

DLGLib

COLLABORATORS

	<i>TITLE :</i> DLGLib		
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>	<i>SIGNATURE</i>
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	DLGLib	1
1.1	DLGLib.Doc	1
1.2	PORT functions	2
1.3	USER functions	4
1.4	dlg.library/ActivatePort	5
1.5	dlg.library/AddArea	6
1.6	dlg.library/AddStruct	6
1.7	dlg.library/AFormat	7
1.8	dlg.library/Age	8
1.9	dlg.library/AmigaTime	9
1.10	dlg.library/AppendFile	9
1.11	dlg.library/ArgParse	10
1.12	dlg.library/BCGet	11
1.13	dlg.library/BCMsg	12
1.14	dlg.library/BCPend	13
1.15	dlg.library/BCResume	13
1.16	dlg.library/BinPos	14
1.17	dlg.library/BoolQuery	15
1.18	dlg.library/BorrowArea	16
1.19	dlg.library/BroadCast	17
1.20	dlg.library/CallEditor	18
1.21	dlg.library/Capitalize	19
1.22	dlg.library/Cat	19
1.23	dlg.library/CD	20
1.24	dlg.library/ChainProgram	21
1.25	dlg.library/CheckUser	21
1.26	dlg.library/ClearLine	22
1.27	dlg.library/CloseGroup	23
1.28	dlg.library/Clr	23
1.29	dlg.library/Copy	24

1.30	dlg.library/CronEvent	25
1.31	dlg.library/DB	26
1.32	dlg.library/DeActivatePort	26
1.33	dlg.library/DelArea	27
1.34	dlg.library/DelDir	28
1.35	dlg.library/DeleteStruct	28
1.36	dlg.library/DeScore	29
1.37	dlg.library/DialogBatch	30
1.38	dlg.library/DirSize	31
1.39	dlg.library/DispBuffer	31
1.40	dlg.library/DispForm	33
1.41	dlg.library/DispMsg	33
1.42	dlg.library/DLGBinSearch	35
1.43	dlg.library/DLGGetSer	35
1.44	dlg.library/DLGPatternMatch	37
1.45	dlg.library/DLGProtoStatus	37
1.46	dlg.library/DLGQuery	38
1.47	dlg.library/DLGReleaseSer	39
1.48	dlg.library/DLGSearch	40
1.49	dlg.library/Draw_Line	41
1.50	dlg.library/EnterArea	41
1.51	dlg.library/Exists	42
1.52	dlg.library/ExistsGlobalArea	43
1.53	dlg.library/FileCopy	43
1.54	dlg.library/FileSize	44
1.55	dlg.library/FreeArea	45
1.56	dlg.library/FreeAreaInfo	46
1.57	dlg.library/FreeMenu	46
1.58	dlg.library/FreePort	47
1.59	dlg.library/FreePortInfo	48
1.60	dlg.library/FreeResource	48
1.61	dlg.library/FreeResReport	49
1.62	dlg.library/GetAreaInfo	50
1.63	dlg.library/GetChar	50
1.64	dlg.library/GetComment	51
1.65	dlg.library/GetComputerType	52
1.66	dlg.library/GetDevName	52
1.67	dlg.library/GetFileDate	53
1.68	dlg.library/GetFirstStruct	53

1.69	dlg.library/GetHiLowFPointers	54
1.70	dlg.library/GetHiLowPointers	55
1.71	dlg.library/GetLang	56
1.72	dlg.library/GetLevel	56
1.73	dlg.library/GetOrigin	57
1.74	dlg.library/GetPath	58
1.75	dlg.library/GetPortInfo	59
1.76	dlg.library/GetResReport	60
1.77	dlg.library/GetStruct	61
1.78	dlg.library/HandleBCMsgs	62
1.79	dlg.library/ImmedLockPort	62
1.80	dlg.library/ImportPublicMsg	63
1.81	dlg.library/Inform	64
1.82	dlg.library/IntQuery	65
1.83	dlg.library/KillMsg	66
1.84	dlg.library/LeaveArea	67
1.85	dlg.library/ListAreas	67
1.86	dlg.library/ListPorts	68
1.87	dlg.library/ListSIGS	69
1.88	dlg.library/LoadLang	70
1.89	dlg.library/LockArea	70
1.90	dlg.library/LockMenu	71
1.91	dlg.library/LockPort	72
1.92	dlg.library/LockResource	73
1.93	dlg.library/LogOut	73
1.94	dlg.library/MDate	74
1.95	dlg.library/More	75
1.96	dlg.library/NextInGroup	76
1.97	dlg.library/OpenGroup	76
1.98	dlg.library/OverlayProgram	77
1.99	dlg.library/Pause	78
1.100	dlg.library/PrintSpace	78
1.101	dlg.library/PurgeMenu	79
1.102	dlg.library/PutChar	80
1.103	dlg.library/PutHiLowFPointers	80
1.104	dlg.library/PutHiLowPointers	81
1.105	dlg.library/ReadArea	82
1.106	dlg.library/ReadChar	83
1.107	dlg.library/ReadRam	83

1.108dlg.library/ReadUser	84
1.109dlg.library/ReceiveFile	85
1.110dlg.library/ResourceMsg	86
1.111dlg.library/ResumeTime	86
1.112dlg.library/ScreenBuffer	87
1.113dlg.library/ScreenMsg	88
1.114dlg.library/ScreenPath	88
1.115dlg.library/SDraw_Line	89
1.116dlg.library/SearchEnd	89
1.117dlg.library/SearchNext	90
1.118dlg.library/SearchStart	91
1.119dlg.library/SendBulletin	91
1.120dlg.library/SendFile	92
1.121dlg.library/SendCtlMsg	93
1.122dlg.library/SendPrivateMsg	94
1.123dlg.library/SendPublicMsg	95
1.124dlg.library/SendRawMsg	96
1.125dlg.library/SmartRename	97
1.126dlg.library/SMDate	98
1.127dlg.library/Stricmp	98
1.128dlg.library/StripPath	99
1.129dlg.library/StripSpaces	100
1.130dlg.library/Strnicmp	100
1.131dlg.library/Substitute	101
1.132dlg.library/SuspendTime	102
1.133dlg.library/TBaud	102
1.134dlg.library/TCheckCarrier	103
1.135dlg.library/TColors	103
1.136dlg.library/TCont	104
1.137dlg.library/TDevQuery	105
1.138dlg.library/TFreeze	105
1.139dlg.library/TGetSer	106
1.140dlg.library/TGetTitle	107
1.141dlg.library/TimeUntilShutdown	107
1.142dlg.library/TInTrans	108
1.143dlg.library/TKill	108
1.144dlg.library/TOutTrans	109
1.145dlg.library/TransferPortLock	110
1.146dlg.library/TranslateBuffer	110

1.147dlg.library/TRecover	111
1.148dlg.library/TScreen	112
1.149dlg.library/TSendBreak	113
1.150dlg.library/TSetFlags	113
1.151dlg.library/TString	115
1.152dlg.library/TTimeDelay	115
1.153dlg.library/TTitle	116
1.154dlg.library/TUnSetFlags	117
1.155dlg.library/TWindow	117
1.156dlg.library/TWinHeight	118
1.157dlg.library/UnderScore	119
1.158dlg.library/UnpackTime	119
1.159dlg.library/Upper	120
1.160dlg.library/WaitingMail	121
1.161dlg.library/WhenEvent	121
1.162dlg.library/WriteEvent	122
1.163dlg.library/WriteLog	123
1.164dlg.library/WriteRam	123
1.165dlg.library/WriteUser	124
1.166dlg.library/XAFPrintf	125
1.167dlg.library/XASPrintf	126
1.168General STRUCTURE functions	127
1.169Formatted I/O functions	128
1.170Time Functions	129
1.171File Manipulation Functions	130
1.172LOGGING functions	131
1.173UTILITY functions	131
1.174BROADCAST Functions	132
1.175AREA functions	132
1.176EXEC functions	134
1.177SERIAL functions	134
1.178RESOURCE functions	135
1.179AFPrintf()	135
1.180format.c	136
1.181ASPrintf()	137
1.182Using this guide	138
1.183Distribution	139
1.184Copyright	140
1.185Credits	140
1.186Contacts	140
1.187Index	141

Chapter 1

DLGLib

1.1 DLGLib.Doc

```
$Id: DLGLib 1.3 1996/11/20 19:39:56 Jeff_Grimmett Exp ↵
Jeff_Grimmett $
```

As a developer of DLG utilities, this guide is designed to aid you in creating programs to work well with DLG Pro. Library functions are arranged in a manner similar to the SAS/C library guide for your convenience.

```
~Using~this~guide~
```

```
~Distribution~~~~~
```

```
~Copyright~~~~~
```

```
~Credits~~~~~
```

```
~Contacts~~~~~
```

```
~Alphabetical~Index
```

```
The functions are arranged by category, for your convenience, and ↵
an
```

alphabetical index is included for an alternative method of locating the desired function.

```
~~~~~PORT~functions
```

```
-- functions that address external ports.
```

```
~~~~~USER~functions
```

```
-- functions that manipulate user data
```

```
STRUCTURE~Functions
```

```
-- manipulate structured data items
```

```
~~~~~I/O~functions
```

```
-- user I/O functions
```

```
~~~~~TIME~Functions
```



```

-- functions related to time

~~~~~FILE~Functions
-- functions that control file I/O

~~LOGGING~functions
-- functions for logging and informing

~~~STRING~functions
-- strings and other things

BROADCAST~Functions
-- Broadcast to and from ports/users

~~~~~AREA~functions
-- Manipulation of file and message areas

~~~~~EXEC~functions
-- call or use external programs

~~~SERIAL~functions
-- Control I/O on the serial port

~RESOURCE~functions
-- Misc resources and structures

```

Along with the above functions, please see
 format.c
 for routines that will
 interface your program to DLG's internal print routines.

1.2 PORT functions

Port functions relate to the activation, deactivation, and ↵
 management of
 external data ports. It is through these ports that the users access the
 BBS. Many of these functions are extremely low-level, so understanding of
 them is essential.

```

~~~~~ActivatePort()
-- Activate a port

~~DeActivatePort()
-- Deactivate a port

~~~~~FreePort()
-- Free a lock on a port

~~~~~GetDevName()
-- Report the port application is running on

~~~~~FreePortInfo()
-- Free information about a port

```

```
~~~~~GetPortInfo()
-- Get information about a port

~~~ImmedLockPort()
-- Lock a port with an "immediate" lock

~~~~~ListPorts()
-- Gets a list of active ports

~~~~~LockPort()
-- Lock a port

~~~~~Logout()
-- Log user out

~~~~~TBaud()
-- Set the baud rate for a port

~~~TCheckCarrier()
-- Checks for the presence of a carrier

~~~~~TColors()
-- Change the colors for a port

~~~~~TCont()
-- Unfreeze a port

~~~~~TDevQuery()
-- Get information about a port

~~~~~TFreeze()
-- Suspend all I/O on a port

~~~~~TGetSer()
-- Get serial information for a port

~~~~~TGetTitle()
-- Get teh screen/window title for a port

~~~~~TInTrans()
-- Set the input translation table for a port

~~~~~TKill()
-- Kill a port

~~~~~TOutTrans()
-- Set the output translation table for a port

TransferPortLock()
-- Change the lock on a port

~~~~~TRecover()
-- Recover a killed port

~~~~~TScreen()
-- Open/close a screen for a port
```

```

~~~~~TSendBreak()
-- Not currently implemented

~~~~~TSetFlags()
-- Set handler flags

~~~~~TString()
-- Pretend a user typed a string

~~~~~TTimeDelay()
-- Set the timeout delay for a port

~~~~~TTitle()
-- Change the screen/window title for a port

~~~~~TUnSetFlags()
-- Unset handler flags

~~~~~TWindow()
-- Open/close a window on a port

~~~~~TWinHeight()
-- Change the height of the window on a port

```

1.3 USER functions

These functions present you with ways to manipulate and read user data ↔ without going through the trouble of decyphering everything yourself. Most of these functions are RELATIVELY harmless, except any user that you mess up will of course be slightly upset about it...

