSO

binary(3L)

DIAGNOSTICS

BUGS

NAME

dskacc, mdskacc - get number of disk accesses

SYNOPSIS

dskacc()

mdskacc()

DESCRIPTION

dskacc returns number of disk accesses of current process and its children since its last call

mdskacc is identical to diskacc except it is used by the measurment subroutines

LIBRARY

/lib/lib1.a

NAME

dtogl - turn data collection on or off

SYNOPSIS

```
dtogl(chptr,anlptr,stptr,ctlptr)
char *chptr;
char *anlptr;
char *stptr;
register struct DD_CNTL *ctlptr;
```

DESCRIPTION

This routine will turn the data collection for a specified office and channel on or off. This is done by placing the proper value in the flag byte associated with the appropriate function in the data collection control record of the office of interest. The flag value will be set to a channel name index if collection is turned on or to a -1 if collection is turned off. when data collection for a particular channel is to be turned on or off, the pointers for that channel will be zeroed. Since one channel may be collecting data for several types of analysis, checks of all flag bytes are made before zeroing any pointers. Additionally, since collection for a particular analysis/channel may be turned on several times in succession (and subsequently turned off an equal number of times), a counter is kept with each flag byte. This counter is incremented each time collection is turned on for an analysis/channel, and decremented when collection is turned off; data collection for an analysis/chan- nel will not actually be turned off until this counter is decremented to zero.

The routine returns one of several values:

0 - everything okay
1 - invalid channel or function
2 - stptr (see below) points to a string which is neither "START" nor "STOP"
3 - an attempt was made to turn off an analysis/channel which was already off
4 - an attempt was made turn collection on or off for an illegal analysis/channel
5 - an analysis/channel count was illegal (less than 0 or greater than maximum)

Arguments:

chptr	is a pointer to a string which is the name of the channel of interest.
anlptr	is a pointer to a string which is the name of the analysis function for which data is being (or will be) collected
stptr	is a pointer to a string which must be either "START" (for collection turn-on) or "STOP" (for collection turn-off)
ctlptr	points to the structure to be updated

Note that this routine will be manipulating data in the structure which is pointed to by the calling parameter ctlptr.

LIBRARY

/lib/lib1.a

NAME

e_acctxt -- provide access to the stored OM text field

SYNOPSIS

```
char *e_acctxt()
```

DESCRIPTION

This subroutine returns a pointer to the string containing the text field of the current stored OM.

LIBRARY

/lib/lib1.a

NAME

e_assert -- internal program constistancy check

SYNOPSIS

```
#include <errfct.h>

e_assert(true_expression, msg)
char *msg;
```

DESCRIPTION

If the true expression is not true an internal error OM is generated and output and the trap function is called. Msg is a pointer to the OM string that is to be output.

Return values:

```
 0 -- if true expression is true
-1 -- if not
```

LIBRARY

/lib/lib1.a

NAME

e_clstst -- error class test

SYNOPSIS

```
#include <errfct.h>
#include <syserr.h>

e_clstst(INHXXX | INHYYY | INHZZZ ...)
```

DESCRIPTION

The arguments INHXXX, etc. are defined in /usr/include/errfct.h and define the error classes to be tested. e_clstst returns INHXXX (non-zero) if errno is equal to EXXX, as defined in /usr/include/syserr.h; zero is returned otherwise. If the argument is NOERR, non-zero is always returned.

FILES

/usr/include/errfct.h /usr/include/syserr.h

LIBRARY

/lib/lib1.a

NAME

e_form - format and store an output message (OM)

SYNOPSIS

```
e_form (errnum, msg1, msg2, ..., 0)
   char *errnum;
   char *msg1;
   char *msg2;
    ...
```

DESCRIPTION

Store an OM for subsequent output by e_output(3L). This is normally called by other routines e_stdio(3L), e_splerr(3L), e_intern(3L)), but may be called directly. It is particularly useful when the text field of the OM is a concatenation of several strings.

The errnum argument specifies the error number of the OM. It should point to a three-digit numerical string. The error code for the OM is determined by the standard error code for the process as set up by e_setcode or e_setup(3L).

The strings pointed to by the (variable number of) msg? arguments are concatenated to become the text field of the OM.

LIBRARY

/lib/lib1.a

NAME

e_init, e_term -- generate initialization and termination messages for autonamous processes.

SYNOPSIS

```
#include <efffct.h>

e_init()

e_term(exitcode)
```

DESCRIPTION

Initialization and Termination messages should be generated with e init when the program is initialized; and e term when the program terminates in a controlled manner, whether because of the receipt of a signal or because of a fatal error. e init requires no arguments. e term requires an exit status as an argument, which it passes to the exit(2) function. These functions automatically create and remove, respectively, a temporary file with a name based upon the progran name (as set by e setup(3L) or e setname(3L)). When the file exists when e init is called, it is presumed that the program died in a uncontrolled manner. Thus a "REINITIALIZED" message, with the standard alarm level, is generated in this case. Note that all the temporary files are automatically removed when the system is rebooted.

Either e setup(3L) or e setname(3L) must be used to set the program name before e init is called. This is needed both to formulate the temporary file name and for inclusion in the messages.

The return value of e init is zero if a normal initialization is detected and -1 if an abnormal reinitialization is detected. The e term function never returns.