```

~~~~~AddArea()
-- Append an area to user's global area list

~~~~~Age()
-- Computer a user's age

~~~~~CheckUser()
-- Check whether a user exists

~~~~~CloseGroup()
-- end access of a group

~~~~~DelArea()
-- Delete an area from user's global area list

ExistsGlobalArea()
-- Checks if area exists in user's global area list

~GetComputerType()
-- Get the name of a computer type

~~~~~GetLevel()

```

```

-- Get the level of a user

~~~~~NextInGroup()
-- get the next name in a group

~~~~~OpenGroup()
-- Open a group to be accessed

~~~~~ReadRam()
-- Read user's RAM_FILE structure

~~~~~ReadUser()
-- Read a user's USER_DATA and RAM_FILE structures

~~~~~WriteRam()
-- Write a user's RAM_FILE structure

~~~~~WriteUser()
-- Write a user's USER_DATA structure

```

1.4 dlg.library/ActivatePort

NAME

ActivatePort -- Activate a port

SYNOPSIS

```

result = ActivatePort(port,bgcommand)
                A0  A1
LONG ActivatePort(char *,char *)

```

FUNCTION

Activates a port for use with the resource manager. A port must be activated before other calls (such as locking or freeing the port) can be made.

INPUTS

```

port          -- Three-character port name

bgcommand    -- Command to be run whenever the port becomes free, or
              "" for no background command.

```

RESULT

The result is an error message (see resman.h for #defines).

EXAMPLE

```

result = Activateport("TR0","DLG:Setup TR0");

```

NOTES

BUGS

SEE ALSO

DeActivatePort()

1.5 dlg.library/AddArea

NAME

AddArea -- Append an Area to one of the global area files

SYNOPSIS

```
result = AddArea(path, area)
                A0    D0
BOOL AddArea(char *, USHORT)
```

FUNCTION

Appends an Area to one of the global area files. The global area files are in the user's directory and contain an area list of short intergers. These are the GlobalAreas.file, GlobalAreas.msg and GlobalAreas.archive.

If the global area file doesn't exist, it is created. No checking is done to see if the area already exists in the global area file.

INPUTS

path -- Complete path and filename of the global area file

area -- Area number to be appended.

RESULT

TRUE area successfully appended
FALSE area not appended

EXAMPLE

```
result = AddArea("User:Joe_Smith/GlobalAreas.msg", 10);
```

NOTES

BUGS

SEE ALSO

```
DelArea()
,
ExistsGlobalArea()
```

1.6 dlg.library/AddStruct

NAME

AddStruct -- Add a structure to a file

SYNOPSIS

```
result = AddStruct(filename, structptr, structsize, fieldsize)
                A0      A1      D0      D1
LONG AddStruct(char *, char *, USHORT, USHORT)
```

FUNCTION

Adds a structure to a sorted file on disk, or replaces the structure if it already exists.

INPUTS

```
filename    -- Filename to place structure in.  If the file
              doesn't exist, it will be created.

structptr   -- Pointer to structure.

structsize  -- Size of structure.

fieldsize   -- Size of keyfield (1st field) for sorting.  This
              field must be a string.
```

RESULT

```
-1 if operation failed
 0 if structure was replaced (already existed)
 1 if structure was added
```

EXAMPLE

```
result = AddStruct("Structures.dat",mystructure,
                  sizeof(*mystructure),10);
```

NOTES

Common mistake #426: do not use `strlen()` to determine the size of the first field of the structure! `strlen()` only returns the length of the string UP TO THE TERMINATING NULL (`\0`) in the string! Either specify the correct length of the string, or use another method to find the size of the structure element.

BUGS

SEE ALSO

```
DeleteStruct ()
,
GetStruct ()
,
GetFirstStruct ()
```

1.7 dlg.library/AFormat

NAME

AFormat -- Low level I/O routine

SYNOPSIS

```
result = AFormat(User,data,putsub,fmt,argp)
          A0  A1  A2      D0  A3
LONG AFormat(struct USER_DATA *,void *,int (*putsub)(),char *,char *)
```

FUNCTION

Does standard 'C'-style formatting. This function is generally not called directly.

INPUTS

User -- Optional USER_DATA structure (used for ansi color).

data -- Data that gets passed through to putsub().

putsub -- Function to be called to output a character. This function takes two arguments -- the character to be output, and a pointer to the data being passed through.

fmt -- Format string containing text and switches (see any printf() documentation for examples of the switches).

argp -- Pointer to a memory area (usually the stack) that contains the arguments to the formatting statements. Note that all arguments must be long values.

RESULT

The result is the number of characters output.

EXAMPLE

NOTES

Compatible with most printf() format strings. If the User structure is passed, the format string may include DLG %a and %b color codes. There is no floating point support nor is %x formatting supported.

BUGS

When using format.c's AFPrintf() or ASPrintf() to interface to this function, all arguments are converted to LONGs. A format of "%hd" should not be used and will cause invalid results, use "%d" instead. If you call this function directly, you should use "%hd" for SHORTs and will get the proper results.

SEE ALSO

```
XAFPrintf()
,
XASPrintf()
```

1.8 dlg.library/Age

NAME

Age -- Compute a person's age

SYNOPSIS

```
result = Age(year, month, day)
           D0   D1   D2
LONG Age (SHORT, SHORT, SHORT)
```

FUNCTION

Computes an age based on a birth year, month, day, and the current date.

INPUTS

year -- Year the person was born.

month -- Month the person was born (January = 0).

day -- Day the person was born.

RESULT

The person's age, in years.

EXAMPLE

```
age = Age(1969,10,21);
```

NOTES

BUGS

SEE ALSO

1.9 dlg.library/AmigaTime

NAME

AmigaTime -- Get the current time

SYNOPSIS

```
result = AmigaTime()
```

```
ULONG AmigaTime(void)
```

FUNCTION

Get the current system time, in seconds elapsed since midnight, January 1, 1978.

INPUTS

RESULT

The time

EXAMPLE

```
secs = AmigaTime();
```

NOTES

BUGS

SEE ALSO

UnpackTime()

,
MDate()

,
SMDate()

1.10 dlg.library/AppendFile

NAME

AppendFile -- Append a timestamped line to a file

SYNOPSIS

```
result = AppendFile(filename,buffer)
```

A0 A1

```
LONG AppendFile(char *,char *)
```

FUNCTION

Appends a string to a file, preceded by a timestamp.

INPUTS

filename -- File to append line to.

buffer -- Buffer to be written.

RESULT

-1 if operation failed

0 if operation was successful

EXAMPLE

```
result = AppendFile("MyLogFile","Something important happened");
```

NOTES

BUGS

SEE ALSO

1.11 dlg.library/ArgParse

NAME

ArgParse -- Parse a string into an array of words

SYNOPSIS

```
result = ArgParse(string,argarray,maxnum)
```

A0 A1 D0

```
LONG ArgParse(char *,char **,UBYTE)
```

FUNCTION

Parses a string into an array of words.

INPUTS

string -- String to be parsed

argarray -- Array of char pointers. Each will be set to point to a word, with the last being set to NULL. The array should have at least one more than 'maxnum' char pointers allocated.

maxnum -- Maximum number of arguments to be parsed.

RESULT

The number of arguments parsed.

EXAMPLE

```
numargs = ArgParse("Arg1 Arg2 Arg3",MyArgs,4);
```

NOTES

This function actually CHANGES the original string (ala strtok()), so if you want to preserve the original contents of your string, either operate on a copy of it or make a copy for safekeeping.

ArgParse does not clear out the char array it is pointed to between invocations, so there may be residual words in any unused parts of the array from one invocation to the next. The ONLY legitimate way to know how many arguments were parsed is to read the return value from the function.

The results of ArgParse can best be regarded as analogous to **argv and argc usage in normal C programming.

BUGS

SEE ALSO

1.12 dlg.library/BCGet

NAME

BCGet -- Get a BroadCast message from the resource manager

SYNOPSIS

```
result = BCGet(port,buffer)
           A0  A1
LONG BCGet(char *,char *)
```

FUNCTION

Gets a pending BroadCast message from the resource manager if one is available.

INPUTS

port -- Port the application is running on.

buffer -- Buffer to place the message in (should have room for 80 characters plus null termination).

RESULT

Error message as define in broadcast.h.

EXAMPLE

```
while(BCGet(port,buf)==BCNOERR) printf("Message is [%s]\n",buf);
```

NOTES

BUGS

SEE ALSO

HandleBCMgs()

```

    ,
    BCPend()
    ,
    BCResume()
    ,
    BCMsg()

```

1.13 dlg.library/BCMsg

NAME

BCMsg -- Low-level broadcast routine

SYNOPSIS

```

result = BCMsg(ports,buffer,type,flags)
           A0   A1   D0   D1
LONG BCMsg(char *,char *,UBYTE,UBYTE)

```

FUNCTION

Routine used by all other broadcast functions to send messages to the BroadCast Manager. Should not be called directly.

INPUTS

```

ports  -- String containing list of ports to be affected
         (example "TR0TR1TL0").

buffer -- Buffer to be broadcast, or filled in (
         BCGet
         ).

type   -- Type of message (see broadcast.h for #defines).

flags  -- Misc. flags (see broadcast.h for #defines).

```

RESULT

Error message as defined in broadcast.h.

EXAMPLE

NOTES

BUGS

SEE ALSO

```

Broadcast()
,
BCPend()
,
BCResume()
,
BCGet()
,
HandleBCMsgs()

```

1.14 dlg.library/BCPend

NAME

BCPend -- Pend automatic printing of BroadCast messages

SYNOPSIS

```
result = BCPend(port)
        A0
LONG BCPend(char *)
```

FUNCTION

Suspends the automatic printing of BroadCast messages on a port by the BroadCast Manager. Messages should be suspended by applications that don't wish to be interrupted by messages or that wish to handle messages internally (using

```
BCGet()
).
```

Applications that suspend messages should always make sure to resume printing when they exit (using

```
BCResume()
).
```

INPUTS

port -- Port to suspend messages on.

RESULT

Error message as defined in broadcast.h.

EXAMPLE

```
error = BCPend("TR0");
```

NOTES

BUGS

SEE ALSO

```
BCResume()
,
BCGet()
,
HandleBCMsgs()
,
BroadCast()
,
BCMsg()
```

1.15 dlg.library/BCResume

NAME

BCResume -- Resume printing of BroadCast messages

SYNOPSIS

```
result = BCResume(port)
```

```

                                A0
LONG BCResume(char *)

FUNCTION
  Resumes the printing of BroadCast messages on a port by the resource
  manager.

INPUTS
  port -- Port to resume messages on.

RESULT
  Error message as define in broadcast.h.

EXAMPLE
  error = BCResume("TR0");

NOTES

BUGS

SEE ALSO

```

```

    BCPend()
    ,
    BCGet()
    ,
    HandleBCMsgs()
    ,
    BroadCast()
    ,
    BCMsg()

```

1.16 dlg.library/BinPos

```

NAME
  BinPos -- Binary search for a structure in a file

SYNOPSIS
  result = BinPos(fh, filesize, structptr, structsize, fieldsize, returnptr)
                                A0 D0      A1      D1      D2      A2
LONG BinPos(BPTR, ULONG, char *, USHORT, USHORT, ULONG *)

FUNCTION
  Carries out a binary search for a structure in a file on disk.

INPUTS
  fh          -- AmigaDOS filehandle to search in.

  filesize   -- Size of the file.

  structptr  -- Pointer to structure to search for. The keyfield
               must be filled in.

  structsize -- Size of structure.

```

fieldsize -- Size of keyfield (1st field of structure).
Keyfield must be a string.

returnptr -- Pointer to return file position in.

RESULT

<0 if structure should fit before '*returnptr' in file
0 if structure should replace structure at '*returnptr' in file
>0 if structure should fit after '*returnptr' in file

The contents of 'returnptr' are modified to give a reference point.

EXAMPLE

```
result = BinPos(MyFileHandle,size,MyStruct,sizeof(*MyStruct),10,&ptr);
```

NOTES

BUGS

SEE ALSO

1.17 dlg.library/BoolQuery

NAME

BoolQuery -- Ask a (Y/N) question

SYNOPSIS

```
result = BoolQuery(query,opt,ui)
                A0    D0  A1
BOOL BoolQuery(char *,UBYTE,struct UserInfo *)
```

FUNCTION

To get a (Y/N) response from the user.

INPUTS

query -- String containing a question to ask (the "(Y/n)" or
"(y/N)" is supplied by the routine).

opt -- Default response (TRUE=yes, FALSE=no).

ui -- UserInfo structure as defined in input.h

RESULT

TRUE if user responds 'yes'
FALSE if user reponds 'no'

EXAMPLE

```
yesno = BoolQuery("Yes or No?",TRUE,MyUserInfo);
```

NOTES

BUGS

SEE ALSO

```

    DLGQuery()
    ,
    IntQuery()

```

1.18 dlg.library/BorrowArea

NAME

BorrowArea -- Lock an area for a short period of time

SYNOPSIS

```

result = BorrowArea(area,passwd,reason,pri,flags)
                D0  A0      A1      D1  D2
LONG BorrowArea(USHORT,char *,char *,char,UBYTE)

```

FUNCTION

Locks an area, but should only be used if the area will be locked for a very short period of time (a few seconds at most).

INPUTS

area -- Number of the area to be locked.

passwd -- Password to lock area with.

reason -- Reason the area is being locked.

pri -- Priority for lock (-127 to 128).

flags -- As defined in resman.h.

RESULT

Error message as defined in resman.h.

EXAMPLE

```

result = BorrowArea(20,"MyPasswd","Doing something",0,MSGLOCK);

```

NOTES

An area should be entered (`EnterArea()`) before `BorrowArea` is used and the area must be released using `FreeArea`.

An application is allowed to borrow an area even if there are other users (`EnterArea()`) in the area.

Once an Area has been borrowed, all other `BorrowArea` calls will wait until the area is freed. Pending `BorrowAreas` will be honored on a First-In, First-Out basis.

Consider `BorrowArea()/FreeArea()` like the Amiga executive functions `Forbid()/Permit()`. Use `BorrowArea()/FreeArea()`

`_only_`
for short term locks. Otherwise, users might think that the BBS
has locked up.

Never try to `BorrowArea()` the same area without doing a

`FreeArea()`
between the `BorrowAreas`. Your program (and port) will
hang.

All of the `DLG.Library` functions that deal with message and file
areas do their own `BorrowArea/FreeArea`, so you `_must_` have freed
the area before using them.

BUGS

SEE ALSO

```
LockArea()
,
FreeArea()
,
EnterArea()
,
LeaveArea()
```

1.19 dlglibrary/BroadCast

NAME

`BroadCast` -- Broadcast a message

SYNOPSIS

```
result = Broadcast(ports,buffer,flags)
                A0    A1    D0
LONG Broadcast(char *,char *,UBYTE)
```

FUNCTION

`BroadCast` a message to port(s) using the `BroadCast Manager`.

INPUTS

`ports` -- String containing a list of ports to broadcast to
(example - "TR0TR1TL0").

`buffer` -- String to be broadcast (max 68 characters).

`flags` -- Misc. flags (see `broadcast.h` for #defines).

RESULT

Error message as defined in `broadcast.h`.

EXAMPLE

```
error = Broadcast("TR0TL0","Hi there");
```

NOTES

BUGS

SEE ALSO

```

    BCPend()
    ,
    BCResume()
    ,
    BCGet()
    ,
    HandleBCMsgs()
    ,
    BCMsg()

```

1.20 dlg.library/CallEditor

NAME

CallEditor -- Calls the User's editor and edit a file

SYNOPSIS

```

    result = CallEditor(reply, header, body, type, user, ram, port)
                    A0      A1      A2      D0      A3      D1      D2

```

LONG CallEditor(char *, char *, char *, char, struct USER_DATA *, struct Ram_File *, char *)

FUNCTION

Call the User's selected editor to edit a file

INPUTS

reply -- Reply Message Path/FileName or NULL if not a reply

header -- FidoNet Message header Path/FileName or NULL if not a fidonet message

body -- Message body file

type -- Message type (see msg.h)

user -- Address of the USER_DATA structure

ram -- Address of the Ram_File structure

port -- DLG Port that the application is running on

RESULT

0 = successful

-1 = unsuccessful (user aborted, editor not found)

EXAMPLE

NOTES

If the User's editor is invalid or is no longer valid for the user then the default is to use DLG:LineEdit.

Handles the execution of a local editor if on a local port and

DLGConfig:Batch/LocExtEditor exists.

Screens the resulting body file for bad language.

BUGS

SEE ALSO

1.21 dlg.library/Capitalize

NAME

Capitalize -- Capitalize a string

SYNOPSIS

Capitalize(string)

A0

void Capitalize(char *)

FUNCTION

Capitalizes a string (makes the first character of each word a capital letter).

INPUTS

string -- String to be capitalized.

RESULT

none

EXAMPLE

```
Capitalize("capitalize this");
```

NOTES

BUGS

SEE ALSO

Upper()

1.22 dlg.library/Cat

NAME

Cat -- To concatenate two files together

SYNOPSIS

result = Cat(file1, file2, joiner)

A0 D0 A1

long Cat(char *, char *, char *)

FUNCTION

To concatenate two files together. File2 is appended to File1 and the two files can be (optionally) separated by the joiner string. If

File1 doesn't exist, it is created. If File2 doesn't exist an error is returned.

INPUTS

file1 -- Path of file to be appended to.

file2 -- Path of file to append.

joiner -- Optional string that separates the two files (NULL if not used).

RESULT

total number of bytes added to file1.

EXAMPLE

```
size += Cat(msgbody, sigfile, "\015");
```

NOTES**BUGS****SEE ALSO**

```
AppendFile()  
,  
Copy()  
,  
FileCopy()
```

1.23 dlg.library/CD

NAME

CD -- Change Directory

SYNOPSIS

```
result = CD(directory)  
A0  
BOOL CD(char *)
```

FUNCTION

To change the default directory of the current CLI.

INPUTS

directory -- directory path

RESULT

TRUE if successful
FALSE if unsuccessful

EXAMPLE

```
result = CD("File:");
```

NOTES**BUGS**

SEE ALSO

1.24 dlg.library/ChainProgram

NAME

ChainProgram -- Sets up DLG to execute another program

SYNOPSIS

```
result = ChainProgram(program, port)
                        A0      A1
```

```
BOOL ChainProgram(char *, char *)
```

FUNCTION

Sets up DLG to execute another program when the current program exits.

INPUTS

program -- full path and filename of program to execute

port -- DLG Port (should be 4 characters) that the program
is running on.

RESULT

```
TRUE if successful
FALSE if unsuccessful
```

EXAMPLE

```
ChainProgram("DLG:Menu", port);
CloseLibrary(DLGBase);
exit(0);
```

NOTES

ChainProgram sets up DLG to execute another program after the current program exits. Generally, it should be used as shown above. If the current program was overlayed (and not chained), then when the current program ends it will return to the program that overlayed it and (generally) the chain request will be ignored.

When the current program ends, the new program will be run using the existing CLI. If there is a resident version of the program it will be used and any path will be ignored.

BUGS

SEE ALSO

OverlayProgram()

1.25 dlg.library/CheckUser

NAME

CheckUser -- Check whether a user exists

SYNOPSIS

```
result = CheckUser(name)
        A0
LONG CheckUser(char *)
```

FUNCTION

Checks whether a user has an account on the system.

INPUTS

name -- Name to be checked.

RESULT

1 if user has an account on the system
2 if a group exists by the specified name
0 if neither a user nor a group exists by the specified name

EXAMPLE

```
if(!CheckUser(UserName))
    printf("User or group [%s] doesn't exist\n",UserName);
```

NOTES

BUGS

SEE ALSO

1.26 dlg.library/ClearLine

NAME

ClearLine -- Flushes all characters from the input line.

SYNOPSIS

```
ClearLine()
```

```
void ClearLine(void)
```

FUNCTION

Waits for 4/10 of a second with no data on the input line.
Any data read is flushed.

INPUTS

none.

RESULT

none

EXAMPLE

```
ClearLine();
```

NOTES

BUGS

SEE ALSO

1.27 dlg.library/CloseGroup

NAME
CloseGroup -- End access of a group

SYNOPSIS

```
CloseGroup(fh)
    A0
void CloseGroup(BPTR)
```

FUNCTION

Ends access of a group that has been opened with
OpenGroup()
.

INPUTS

fh -- AmigaDOS FileHandle returned by
OpenGroup()
.

RESULT

none

EXAMPLE

```
CloseGroup(fh);
```

NOTES

BUGS

SEE ALSO

```
OpenGroup()
,
NextInGroup()
```

1.28 dlg.library/Clr

NAME

Clr -- Clear the screen

SYNOPSIS

```
Clr(ansi)
    D0
void Clr(UBYTE)
```

FUNCTION

Clears the user's screen (or prints two spaces if the user has screen
clears turned off).

INPUTS
ansi -- User's ANSI settings.

RESULT
none

EXAMPLE
Clr(User.Ansi_Flag);

NOTES

BUGS

SEE ALSO

1.29 dlg.library/Copy

NAME
Copy -- Copies one file to another

SYNOPSIS
Copy(source, dest)
A0 A1
long Copy(char *, char *)

FUNCTION
Copies all of a file into another file. Should the destination drive become full during the copy and the DLGConfig:Batch/DriveIsFull.batch exists, it will be executed.

INPUTS
source -- Full Path/Name of source file.

dest -- Full Path/Name of destination file.

RESULT
2 = Can't open destination file
1 = Can't open source file
0 = successful
-1 = output drive is full

EXAMPLE
Copy("T:File1", "T:File2");

NOTES

BUGS

SEE ALSO

FileCopy()
,
SmartRename()

1.30 dlg.library/CronEvent

NAME

CronEvent -- Send message to TPTCron

SYNOPSIS

```
result = CronEvent(messtype,time,command)
                D0      D1   A0
LONG CronEvent(UBYTE,ULONG,char *)
```

FUNCTION

Sends a message to TPTCron

INPUTS

```
messtype -- One of the following values, as defined in cron.h:
          #define ADDEVENT      1  -- Add an event
          #define DELEVENT      2  -- Delete an event
          #define LISTEVENTS    3  -- List the dynamic event list
          #define CRONEXIT      4  -- Shut down TPTCron
          #define WHENEVENT     5  -- Query next occurrence of event
          #define READFILE      6  -- Read a crontab file
          #define TABLIST       7  -- List the permanent event list
          #define CHANGEDIR     8  -- Change TPTCron's current
                                   directory

time      -- Time for event to be added (for ADDEVENT call), in minutes
           from current time.

command   -- Command to be added for ADDEVENT call.

           Command to be deleted (* and ? wildcards supported) for a
           DELEVENT call.

           Command to be queried (* and ? wildcards supported) for a
           WHENEVENT call.

           Crontab file to be read for a READFILE command.

           New directory for a CHANGEDIR call.
```

RESULT

Most commands return the following errors, as defined in cron.h:

```
#define CNOERR      0      -- Success
#define OUTFMEM     -1     -- Out of memory error
#define BADSYNTAX   -2     -- Invalid arguments
#define NOCRON      -3     -- TPTCron not currently active
#define TABNOTFOUND -4     -- Crontab file not found (READFILE)
#define NOEVENTS    -5     -- No events to list (LISTEVENTS)
#define DIRNOTFOUND -6     -- Directory not found (CHANGEDIR)
```

The DELEVENT command returns the number of events deleted.

The WHENEVENT command returns the number of minutes until the first match.

EXAMPLE

NOTES

BUGS

SEE ALSO

WhenEvent ()

1.31 dlg.library/DB

NAME

DB -- Output a debugging string

SYNOPSIS

```
result = DB(string)
          A0
BOOL DB(char *)
```

FUNCTION

Outputs a datestamped debugging string to standard output and waits for a second.

INPUTS

string -- String to be output.

RESULT

TRUE

EXAMPLE

```
DB("Some debugging");
```

NOTES

BUGS

SEE ALSO

1.32 dlg.library/DeActivatePort

NAME

DeActivatePort -- Deactivate a port

SYNOPSIS

```
result = DeActivatePort(port,passwd)
          A0  A1
LONG DeActivatePort(char *,char *)
```

FUNCTION

Removes a port from resource management.

INPUTS

```
port    -- Three-character port name.

passwd -- Password to lock port with (the port must be locked before it
        is deactivated to ensure that nobody else is using it).
```

RESULT

The result is an error message (see resman.h for #defines).

EXAMPLE

```
result = DeactivatePort("TR0","Deactivating");
```

NOTES

BUGS

SEE ALSO

ActivatePort()

1.33 dlg.library/DelArea

NAME

DelArea -- Delete an Area from one of the global area files

SYNOPSIS

```
result = DelArea(path, area)
           A0    D0
BOOL DelArea(char *, USHORT)
```

FUNCTION

Delete an Area from one of the global area files. The global area files are in the user's directory and contain an area list of short intergers. These are the GlobalAreas.file, GlobalAreas.msg and GlobalAreas.archive.

The area is deleted from the list for every occurrence.

INPUTS

```
path -- Complete path and filename of the global area file

area -- Area number to be deleted.
```

RESULT

```
TRUE  area successfully deleted
FALSE area not deleted (not found, file doesn't exist)
```

EXAMPLE

```
result = DelArea("User:Joe_Smith/GlobalAreas.msg", 10);
```

NOTES

BUGS

SEE ALSO

AddArea()

1.34 dlg.library/DelDir

NAME

DelDir -- Delete a directory, all of it's files and any subdirectories and their files.

SYNOPSIS