LIBRARY

/lib/lib1.a

NAME

e_intern - generate "internal" output message (OM)

SYNOPSIS

```
# include <errfct.h>
e_intern(NORMINTERN, msg, inhflag)
   char *msg;
   int inhflag;
e_intern(TRINTERN, msg, inhflag)
```

DESCRIPTION

An OM is created to describe an "internal" No.2 SCCS error (such as failure of an internal program consistncy check, or other error that "can't happen"). Normally NORMINTERN is used as the first argument; TRINTERN is used if you always want the field to submit a TR if the problem shows up. The msg argument should point to an (upper case) string serving to identify but not explain the error condition.

The inhflag argument is the standard error system inhibit flag (See e_syscall(3L)). In this case only the values **ALLERR** (output the OM) or **NOERR** (just store away the OM) are meaningful.

LIBRARY

/lib/lib1.a

NAME

e_new - modify a stored output message (OM)

SYNOPSIS

```
#include <errfct.h>
e_newcode (errcode)
   char *errcode;
e_newnum (errnum)
   char *errnum;
e_newnote ()
e_newlvl (almlvl)
   int almlvl;
e_newlname (libname)
   char *libname;
char (*e_acctxt())
```

DESCRIPTION

A stored OM (See e_syscall(3L), e_splerr(3L), e_intern(3L), e_form(3L)) may be modified.

E_newcode changes the error code of the OM (See sccerr(3L));

e_newnum changes the error number;

e_newnote makes the OM a "notification" alarm (ie, alarm level "none" but alerted);

e_newlname inserts a library routine name into an OM for errors occurring with library routines.

The arguments errcode, errnum, and almlvl are identical to those used in e_setup(3L).

E_acctxt returns a pointer to the null terminated text field of the OM so that it may be modified. It may be lengthened, but only up to the second null byte.

SEE ALSO

e_output(3L), e_setup(3L)

DIAGNOSTICS

none

LIBRARY

/lib/lib1.a

NAME

e_reflex -- Input Message error reporting

SYNOPSIS

```
e_already(x,opt)  char *x, *opt;

e_arb(s1,s2,......$n,0)  char *s1, *s2, ....,*$n;

e_backgrd()

e_chl(x,opt)  char *x, *opt;

e_exkw(x,opt)  char *x, *opt;

e_incd(x,y,opt)  char *x, *y, *opt;

e_inckw(x,y,opt)  char *x, *y, *opt;

e_inperr(opt)  char *opt;

e_invc(x,opt)  char *x, *opt;

e_invd(x,y,opt)  char *x, *y, *opt;

e_invkw(x,opt)  char *x, *opt;

e_ip()

e_kw(s1,s2, .....,0)  char *s1, *s2, ..

e_loc(x,opt)  char *x, *opt;

e_lperm(x,opt)  char *x, *opt;

e_misd(x,opt)  char *x, *opt;

e_miskw(x,opt)  char *x, *opt;

e_ng(s1,s2, .....,0)  char *s1, *s2, ..

e_ofc(x,opt)  char *x, *opt;

e_ok()

e_perm(x,opt)  char *x, *opt;

e_pf()

e_punct(x,y,opt)  char *x, *y, *opt;

e_rgerr(x,opt)  char *x, *opt;

e_rlbsy(opt)  char *opt;
```

```
e_rlovld(opt)  char *opt;

e_sched()

e_spinoff()

e_ssys(x,opt)  char *x, *opt;

e_stderr()

e_stdin()

e_stdout()

e_syntax(s1,s2, .....,0)  char *s1, *s2, ..

e_uperm(x,y,opt)  char *x, *y, *opt;
```

DESCRIPTION

Each function (except e_ok , e_ip , and e_pf) is provided as a standard method of responding to reflexive errors in an Input Message (IM) or in a prompted user response error.

e_ok , e_ip , and e_pf are provided for the generation of common, high usage non-error messages.

Each function is listed below with the error message format. Note, however, that the message acknowlegements (OK, NG, etc.) are not preceeded by a newline and will be outputted on the same line as the input command. Some functions require arguments "x" and "y" of type (char *). If non-zero, the string is inserted in the message where indicated by <x> and <y>. Some functions require an argument "opt" of type (char *). If non-zero, the string contained in square brackets will be outputted in the formats below.

Those functions accepting an arbitrary number of arguments "s1", "s2",... of type (char *) require that the last argument be zero.

Each of the routines produces the following output which is directed to file descriptor 2.

```
e_already(x,opt)
 NG
 ALREADY <x> [; <opt>]

e_arb(s1,s2,.....,0)
  s1
s2 s3 ......

e_backgrd()
```

```
 NG
 Command can not executed in the background

e_chl(x,opt)
 ?E
 INVALID CHL: <x> [; <opt>]

e_exkw(x,opt)
 ?E
 EXTRA KEYWORD <x> [; <opt>]

e_incd(x,y,opt)
 ?E
 INCONSISTENT DATA <x>[, WITH <y>][; <opt>]

e_inckw(x,y,opt)
 ?E
 INCONSISTENT KEYWORD <x>[, WITH <y>][; <opt>]

e_inperr(opt)
 ?E
 INPUT ERROR [; <opt>]

e_invc(x,opt)
 ?E
 INVALID CHARACTER <x>[; <opt>]

e_invd(x,y,opt)
 ?E
 INVALID DATA <x> [FOR KEYWORD <y>][; <opt>]

e_invkw(x,opt)
 ?E
 INVALID KEYWORD: <x>[; <opt>]

e_ip()
 IP

e_kw(s1,s2,....,0)
 VALID KEYWORDS: <s1>
           <s2>
             .
             .

e_loc(x,opt)
 ?E
 INVALID LOCATION: <x>[; <opt>]

e_lperm(x,opt)
 ?E
 <x> NOT IN THIS LOCATION[; <opt>]

e_misd(x,opt)
```