```
result = DelDir(path, user)
           A0    A1
BOOL DelDir(char *, struct USER_DATA *)
```

FUNCTION

Deletes a directory and all of it's files and any subdirectories and their files. Can optionally print progress and success failure messages to the current CLI output device.

INPUTS

path -- Complete path and filename of the directory to delete

user -- Address of the USER_DATA structure (optional) if message printing is required.

RESULT

TRUE directory successfully deleted

FALSE directory not deleted (not found, protected files, etc)

EXAMPLE

```
result = DelDir("User:Joe_Smith", &User); // Output progress to user
result = DelDir("User:Joe_Smith", NULL);  // No output to user
```

NOTES

BUGS

SEE ALSO

DirSize()

1.35 dlg.library/DeleteStruct

NAME

DeleteStruct -- Delete a structure from a file

SYNOPSIS

```
result = DeleteStruct(filename, structptr, structsize, fieldsize)
           A0          A1          D0          D1
LONG DeleteStruct(char *, char *, USHORT, USHORT)
```

FUNCTION

Deletes a structure from a sorted file on disk.

INPUTS

filename -- Filename to delete structure from. If the structure is the last in the file, the file will be deleted.

structptr -- Pointer to structure. Only the keyfield need be filled in.

structsize -- Size of structure.

fieldsize -- Size of keyfield (1st field) for sorting. This field MUST be a string.

RESULT

-1 if operation failed
0 if operation was successful

EXAMPLE

```
result = DeleteStruct("Structures.dat",mystructure,
                    sizeof(*mystructure),10);
```

NOTES

Common mistake #426: do not use strlen() to determine the size of the first field of the structure! strlen() only returns the length of the string UP TO THE TERMINATING NULL (\0) in the string! Either specify the correct length of the string, or use another method to find the size of the structure element.

BUGS

SEE ALSO

```
AddStruct ()
'
GetStruct ()
'
GetFirstStruct ()
```

1.36 dlg.library/DeScore

NAME

DeScore -- De-underscores a string

SYNOPSIS

```
DeScore(string)
A0
void DeScore(char *)
```

FUNCTION

Replaces all underscore characters '_' with spaces in a string. Useful for converting a user's directory name into a username.

INPUTS

```
string -- String to be descored.
```

```
RESULT  
none
```

```
EXAMPLE  
DeScore("John_Doe");
```

```
NOTES
```

```
BUGS
```

```
SEE ALSO
```

```
UnderScore()
```

1.37 dlg.library/DialogBatch

```
NAME
```

```
DialogBatch -- Execute a DLG batch file
```

```
SYNOPSIS
```

```
result = DialogBatch(path, User, Ram, Port)  
                A0    A1    A2    A3  
BOOL DialogBatch(char *, struct USER_DATA *, struct Ram_File *, char *)
```

```
FUNCTION
```

```
Executes a DLGBatch file.
```

```
INPUTS
```

```
path -- Full path and filename of DLG batchfile to execute
```

```
User -- USER_DATA structure.
```

```
Ram -- Ram_File structure of the user.
```

```
port -- DLG Port (should be 4 characters) that the program  
is running on.
```

```
RESULT
```

```
TRUE if successful  
FALSE if batch file not found
```

```
EXAMPLE
```

```
DialogBatch("DLGConfig:Batch/Login.DLGBatch", User, Ram, Port);
```

```
NOTES
```

```
BUGS
```

```
SEE ALSO
```

1.38 dlg.library/DirSize

NAME

DirSize -- Returns the number of bytes in the directory.

SYNOPSIS

```
result = DelDir(path)
          A0
LONG DirSize(char *)
```

FUNCTION

Returns the number of bytes used by the directory. It accounts for all files and subdirectories and their files. It also accounts for the directory entry required for each file and/or subdirectory.

INPUTS

path -- Complete path and filename of the directory

RESULT

size of the directory

EXAMPLE

```
result = DirSize("User:Joe_Smith");
```

NOTES

BUGS

SEE ALSO

DelDir()

1.39 dlg.library/DispBuffer

NAME

DispBuffer -- Display a buffer to the user

SYNOPSIS

```
result = DispBuffer(fh,buffer,screenpos,indent,
                   D0 A0      A1      D1
                   ibuf,header,breakbuf,User)
                   A2      D2      A3      D3
```

```
LONG DispBuffer(BPTR, char *, USHORT *, USHORT, char *,
                USHORT, char *, struct USER_DATA *)
```

FUNCTION

Displays a buffer to the user, formatted to their user settings. This routine will do word-wrap according to the user's screen width, will display more prompts at the appropriate places, and will even scroll the contents of the buffer leaving a specified number of lines at the top as a header.

INPUTS

fh -- AmigaDOS FileHandle to send output to.

buffer -- Buffer to be displayed. The buffer must be null terminated.

screenpos -- Address of an unsigned, short value that contains the current row output is at on the user's screen. This variable will be updated to hold the new row after the display has ended.

indent -- Optional number of characters to indent each line (except the first).

ibuf -- Optional buffer (use NULL for no buffer) to print instead of spaces for indenting. For example, an 'indent' of 3 and an 'ibuf' of " > " would be good for displaying the buffer as a quote for a message.

header -- Number of lines to be left at the top of the screen as a header. This number of lines will always stay the same, as the rest of the buffer scrolls or pages beneath.

breakbuf -- Buffer containing characters that, if received as input during the display, will cause the display to terminate. Usually, you want to have at least ^C "\003", and you might want other characters as well. For example, when DispBuffer() is used to display DLG menus, all of the letters for the current menu options are placed in the break buffer to allow for hotkeying during menu display.

User -- USER_DATA structure.

RESULT

Character used to break the display, if one of the 'breakbuf' characters was typed

0 if entire buffer was displayed
 -1 if output was stopped because user responded 'no' to a 'more' prompt
 -2 if entire buffer was displayed after user responded '=' to a 'more' prompt

'screenpos' is updated to reflect the new screen position.

EXAMPLE

```
DispBuffer(Output(), MyBuf, &screenpos, 0, NULL, 0, "\003", User);
```

NOTES

BUGS

SEE ALSO

```
DispForm()
,
DispMsg()
```

1.40 dlg.library/DispForm

NAME

DispForm -- Display a file with DLG's '%' switches

SYNOPSIS

```
result = DispForm(filename,DispUser,Trans,Ram,port)
                A0      A1      A2      A3  D0
```

```
BOOL DispForm(char *,struct USER_DATA *,struct USER_DATA *,
               struct Ram_File *,char *)
```

FUNCTION

Displays a file, interpreting DLG's '%' switches in the process.

```
DispBuffer()
    is used to do the main display job.
```

INPUTS

filename -- File to be displayed.

DispUser -- USER_DATA structure of the user that is seeing the display.

Trans -- USER_DATA structure to be used for translating '%' switches.

Ram -- Ram_File structure of the user that is seeing the display.

port -- Port the user is on.

RESULT

```
-1 if file couldn't be opened
Otherwise, the result of the call to
DispBuffer()
    is returned
```

EXAMPLE

```
DispForm("MyFile.txt",User,User,Ram,"TR0");
```

NOTES

BUGS

SEE ALSO

```
DispBuffer()
,
DispMsg()
```

1.41 dlg.library/DispMsg

NAME

DispMsg -- Displays the text of a message

SYNOPSIS

```
result = DispMsg(md,User)
           A0 A1
LONG DispMsg(struct MsgDisplay *,struct USER_DATA *)
```

FUNCTION

Displays the text of a message, ignores Kludge lines and does message quoting highlighting.

INPUTS

md -- MsgDisplay structure. The format of this structure is (from msg.h):

```
USHORT area          -- Number of area the message is in.

char *transarray    -- Translation matrix (or NULL).

char *passwd        -- Password to lock message area with.

char *filename      -- Name of file containing message.

USHORT *screenpos   -- Address of an unsigned short containing the
                    current screen position

LONG findex         -- Offset from the beginning of the message file
                    that the text begins (used to skip message
                    header). This should be 190 for a standard DLG
                    message, or, optionally (and better for future
                    upgrades), use sizeof(struct Msg_Header).

char *breakbuf      -- Break buffer to be passed to
                    DispBuffer()
                    .

ULONG flags         -- MSG_STRIPSB  if you want 'seen-by' lines
                    stripped
                    MSG_MSGAREA  if displaying from a Message Area
                    MSG_FILEAREA if displaying from a File Area
```

User -- USER_DATA structure.

RESULT

```
-1 if file couldn't be opened
Otherwise, the result of the call to
    DispBuffer()
    is returned
```

EXAMPLE

NOTES

Do not set MSG_STRIPSB unless in an Echo area, otherwise the message may not display.

Assumes the size of the message header is 6 lines.

BUGS

SEE ALSO

```
DispBuffer()
,
DispForm()
```

1.42 dlg.library/DLGBinSearch

NAME

DLGBinSearch -- Search for a structure in a sorted array

SYNOPSIS

```
result = DLGBinSearch(array,structptr,structsize,fieldsize,elements)
                    A0    A1        D0          D1          D2
char *DLGBinSearch(char *,char *,USHORT,USHORT,USHORT)
```

FUNCTION

Searches for a structure in a sorted array.

INPUTS

```
array          -- Pointer to memory block to be searched.

structptr     -- Pointer to structure to find (keyfield must be filled
                in). This can just be a pointer to the keyfield, it need
                not be the entire structure.

structsize    -- Size of structure.

fieldsize     -- Length of keyfield (1st field). This field must be a
                string, and the array of structures must be sorted by
                the keyfield.

elements      -- Number of structures in the array.
```

RESULT

Pointer to structure, if found, otherwise NULL. Note that the pointer is to the actual structure in the array, not a copy of it.

EXAMPLE

```
mystruct = DLGBinSearch(structures,"Keyfield",sizeof(*mystruct),20,32);
if(!mystruct) printf("Couldn't find it\n");
```

NOTES

BUGS

SEE ALSO

```
DLGSearch()
```

1.43 dlg.library/DLGGetSer

NAME

DLGGetSer -- Take control of the serial.device for a port

SYNOPSIS

```
result = DLGGetSer(port,protocol)
                A0   D0
struct DLGSerInfo *DLGGetSer(char *,char)
```

FUNCTION

Allows an application to take direct control of the serial.device for a port in order to do a file transfer.

INPUTS

port -- Port to be used.

protocol -- Single character name of protocol.

RESULT

DLGSerInfo structure, or NULL if operation failed. The structure is as follows (as defined in dlgproto.h):

```
char port[4]          -- The port being used.
struct IOExtSer *read  -- The IO message for reading.
struct IOExtSer *write -- The IO message for writing.
struct MsgPort *readbak -- Backup copy of message port for reading
                        (shouldn't be touched by application).
struct MsgPort *writebak -- Backup copy of message port for writing
                        (shouldn't be touched by application).
char titlebak[71]     -- Backup copy of screen/window title
                        (shouldn't be touched by application).
char title[71]        -- New title to be used (shouldn't be
                        touched by application).
unsigned char flags   -- Not currently used.
```

EXAMPLE

```
serinfo = DLGGetSer("TR0","X");
```

NOTES

BUGS

SEE ALSO

```
DLGReleaseSer()
,
DLGProtoStatus()
```

1.44 dlg.library/DLGPatternMatch

NAME

DLGPatternMatch -- Check if a string matches a pattern

SYNOPSIS

```
result = DLGPatternMatch(pat, str)
                        A0 A1
BOOL DLGPatternMatch(char *, char *)
```

FUNCTION

Checks whether a string matches a wildcard pattern.

INPUTS

pat -- Pattern to be used. '*' (match any number of characters) and '?' (match any single character) are supported wildcards.

str -- String to match against the pattern.

RESULT

TRUE if the string matches the pattern
FALSE if it doesn't

EXAMPLE

```
if(!DLGPatternMatch("*.c", "myprog.c"))
    printf("Hmmm, that should have matched\n");
```

NOTES

BUGS

SEE ALSO

1.45 dlg.library/DLGProtoStatus

NAME

DLGProtoStatus -- Update the status of a transfer

SYNOPSIS

```
DLGProtoStatus(dsi, fsize, bytes, msg)
                A0 D0 D1 A1
void DLGProtoStatus(struct DLGSerInfo *, ULONG, ULONG, char *)
```

FUNCTION

Updates the status information of a file transfer in the title bar.

INPUTS

dsi -- DLGSerInfo structure returned by
DLGGetSer()

fsize -- Size of file being transferred (or 0 to not update the file size).

bytes -- Number of bytes transferred so far (or 0 to not update this field).

msg -- A message to be displayed temporarily instead of the other information (or NULL for no message).

RESULT
none

EXAMPLE
DLGProtoStatus(serinfo,10367,1024,NULL);

NOTES

BUGS

SEE ALSO

```
DLGGetSer()
,
DLGReleaseSer()
```

1.46 dlg.library/DLGQuery

NAME
DLGQuery -- Get input from the user

SYNOPSIS
result = DLGQuery(query,ui)
 A0 A1
LONG DLGQuery(struct Query *,struct UserInfo *)

FUNCTION
DLG's low-level user input routine.

INPUTS
query -- Query structure (defined in input.h). Has following elements:

```
char *prompt -- Prompt to be displayed before input is
                requested.

char *template -- Template for input. Underscores "_" represent
                input, other characters are printed. For
                example, "(____) ___-____" would make a good
                template for entering phone numbers. This
                field is also used to hold the array of
                possible input strings when in 'guess' mode.

char *string -- Buffer to put the input in.

char *defstring -- Default input to use if user just hits return.
                This field is also used as the initial item
                when in 'guess' mode.

char *valid -- String containing valid input characters. For
```

example, "0123456789-" would get only signed, numeric input.

USHORT length -- Maximum number of characters to be placed in the 'string' field. This field also holds the number of characters per input item when in 'guess' mode.

USHORT typelength -- Maximum number of characters the user is allowed to type (you often want the user to be able to type more than you want to read so that they can type command stacks). This field holds the number of items in the input array when in 'guess' mode.

ULONG flags -- Flags as defined in input.h.

ui -- UserInfo structure as defined in input.h.

RESULT

Number of characters read, or item selected if in 'guess' mode.

EXAMPLE

NOTES

'Guess' mode needs more explanation. If the input you want is one element of a sorted array of strings (such as user names), you can pass in a pointer to a block of memory holding these input strings. This pointer is placed in the 'template' field of the Query structure. The 'length' field holds the length of each input string. The 'typelength' field holds the number of input items there are. The 'defstring' holds the number of the input string to start at. The user will then be able to cursor up and down through the list of input items, and the input will automatically be completed as the user types. The return value is the number of the string the user selected.

BUGS

SEE ALSO

```
BoolQuery()
'
IntQuery()
```

1.47 dlg.library/DLGReleaseSer

NAME

DLGReleaseSer -- Release a hold on the serial.device for a port.

SYNOPSIS

```
DLGReleaseSer(dsi)
    A0
void DLGReleaseSer(struct DLGSerInfo *)
```

FUNCTION

Releases the hold an application has on the serial.device for a port after a file transfer is finished.

INPUTS

dsi -- DLGSerInfo structure returned by
DLGGetSer()
.

RESULT

none

EXAMPLE

```
DLGReleaseSer(serinfo);
```

NOTES

BUGS

SEE ALSO

```
DLGGetSer()  
,  
DLGProtoStatus()
```

1.48 dlg.library/DLGSearch

NAME

DLGSearch -- Search for a structure in an array

SYNOPSIS

```
result = DLGSearch(array,structptr,structsize,fieldsize,elements)  
          A0      A1          D0          D1          D2  
char *DLGSearch(char *,char *,USHORT,USHORT,USHORT)
```

FUNCTION

Searches for a structure in an array (the array need not be sorted).

INPUTS

array -- Pointer to memory block to be searched.

structptr -- Pointer to structure to find (keyfield must be filled in). This can just be a pointer to the keyfield, it need not be the entire structure.

structsize -- Size of structure.

fieldsize -- Length of keyfield (1st field). This field must be a string.

elements -- Number of structures in the array.

RESULT

Pointer to structure, if found, otherwise NULL. Note that the pointer is to the actual structure in the array, not a copy of it.

EXAMPLE

```
mystruct = DLGBinSearch(structures,"Keyfield",sizeof(*mystruct),20,32);
if(!mystruct) printf("Couldn't find it\n");
```

NOTES

BUGS

SEE ALSO

DLGBinSearch()

1.49 dlg.library/Draw_Line

NAME

Draw_Line -- Draw a line of dashes ('-').

SYNOPSIS

```
Draw_Line(size)
    D0
void Draw_Line(UBYTE)
```

FUNCTION

Draws a line followed by a newline to the output device of the CLI.

INPUTS

size -- size of the line including the newline

RESULT

none

EXAMPLE

```
Draw_Line(20);
```

NOTES

BUGS

SEE ALSO

SDraw_Line()

1.50 dlg.library/EnterArea

NAME

EnterArea -- Enter an area

SYNOPSIS

```
result = EnterArea(area,flags)
    D0  D1
LONG EnterArea(USHORT,UBYTE)
```


FUNCTION

Enters an area, letting the resource manager know that there is someone in the area and that it cannot be locked with
 LockArea()
 .

INPUTS

area -- Area to enter.

flags -- As defined in resman.h.

RESULT

Error message as defined in resman.h.

EXAMPLE

```
result = EnterArea(20,MSGLOCK);
```

NOTES

BUGS

SEE ALSO

```
    LeaveArea()  
    ,  
    BorrowArea()  
    ,  
    LockArea()  
    ,  
    FreeArea()
```

1.51 dlg.library/Exists

NAME

Exists -- Check if a file or directory exists

SYNOPSIS

```
result = Exists(filename)  
          A0  
BOOL Exists(char *)
```

FUNCTION

Checks to see if a file exists on disk.

INPUTS

filename -- Full path of file or directory to search for.

RESULT

TRUE if file exists
FALSE if it doesn't

EXAMPLE

```
if(!Exists(Filename)) printf("[%s] doesn't exist\n",Filename);
```

NOTES

BUGS

SEE ALSO

1.52 dlg.library/ExistsGlobalArea

NAME

ExistsGlobalArea -- Checks for an Area in a user's global area file

SYNOPSIS

```
result = ExistsGlobalArea(path, area)
                        A0    D0
BOOL ExistsGlobalArea(char *, USHORT)
```

FUNCTION

Checks for an Area in one of the global area files. The global area files are in the user's directory and contain an area list of short intergers. These are the GlobalAreas.file, GlobalAreas.msg and GlobalAreas.archive.

INPUTS

path -- Full path and filename of the global area file

area -- Area number to be found.

RESULT

TRUE area found
FALSE area not found (not found, or file doesn't exist)

EXAMPLE

```
result = ExistsGlobalArea("User:Joe_Smith/GlobalAreas.msg", 10);
```

NOTES

While one would be tempted to check for the file first (see
Exists()
) ,

keep in mind that if the file does NOT exist, the user has no global areas defined. Therefore, your code will be faster if you just use this function and not worry about whether the file is there or not.

BUGS

SEE ALSO

```
AddArea ()
'
DelArea ()
```

1.53 dlg.library/FileCopy

```

NAME
FileCopy -- Copy a file

SYNOPSIS
result = FileCopy(afh,afh,iofs,oofs,size)
                A0 A1 D0 D1 D2
LONG FileCopy(BPTR,BPTR,ULONG,ULONG,ULONG)

FUNCTION
Copy all or part of a file onto all or part of another file.

INPUTS
afh -- AmigaDOS FileHandle of an open input file.

afh -- AmigaDOS FileHandle of an open output file.

iofs -- Offset to begin copying from in the input file.

oofs -- Offset to begin copying to in the output file.

size -- Number of bytes to copy.

RESULT
0 successful
-1 output drive is full

EXAMPLE
FileCopy(in,out,0,0,size);

NOTES

BUGS

SEE ALSO
Copy()
,
SmartRename()

```

1.54 dlg.library/FileSize

```

NAME
FileSize -- Get the size of a file

SYNOPSIS
result = FileSize(filename,size)
                A0 A1
LONG FileSize(char *,ULONG *)

FUNCTION
Gets the size of a file on disk.

INPUTS
filename -- Full path to file.

```

size -- Address of long to put file size into.

RESULT

-1 if operation failed
0 if operation was successful

EXAMPLE

```
if(FileSize("myfile",&size)==-1) printf("Something got messed up\n");
else printf("The size is [%lu]\n",size);
```

NOTES

BUGS

SEE ALSO

1.55 dlgl.library/FreeArea

NAME

FreeArea -- Free a lock on an area

SYNOPSIS

```
result = FreeArea(area,passwd,flags)
           D0   A0   D1
LONG FreeArea(USHORT,char *,UBYTE)
```

FUNCTION

Frees a lock that an application holds on an area.

INPUTS

area -- Number of the area to be freed

passwd -- Password the area was locked with.

flags -- As defined in resman.h.

RESULT

Error message as defined in resman.h

EXAMPLE

```
result = FreeArea(20,"MyPasswd",MSGLOCK);
```

NOTES

BUGS

SEE ALSO

```
BorrowArea()
,
LockArea()
,
EnterArea()
,
```

LeaveArea()

1.56 dlg.library/FreeAreaInfo

NAME

FreeAreaInfo -- Free AreaInfo structure

SYNOPSIS

```
result = FreeAreaInfo(istruct)
                        A0
LONG FreeAreaInfo(struct DLGAreaInfo *)
```

FUNCTION

Frees an DLGAreaInfo structure obtained with GetAreaInfo().

INPUTS

istruct -- DLGAreaInfo structure obtained with GetAreaInfo().

RESULT

Error message as defined in resman.h.

EXAMPLE

```
result = FreeAreaInfo(istruct);
```

NOTES

In previous versions of DLG's includes and autodocs, the DLGAreaInfo structure was called the AreaInfo structure, which was a conflict with an AmigaDOS system structure.

BUGS

SEE ALSO

GetAreaInfo()

1.57 dlg.library/FreeMenu

NAME

FreeMenu -- Free a menu

SYNOPSIS

```
result = FreeMenu(port, name, passwd)
                        A0  A1  A2
LONG FreeMenu(char *, char *, char *)
```

FUNCTION

Free a menu locked with
LockMenu()
.

INPUTS

port -- Port the application is running on.

name -- Name of menu.

passwd -- Password menu was locked with.

RESULT

Error message as defined in resman.h.

EXAMPLE

```
result = FreeMenu("TR0", "main", "mypasswd");
```

NOTES

This function is not ready for use by third-party developers.

BUGS

SEE ALSO

```
LockMenu()
,
PurgeMenu()
```

1.58 dlg.library/FreePort

NAME

FreePort -- Free a lock on a port

SYNOPSIS

```
result = FreePort(port, passwd)
           A0   A1
LONG FreePort(char *, char *)
```

FUNCTION

Frees a lock that an application currently holds on a port.

INPUTS

port -- Three-character port name.

passwd -- Password the port was previously locked with.

RESULT

The result is an error message (see resman.h for #defines).

EXAMPLE

```
FreePort("TR0", "MyPassword");
```

NOTES

BUGS

SEE ALSO

```
LockPort()
,
ImmedLockPort()
```

```

    ,
    TransferPortLock()

```

1.59 dlg.library/FreePortInfo

NAME

FreePortInfo -- Free information about a port

SYNOPSIS

```

result = FreePortInfo(istruct)
                    A0
LONG FreePortInfo(struct PortInfo *)

```

FUNCTION

Frees a PortInfo structure obtained with
GetPortInfo()

.

INPUTS

istruct -- PortInfo structure obtained with
GetPortInfo()

.

RESULT

Error message as defined in resman.h.

EXAMPLE

```

result = FreePortInfo(&istruct);

```

NOTES

BUGS

SEE ALSO

GetPortInfo()

1.60 dlg.library/FreeResource

NAME

FreeResource -- Free a miscellaneous resource

SYNOPSIS

```

result = FreeResource(name,passwd)
                    A0  A1
LONG FreeResource(char *,char *)

```

FUNCTION

Frees a miscellaneous named resource locked with
LockResource()

.

INPUTS

```
name    -- Name of resource to be freed.

passwd  -- Password resource was locked with.
```

RESULT

```
Error message as defined in resman.h.
```

EXAMPLE

```
result = FreeResource("MyResource", "MyPasswd");
```

NOTES

BUGS

SEE ALSO

```
LockResource()
```

1.61 dlg.library/FreeResReport

NAME

```
FreeResReport -- Free a resource report
```

SYNOPSIS

```
result = FreeResReport(lst)
                        A0
LONG FreeResReport(struct List *)
```

FUNCTION

```
Frees the resource report structure returned by
GetResReport()
.
```

INPUTS

```
lst -- The List structure returned by
GetResReport()
.
```

RESULT

```
Error message as defined in resman.h.
```

EXAMPLE

```
error = FreeResReport(resrep);
```

NOTES

```
The List structure is an AmigaDOS double-linked list as described in
RKM: Libraries.
```

BUGS

SEE ALSO

```
GetResReport()
```

1.62 dlg.library/GetAreaInfo

NAME

GetAreaInfo -- Get information about a message/file area

SYNOPSIS

```
result = GetAreaInfo(istruct, flags)
                        A0      D0
LONG GetAreaInfo(struct DLGAreaInfo *, UBYTE)
```

FUNCTION

Gets information about a message or file area from the resource manager.

INPUTS

istruct -- DLGAreaInfo structure to be filled in. This structure is as follows (as defined in resman.h):

USHORT area -- Area to get info about (must be filled in).

char *passwd -- Password area is locked with (filled in by the resource manager).

char *reason -- Reason the area is locked (filled in by the resource manager).

char priority -- Priority of the lock (filled in by the resource manager).

UBYTE users -- Number of users in the area (filled in by the resource manager).

flags -- MSGLOCK for a message area, FILELOCK for a file area.

RESULT

Error message as defined in resman.h.
The DLGAreaInfo structure is filled in.

EXAMPLE

```
result = GetAreaInfo(&istruct, MSGLOCK);
```

NOTES

BUGS

SEE ALSO

FreeAreaInfo()

1.63 dlg.library/GetChar

NAME

GetChar -- Read a character from the user

SYNOPSIS

```
result = GetChar()
```

```
char GetChar(void)
```

FUNCTION

Reads a character from the user on standard input.

INPUTS

none

RESULT

The character.

EXAMPLE

```
c = GetChar();
```

NOTES

BUGS

SEE ALSO

```
PutChar()  
,  
ReadChar()
```

1.64 dlg.library/GetComment

NAME

GetComment -- Get a file's comment

SYNOPSIS

```
result = FileSize(filename, comment)  
                A0      A1  
LONG FileSize(char *,char *)
```

FUNCTION

Gets the comment of a file on disk.

INPUTS

```
filename -- Full path to file.
```

```
comment  -- Address of string to put file comment into.
```

RESULT

```
-1 if operation failed  
0 if operation was successful
```

EXAMPLE

```
GetComment("myfile", comment);
```

NOTES

BUGS

SEE ALSO

1.65 dlg.library/GetComputerType

NAME

GetComputerType -- Get the name of a computer type

SYNOPSIS

```
result = GetComputerType(number, string)
                        D0      A0
LONG GetComputerType(SHORT, char *)
```

FUNCTION

Gets the name of the computer type corresponding to a particular number

INPUTS

number -- The number of the computer type.

string -- String to be filled in (should be 36 characters).

RESULT

The number of the computer type, or 0 if the computer type file couldn't be opened.

EXAMPLE

```
num = GetComputerType(User->Computer_Type, buf);
```

NOTES

BUGS

SEE ALSO

1.66 dlg.library/GetDevName

NAME

GetDevName -- Find out what port the user is on

SYNOPSIS

```
result = GetDevName(port)
                        A0
LONG GetDevName(char *)
```

FUNCTION

Determines what port the application is running on.

INPUTS

port -- Port string to be filled in (should be 4 characters).

RESULT

0 if port was found

-1 if port could not be determined

EXAMPLE

```
if (GetDevName(port)==-1) printf("Unable to determine port\n");
```

NOTES

BUGS

SEE ALSO

1.67 dlg.library/GetFileDate

NAME

GetFileDate -- Get the date of a file

SYNOPSIS

```
result = GetFileDate(filename, date)
                        A0          A1
BOOL GetFileDate(char *, LONG *)
```

FUNCTION

Gets the date of a file on disk.

INPUTS

filename -- Full path to file.

date -- Address of long to put file date into.

RESULT

0 if operation failed
1 if operation was successful

EXAMPLE

```
GetFileDate("myfile", &date);
```

NOTES

BUGS

SEE ALSO

1.68 dlg.library/GetFirstStruct

NAME

GetFirstStruct -- Get the first structure from a file

SYNOPSIS

```
result = GetFirstStruct(filename, structptr, structsize)
                        A0          A1          D0
LONG GetFirstStruct(char *, char *, ULONG)
```

FUNCTION

Gets the first structure from a file on disk.

INPUTS

filename -- Filename to get structure from.

structptr -- Pointer to memory area to place structure information in.

structsize -- Size of structure.

RESULT

-1 if operation failed
0 if operation was successful

EXAMPLE

```
result = GetFirstStruct("Structures.dat",mystructure,
                        sizeof(*mystructure));
```

NOTES

BUGS

SEE ALSO

```
GetStruct()
,
AddStruct()
,
DeleteStruct()
```

1.69 dlg.library/GetHiLowFPointers

NAME

GetHiLowFPointers -- Get the high and low pointers for a file area

SYNOPSIS

```
GetHiLowFPointers(area,username,low,high,pswd)
                D0  A0          A1  A2  A3
void GetHiLowPointers(USHORT,char *,LONG *,LONG *,char *)
```

FUNCTION

Gets the high and low file pointers for a file area.

INPUTS

area -- Number of the area (PVTAREA for a user's private area)

username -- Underscored name of the user, if getting pointers for a private area. Otherwise this field should be NULL.

low -- Pointer to a long value to store low pointer in.

high -- Pointer to a long value to store high pointer in.

pswd -- Password to lock area with when reading pointers file.

RESULT

none

EXAMPLE

```
GetHiLowFPointers (PVTAREA, "John_Doe", &low, &high, "pswd");
```

NOTES

This function puts a lock on the file area, so do not use it on an area that is already locked by the same application, or the program will hang up waiting for itself to release the area so it can lock it.

BUGS

SEE ALSO

```
PutHiLowFPointers ()
,
GetHiLowPointers ()
,
PutHiLowPointers ()
```

1.70 dlgl.library/GetHiLowPointers

NAME

GetHiLowPointers -- Get the high and low pointers for a message area

SYNOPSIS

```
GetHiLowPointers (area, username, low, high, pswd)
                D0  A0      A1  A2  A3
void GetHiLowPointers (USHORT, char *, LONG *, LONG *, char *)
```

FUNCTION

Gets the high and low message pointers for a message area.

INPUTS

```
area      -- Number of the area (PVTAREA for a user's private area)

username  -- Underscored name of the user, if getting pointers for
           a private area. Otherwise this field should be NULL.

low       -- Pointer to a long value to store low pointer in.

high      -- Pointer to a long value to store high pointer in.

pswd      -- Password to lock area with when reading pointers file.
```

RESULT

none

EXAMPLE

```
GetHiLowPointers (PVTAREA, "John_Doe", &low, &high, "pswd");
```

NOTES

This function puts a lock on the file area, so do not use it on an area that is already locked by the same application, or the program

will hang up waiting for itself to release the area so it can lock it.

BUGS

SEE ALSO

```
PutHiLowPointers()  
,  
GetHiLowFPointers()  
,  
PutHiLowFPointers()
```

1.71 dlg.library/GetLang

NAME

GetLang -- Get the language information for a port

SYNOPSIS

```
result = GetLang(port)  
A0  
struct LangStruct *GetLang(char *)
```

FUNCTION

Gets the language information for a port.

INPUTS

port -- Port to be used.

RESULT

A LangStruct, which is formatted as follows (defined in resman.h):

```
char name[21]    -- Name of the language.  
  
char **strings  -- Array of pointers to the language strings (as  
                  read from the language file on disk).  
  
short numstrings -- Number of language strings.
```

EXAMPLE

```
lstruct = GetLang("TR0");
```

NOTES

BUGS

SEE ALSO

```
LoadLang()
```

1.72 dlg.library/GetLevel

NAME

GetLevel -- Get the level of a user.

SYNOPSIS

```
result = GetLevel(name)
          A0
SHORT GetLevel(char *)
```

FUNCTION

Returns the access level of a user. The user name may be underscored or not. If the user can not be found a level of 257 is returned.

INPUTS

name -- User name

RESULT

User level or 257

EXAMPLE

```
level = GetLevel("Joe Smith");
```

NOTES

BUGS

SEE ALSO

1.73 dlg.library/GetOrigin

NAME

GetOrigin -- Get the origin address of a message

SYNOPSIS

```
result = GetOrigin(message, zone, net, node, point)
          A0      A1      A2      A3      D0
BOOL GetLevel(char *, SHORT *, SHORT *, SHORT *, SHORT *)
```

FUNCTION

Find the origin address of a message. The message is first searched for an origin line. Then the kludge lines (FMPT/INTL) are searched for.

INPUTS

message -- Full path and filename of the message

zone -- Address of SHORT to receive the zone

net -- Address of SHORT to receive the net

node -- Address of SHORT to receive the node

point -- Address of SHORT to receive the point

RESULT

TRUE if a origin address was found
 FALSE if no origin was found

EXAMPLE

```
if (!GetOrigin("MSG:1/5403.msg", &zone, &net, &node, &point))
    printf("Can't find origin address\n");
```

NOTES

BUGS

SEE ALSO

1.74 dlg.library/GetPath

NAME

GetPath -- Get the path of a file or filearea

SYNOPSIS

```
result = GetPath(path, area, carea, file)
           A0    D0    A1    A2
```

LONG GetLang(char *, SHORT, struct Msg_Area *, char *)

FUNCTION

Gets the path of a file or filearea.

If no filename is provided, it returns (in path) the path of the filearea. The path is either FILE:<area>/ or the path defined in the area's definition as the ALT area.

If a filename is provided, the file is found (either in the default path, alt path, or in one of the DLGConfig:Misc/FilePaths.BBS paths). If the file can not be found an error is returned.

INPUTS

path -- Pointer to the returned path.

area -- Area of the file.

carea -- Current area info structure. (optional)

file -- Pointer to a filename. (optional)

RESULT

-1 if failed (bad area #, file not found, etc).
 0 default (root) path returned.
 1 alternate path returned.
 2 global file path returned.

EXAMPLE

```
// Get the path for a file area, with area definition.
result = GetPath(path, area, carea, NULL);

// Get the path for a file area, with no area definition
result = GetPath(path, area, NULL, NULL);

// Get the path for a file, with area definition
result = GetPath(path, area, carea, filename);
```

NOTES

If your program has loaded a file area definition, pass it in carea. It's faster to use it than have the function read the definition from disk.

BUGS

SEE ALSO

1.75 dlg.library/GetPortInfo

NAME

GetPortInfo -- Get information about a port

SYNOPSIS

```
result = GetPortInfo(istruct)
                        A0
LONG GetPortInfo(struct PortInfo *)
```

FUNCTION

Gets information about the status of a port from the resource manager.

INPUTS

istruct -- PortInfo structure to be filled in. The structure has the following format (as defined in resman.h):

```
char *port      --      Port to get info about (must be filled in).

char *passwd    --      Password port is locked with (filled in by the
                        resource manager).

char *reason    --      Reason port is locked (filled in by the
                        resource manager).

char priority   --      Priority of lock (filled in by the resource
                        manager).

char *breakcommand -- Command to break lock (filled in by the
                        resource manager).
```

RESULT

Error message as defined in resman.h.

EXAMPLE

```
result = GetPortInfo(&istruct);
```

NOTES

BUGS

SEE ALSO

FreePortInfo()

1.76 dlg.library/GetResReport

NAME

GetResReport -- Get information about many resources

SYNOPSIS

```
result = GetResReport()
```

```
struct List *GetResReport(void)
```

FUNCTION

Gets information about ports, message areas, file areas, miscellaneous locks, and menu sets.

INPUTS

none

RESULT

Exec List structure that contains a node for each resource currently being managed by the resource manager. The nodes are ResRepNode structures (defined in resman.h) which have the following structure:

```
struct Node node --          Node structure for queuing. The kind of
                             node (NODE_MISC, NODE_PORT, NODE_MAREA,
                             NODE_FAREA, or NODE_MENU) is in the
                             ln_Type field.

char *bgcommand --          Background command (for Port nodes).

char *language --           Loaded language (for Port nodes).

char *menu --               Current menu (for Port nodes).

UBYTE users --              Number of users (for Area nodes).

struct List *activelocks -- List of active locks.

struct List *pendinglocks -- List of pending locks.
```

The lock lists are lists of LockRepNode structures (defined in resman.h) which have the following structure:

```

struct Node node -- Node structure for queuing. The ln_Type field
                    contains the type of the node this list of
                    locks is for.

char *reason -- Reason for lock.

UBYTE type -- Type of lock (MSGLOCK or FILELOCK, PENDLOCK (or
                    not), QUICKLOCK (or not), WRITELOCK (or not)).

char *breakcommand -- Break command to release lock (for Port nodes).

```

EXAMPLE

```
resrep = GetResReport();
```

NOTES

Consult RKM: Libraries for more information on the AmigaDOS dual linked lists, thier structure, and how to handle them.

BUGS**SEE ALSO**

FreeResReport()

1.77 dlg.library/GetStruct

NAME

GetStruct -- Get a structure from a file

SYNOPSIS

```

result = GetStruct(filename,structptr,structsize,fieldsize)
                    A0      A1      D0      D1
LONG GetStruct(char *,char *,USHORT,USHORT)

```

FUNCTION

Reads a particular structure from a file on disk.

INPUTS

```

filename -- File to read structure from.

structptr -- Pointer to memory area to place structure information
            in. The keyfield must be filled in.

structsize -- Size of structure.

fieldsize -- Size of keyfield (1st field) for sorting. This field
            must be a string.

```

RESULT

```

-1 if operation failed
0 if operation was successful

```

EXAMPLE

```

result = GetStruct("Structures.dat",mystructure,
                  sizeof(*mystructure),10);

```

NOTES

BUGS

SEE ALSO

```

    GetFirstStruct ()
    ,
    AddStruct ()
    ,
    DeleteStruct ()

```

1.78 dlg.library/HandleBCMsgs

NAME

HandleBCMsgs -- Display all pending BroadCast messages

SYNOPSIS

```

    result = HandleBCMsgs(port)
                    A0
    LONG HandleBCMsgs(char *)

```

FUNCTION

Displays all pending BroadCast messages with a beep before each.

INPUTS

port -- Port to display messages for.

RESULT

Error message as defined in broadcast.h.

EXAMPLE

```

    result=HandleBCMsgs("TR0");

```

NOTES

BUGS

SEE ALSO

```

    BCGet ()
    ,
    BCPend ()
    ,
    BCResume ()
    ,
    BroadCast ()
    ,
    BCMsg ()

```

1.79 dlg.library/ImmedLockPort

NAME

ImmedLockPort -- Lock a port with an "immediate" lock

SYNOPSIS

```
result = ImmedLockPort(port,passwd,reason,pri,bc)
                    A0  A1    A2    D0  A3
LONG ImmedLockPort(char *,char *,char *,char, char *)
```

FUNCTION

Locks a port with an "immediate" lock. This means that the function will return immediately, regardless of whether a lock was obtained. This function will not wait to get a lock if the port is locked by another application.

INPUTS

port -- Three-character port name.

passwd -- Password to lock port with.

reason -- Descriptive reason for lock.

pri -- Priority of lock (-127 to 128). Negative priority locks can be overridden by higher priority locks.

bc -- Break command - a command to be executed that will break the application if a higher priority lock is requested.

RESULT

The result is an error message (see resman.h for #defines).

EXAMPLE

```
result = ImmedLockPort("TR0","MyPasswd","Doing important stuff",-1,
                    "MyBreakProgram");
```

NOTES

BUGS

SEE ALSO

```
FreePort()
,
LockPort()
,
TransferPortLock()
```

1.80 dlg.library/ImportPublicMsg

NAME

ImportPublicMsg -- Import a message into a DLG message area

SYNOPSIS

```
result = ImportPublicMsg(header,body,areainfo,passwd)
                    A0    A1    A2    A3
```

```
LONG ImportPublicMsg(struct Msg_Header *,char *,
                    struct Msg_Area *,char *)
```

FUNCTION

Imports a message into a DLG message area. This routine should be used to bring in a message that came in from another system via some kind of network.

INPUTS

```
header    -- Fidonet message header structure (see msg.h for details).

body      -- Null-terminated block of text that makes up the body of the
           message. This text should be in standard, fidonet format
           as specified by FTS-0001.

areainfo  -- Msg_Area structure of the area to place the message in.
           This structure can be obtained by using
           ReadArea()
           .

pswd      -- Password to lock the area with while the message is being
           written.
```

RESULT

The number the message was assigned in the area, or FALSE if the operation failed

EXAMPLE

```
num = ImportPublicMsg(&header,bodytext,&area,"Importing");
```

NOTES

BUGS

SEE ALSO

```
SendPublicMsg()
,
SendPrivateMsg()
,
SendRawMsg()
,
KillMsg()
```

1.81 dlg.library/Inform

NAME

Inform -- Inform a user of something

SYNOPSIS

```
result = Inform(username,buffer,port,flags)
           A0      A1      A2      D0
BOOL Inform(char *,char *,char *,UBYTE)
```

FUNCTION

Informs a user of something by broadcasting them a message or writing to their event log if they are not online.

INPUTS

username -- Name of user, or "ALL" for all user's online.

buffer -- String to be sent as a message.

port -- Port application is running on (so that the port sending the message doesn't get a copy of a message to "ALL").

flags -- BCIMPORTANTMSG if the message is very important and should be sent even if messages are pending on a port.

RESULT

TRUE if operation is successful
FALSE if function failed

EXAMPLE

```
Inform("ALL",NULL,"System going down in 2 minutes");
```

NOTES

Currently limited to 26 users online at once, otherwise an overflow will occur.

BUGS

SEE ALSO

WriteEvent()

1.82 dlg.library/IntQuery

NAME

IntQuery -- Get an integer value from the user

SYNOPSIS

```
result = IntQuery(query,lower,upper,def,ui)
           A0      D0      D1      D2  A1
LONG IntQuery(char *,SHORT,SHORT,SHORT,struct UserInfo *)
```

FUNCTION

Prompts the user for an integer value.

INPUTS

query -- String to be displayed before input is requested.

lower -- Lower limit on numeric input.

upper -- Upper limit on numeric input.

def -- Default value to be used if user just hits return.

ui -- UserInfo structure as defined in input.h.

RESULT

Number the user typed.

EXAMPLE

```
number = IntQuery("Type a number between 1 and 5: ",1,5,3,MyUserInfo);
```

NOTES

BUGS

SEE ALSO

```
BoolQuery()
'
DLGQuery()
```

1.83 dlg.library/KillMsg

NAME

KillMsg -- Delete a message from an area

SYNOPSIS

```
result = KillMsg(message,area,username,pswd)
           D0      D1   A0      A1
BOOL KillMsg(LONG,USHORT,char *,char *)
```

FUNCTION

Deletes a message from an area.

INPUTS

```
message -- Number of the message to be deleted.

area    -- Number of the area to delete the message from (PVTAREA
           for a user's private area).

username -- Name of user if deleting a message from a user's private
           area.

pswd    -- Password to lock the area with while the message is
           being deleted.
```

RESULT

```
TRUE  if operation was successful
FALSE if function failed
```

EXAMPLE

```
KillMsg(23,PVTAREA,"John Doe","Killing message");
```

NOTES

BUGS

SEE ALSO

```
SendPublicMsg()
```

```
,  
SendRawMsg()  
,  
SendPrivateMsg()  
,  
ImportPublicMsg()
```

1.84 dlg.library/LeaveArea

NAME

LeaveArea -- Leave an area

SYNOPSIS

```
result = LeaveArea(area, flags)  
                D0   D1  
LONG LeaveArea(USHORT, UBYTE)
```

FUNCTION

Leaves an area that was previously entered.

INPUTS

area -- Area to leave.

flags -- As defined in resman.h.

RESULT

Error message as defined in resman.h.

EXAMPLE

```
error = LeaveArea(20, MSGLOCK);
```

NOTES

BUGS

SEE ALSO

```
EnterArea()  
,  
FreeArea()  
,  
BorrowArea()  
,  
LockArea()
```

1.85 dlg.library/ListAreas

NAME

ListAreas -- Display a list of available areas (file or message)

SYNOPSIS

```
result = ListAreas(name, user, type, sig)
```

```

                A0    A1    D0    D1
LONG ListAreas(char *, struct USER_DATA *, char, UBYTE)

```

FUNCTION

Display a list of available areas (file or message).

INPUTS

name -- The non-underscored name of the user (optional). If the name is provided and the user does not have access to the area based on level or the area is not auto-access, then the User.<file|msg> file is checked to see if access has been granted. If the name is not provided then only auto-access areas that the user's level qualifies them for is displayed.

user -- Address of the USER_DATA structure

type -- Type of area list to display 0=Message 1=File

sig -- SIG number to display areas for (0=no sig).

RESULT

0 = successful
-1 = unsuccessful

EXAMPLE

```
error = ListAreas(Ram.Name, &User, 0, 0);
```

NOTES

Honors the user's more prompt.

The "Please Wait.." has been removed and the areas are displayed as processed. It wasn't needed and was wasting memory and the user's patience.

If the user's screen width is too small for a two column display, the areas are now displayed in a single column.

BUGS

SEE ALSO

ListSIGS()

1.86 dlg.library/ListPorts

NAME

ListPorts -- Get a list of active ports

SYNOPSIS

```

result = ListPorts(buf,passwd)
                A0  A1
LONG ListPorts(char *,char *)

```

FUNCTION

Gets a list of active ports.

INPUTS

```
buf      -- Buffer to hold port list, three characters per port.

passwd  -- If non-NULL, only ports locked with this password will
         be listed.
```

RESULT

```
Error message as defined in resman.h.
```

EXAMPLE

```
error = ListPorts(buf, "BBS");
```

NOTES

BUGS

SEE ALSO

1.87 dlg.library/ListSIGS

NAME

```
ListSIGS -- Display a list of available SIGs (file or message)
```

SYNOPSIS

```
result = ListSIGS(user, type, filter)
                A0   D0   D1
LONG ListAreas(struct USER_DATA *, char, char)
```

FUNCTION

```
Display a list of available SIGs (file or message).
```

INPUTS

```
user      -- Address of the USER_DATA structure

type      -- Type of SIG list to display  0=Message 1=File

filter    -- Show all SIGs or only those that the user's access level
         qualify them for.  0=All SIGS  1=Access
```

RESULT

```
0 = successful
-1 = unsuccessful
```

EXAMPLE

```
error = ListSIGS(&User, 0, 0);
```

NOTES

```
Handles the user's more prompt.
```

```
If the user's screen width is too small for a two column display, the
SIGs are now displayed in a single column.
```

BUGS

SEE ALSO

ListAreas()

1.88 dlg.library/LoadLang

NAME

LoadLang -- Load a language

SYNOPSIS

```
result = LoadLang(port,name)
                A0  A1
LONG LoadLang(char *,char *)
```

FUNCTION

Loads a language as the new language for a port.

INPUTS

port -- Port to load language on.

name -- Name of the language to be loaded.

RESULT

Error message as defined in resman.h.

EXAMPLE

```
LoadLand("TR0", "Italian");
```

NOTES

BUGS

SEE ALSO

GetLang()

1.89 dlg.library/LockArea

NAME

LockArea -- Lock an area for an extended period of time

SYNOPSIS

```
result = LockArea(area,passwd,reason,pri,flags)
                D0  A0  A1  D1  D2
LONG LockArea(USHORT,char *,char *,char,UBYTE)
```

FUNCTION

Locks an area for an extended period of time. An application cannot lock an area if there are any users currently in it. The lock will pend until all users have left the area.

INPUTS

area -- Number of the area to be locked.

passwd -- Password to lock area with.

reason -- Reason the area is being locked.

pri -- Priority for lock (-127 to 128).

flags -- As defined in resman.h.

RESULT

Error as defined in resman.h.

EXAMPLE

```
result = LockArea(20, "MyPasswd", "Doing something", 0, MSGLOCK);
```

NOTES

BUGS

SEE ALSO

```
BorrowArea()
,
LeaveArea()
,
FreeArea()
,
EnterArea()
```

1.90 dlg.library/LockMenu

NAME

LockMenu -- Lock a menu

SYNOPSIS

```
result = LockMenu(port, ms, custnum, passwd, reason, pri, flags)
                A0  A1 D0      A2      A3      D1  D2
LONG LockMenu(char *, struct MenuStuff *, USHORT, char *,
              char *, char, UBYTE)
```

FUNCTION

Locks a menu and gets information about it.

INPUTS

port -- Port the application is running on.

ms -- MenuStuff structure.

custnum -- User's custom menu set number.

passwd -- Password to lock menu with.

reason -- Reason for locking menu.

```

pri      -- Priority of lock (-127 to 128).

flags    -- As defined in resman.h.

RESULT
  Error message as defined in resman.h.

EXAMPLE
  result = LockMenu(port, &ms, User->menuset, "Mypasswd", "Displaying", 0,
                    PENDLOCK).

NOTES
  This routine is not yet~suitable to be used by 3rd-party developers.

BUGS

SEE ALSO

```

```

    FreeMenu()
    ,
    PurgeMenu()

```

1.91 dlg.library/LockPort

```

NAME
  LockPort -- Lock a port

SYNOPSIS
  result = LockPort(port, passwd, reason, pri, bc)
                A0  A1      A2      D0  A3
  LONG LockPort(char *,char *,char *,char, char *)

FUNCTION
  Locks a port. The function will not return until a lock is obtained.
  If the port is in use by another application, the function will pend
  until the port is free.

INPUTS
  port      -- Three-character port name.

  passwd    -- Password to lock the port with.

  reason    -- Descriptive reason for lock.

  pri       -- Priority of lock (-127 to 128). Negative priority locks can
             be overridden by higher priority locks.

  bc        -- Break command - a command to be executed that will break the
             application if a higher priority lock is requested.

RESULT
  The result is an error message (see resman.h for #defines).

EXAMPLE
  result = LockPort("TR0", "MyPasswd", "No Reason", 0, NULL);

```

NOTES

BUGS

SEE ALSO

```
FreePort ()
'
TransferPortLock ()
'
ImmedLockPort ()
```

1.92 dlg.library/LockResource

NAME

LockResource -- Get a lock on a miscellaneous resource

SYNOPSIS

```
result = LockResource(name,passwd,reason,pri,flags)
                A0  A1      A2      D0  D1
LONG LockResource(char *,char *,char *,char,UBYTE)
```

FUNCTION

Gets a lock on a miscellaneous named resource.

INPUTS

```
name    -- Name of resource to be locked.  Can be any string.

passwd  -- Password to lock resource with.

reason  -- Reason for locking resource.

pri     -- Priority of lock (-127 to 128).

flags   -- Flags as defined in resman.h.
```

RESULT

Error message as defined in resman.h.

EXAMPLE

```
result = LockResource("MyResource","MyPasswd","No reason",0,PENDLOCK);
```

NOTES

BUGS

SEE ALSO

```
FreeResource ()
```

1.93 dlg.library/LogOut

NAME

Logout -- Performs DLG's basic logout processing

SYNOPSIS

Logout(Ram, User, Port, program)

A0 A1 A2 A3

VOID Logout(struct Ram_File *, struct USER_DATA *, char *, char *)

FUNCTION

Does DLG's basic logout processing (basically everything that GoodBye performs). First checks for carrier and either writes a NORMAL_LOGOUT or LOST_CARRIER event record. Handles any outstanding broadcast messages for that port. Cleans up DLG's temporary files and deletes the Removeuser Cron event. Updates the User data record for time online and last login date. Executes the Logout.DLGBatch file and displays the Logout.txt file.

INPUTS

Ram -- Address of the Ram_File structure of the user.

User -- Address of the USER_DATA structure.

port -- DLG Port (should be 4 characters) that the program is running on.

program -- Name of the program that was executing.

RESULT

TRUE if successful

FALSE if batch file not found

EXAMPLE

```
Logout(Ram, User, Port, "Mess");
```

NOTES

BUGS

SEE ALSO

1.94 dlg.library/MDate

NAME

MDate -- Make a timestamp

SYNOPSIS

MDate(string)

A0

void MDate(char *)

FUNCTION

Makes a string containing a timestamp of the current time.

INPUTS

string -- Pointer to a buffer to place the timestamp in. An example timestamp would be "Mon 3 May 93 1:22". The buffer must be 20 characters long (19 characters plus null-termination).

RESULT
none

EXAMPLE
MDate(mytimestamp);

NOTES

BUGS

SEE ALSO

```
SMDate()
,
UnpackTime()
,
AmigaTime()
```

1.95 dlg.library/More

NAME
More -- Print a "More [Y/n/=]" prompt

SYNOPSIS
result = More(fh,ansi,header)
A0 D0 D1
LONG More(BPTR,UBYTE,UBYTE)

FUNCTION
Prints a "More [Y/n/=]" prompt and waits for the user to respond.

INPUTS
fh -- AmigaDOS FileHandle to print prompt to.

ansi -- User's ANSI settings.

header -- Number of 'header' lines at the top of the screen. Should be 0 if More() is being called directly.

RESULT
0 if user responded 'yes'
1 if user responded 'no'
2 if user responded '='

EXAMPLE
result = More(Output(),User.Ansi_Flag,0);

NOTES

BUGS

SEE ALSO

Pause()

1.96 dlg.library/NextInGroup

NAME

NextInGroup -- Get the next name in a group

SYNOPSIS

```
result = NextInGroup(fh,name)
                A0 A1
BOOL NextInGroup(BPTR,char *)
```

FUNCTION

Gets the next name from a group file opened with OpenGroup().

INPUTS

```
fh    -- AmigaDOS FileHandle returned by OpenGroup().

name  -- Name to be filled in by NextInGroup().
```

RESULT

```
TRUE  if a name was read
FALSE if there are no more names in the group
```

EXAMPLE

```
while(NextInGroup(fh,name)) printf("Next name is [%s]\n",name);
```

NOTES

BUGS

SEE ALSO

```
OpenGroup()
'
CloseGroup()
```

1.97 dlg.library/OpenGroup

NAME

OpenGroup -- Open a group file to be accessed

SYNOPSIS

```
result = OpenGroup(groupname)
                A0
BPTR OpenGroup(char *)
```

FUNCTION

Opens a group file to be accessed.

INPUTS

groupname -- Name of group to be opened.

RESULT

AmigaDOS filehandle of group, or NULL if group couldn't be opened.

EXAMPLE

```
fh = OpenGroup("somegroup");
```

NOTES

Use

```
NextInGroup()
to access the names in the group.
```

BUGS

SEE ALSO

```
CloseGroup()
,
NextInGroup()
```

1.98 dlg.library/OverlayProgram

NAME

OverlayProgram -- Execute another program using the current CLI

SYNOPSIS

```
result = OverlayProgram(string)
                        A0
LONG OverlayProgram(char *)
```

FUNCTION

Execute another program using the current CLI. When the program ends, the current program will continue.

INPUTS

string -- Full path, filename and arguments of the program to execute.

RESULT

Exit code of the program executed.

EXAMPLE

```
result = OverlayProgram("DLG:LineEdit");
```

NOTES

The called program will have a new stack of the same size as the current program. If there is a resident version of the program it will be used and any path will be ignored.

If the file has the script flag set, it will be EXECUTED and the script result will be returned. The script will be able to output to the DLG port, but will not be able to read data from the port.

BUGS

SEE ALSO

ChainProgram()

1.99 dlg.library/Pause

NAME

Pause -- Print a "[Press Return]" prompt

SYNOPSIS

Pause()

BOOL Pause(void)

FUNCTION

Prints a "[Press Return]" prompt on the user's screen, and waits for the user to hit return.

INPUTS

none

RESULT

none

EXAMPLE

```
Pause();
```

NOTES

BUGS

SEE ALSO

More()

1.100 dlg.library/PrintSpace

NAME

PrintSpace -- Print spaces intelligently

SYNOPSIS

```
result = PrintSpace(fh, ansi, spaces)
```

```
      A0 D0  D1
```

```
BOOL PrintSpace(BPTR, UBYTE, USHORT)
```

FUNCTION

Prints spaces intelligently based on a user's ANSI settings.

INPUTS

```
fh      -- AmigaDOS FileHandle to send output to.
```

```
ansi -- User's ANSI settings. If the user has ANSI screen
      positioning enabled, it will be used to move the cursor the
      specified number of spaces (providing that there are enough
      spaces to make this more efficient).
```

```
spaces -- Number of spaces to print.
```

RESULT

```
TRUE
```

EXAMPLE

```
PrintSpace(Output(),User->Ansi_Flag,23);
```

NOTES**BUGS****SEE ALSO**

1.101 dlg.library/PurgeMenu

NAME

```
PurgeMenu -- Remove a menu from use.
```

SYNOPSIS

```
result = PurgeMenu(port,name)
           A0  A1
LONG PurgeMenu(char *,char *)
```

FUNCTION

```
Removes a menu from memory.
```

INPUTS

```
port -- Port application is running on.
```

```
name -- Name of menu set to be purged.
```

RESULT

```
Error message as defined in resman.h.
```

EXAMPLE

```
error = PurgeMenu("TR0","MAIN");
```

NOTES

```
This function is not yet ready for use by third party utilities.
```

BUGS**SEE ALSO**

```
LockMenu()
```

```
,
```

```
FreeMenu()
```

1.102 dlg.library/PutChar

```

NAME
PutChar -- Output a character

SYNOPSIS
PutChar(a, fh)
D0
void PutChar(char, BPTR)

FUNCTION
Sends a character to the specified file.

INPUTS
a -- Character to be output.

fh -- AmigaDOS FileHandle to output character to.

RESULT
none

EXAMPLE
PutChar('A', Output());

NOTES

BUGS

SEE ALSO
GetChar()
,
ReadChar()

```

1.103 dlg.library/PutHiLowFPointers

```

NAME
PutHiLowFPointers -- Write the high and low pointers for a file area

SYNOPSIS
result = PutHiLowFPointers(area, username, low, high, pswd)
D0 A0 D1 D2 A1
BOOL PutHiLowFPointers(USHORT, char *, LONG, LONG, char *)

FUNCTION
Writes the high and low file pointers for a file area.

INPUTS
area -- Number of the area (PVTAREA for a user's private area)

username -- Underscored name of the user, if writing pointers for a
private area. Otherwise this field should be NULL.

low -- New low pointer.

```

high -- New high pointer.

pswd -- Password to lock area with when writing pointers file.

RESULT

TRUE if operation succeeded
FALSE if failure

EXAMPLE

```
if (!PutHiLowFPointers(20, NULL, low, high, "pswd"))
    printf("Couldn't write pointers\n");
```

NOTES

This function puts a lock on the file area, so do not use it on an area that is already locked by the same application, or the program will hang up waiting for itself to release the area so it can lock it.

BUGS

SEE ALSO

```
GetHiLowFPointers()
,
PutHiLowPointers()
,
GetHiLowPointers()
```

1.104 dlg.library/PutHiLowPointers

NAME

PutHiLowPointers -- Write the high and low pointers for a message area

SYNOPSIS

```
result = PutHiLowPointers(area, username, low, high, pswd)
          D0  A0          D1  D2  A1
BOOL PutHiLowPointers(USHORT, char *, LONG, LONG, char *)
```

FUNCTION

Writes the high and low message pointers for a message area.

INPUTS

area -- Number of the area (PVTAREA for a user's private area)

username -- Underscored name of the user, if writing pointers for a private area. Otherwise this field should be NULL.

low -- New low pointer.

high -- New high pointer.

pswd -- Password to lock area with when writing pointers file.

RESULT

TRUE if operation succeeded

FALSE if failure

EXAMPLE

```
if (!PutHiLowPointers(20, NULL, low, high, "pswd"))
    printf("Couldn't write pointers\n");
```

NOTES

This function puts a lock on the message area, so do not use it on an area that is already locked by the same application, or the program will hang up waiting for itself to release the area so it can lock it.

BUGS

SEE ALSO

```
GetHiLowPointers()
,
GetHiLowFPointers()
,
PutHiLowFPointers()
```

1.105 dlgl.library/ReadArea

NAME

ReadArea -- Get information about a message/file area.

SYNOPSIS

```
result = ReadArea(area, msgarea, flag)
           D0   A0   D1
BOOL ReadArea(USHORT, struct Msg_Area *, UBYTE)
```

FUNCTION

Gets information about a message/file area.

INPUTS

area -- Number of the message/file area.

msgarea -- Msg_Area structure to be filled in (defined in msg.h).

flag -- 1 for a file area, 0 for a message area.

RESULT

TRUE if the operation was successful
FALSE if function failed

EXAMPLE

```
if (!ReadArea(3, &area, 1)) printf("Couldn't get info for file area 3\n");
```

NOTES

BUGS

SEE ALSO

1.106 dlg.library/ReadChar

NAME
ReadChar -- Wait for a character

SYNOPSIS
result = ReadChar(micros)
DO
char ReadChar(ULONG)

FUNCTION
Waits for a character for a specified length of time.

INPUTS
micros -- Number of microseconds to wait for.

RESULT
Character that was read, or 0 if function timed out.

EXAMPLE
c = ReadChar(1000);

NOTES

BUGS

SEE ALSO
GetChar()
,
PutChar()

1.107 dlg.library/ReadRam

NAME
ReadRam -- Read user's Ram_File structure

SYNOPSIS
result = ReadRam(RamStruct,port)
A0 A1
BOOL ReadRam(struct Ram_File *,char *)

FUNCTION
Reads the user's Ram_File structure.

INPUTS
RamStruct -- Pointer to Ram_File structure (defined in user.h) to be filled in.
port -- Port the user us on.

RESULT
TRUE if operation was successful
FALSE if fuction failed

EXAMPLE

```
if(!ReadRam(&Ram,"TR0")) printf("Unable to read Ram File\n");
```

NOTES

BUGS

SEE ALSO

```
WriteRam()
,
WriteUser()
,
ReadUser()
```

1.108 dlg.library/ReadUser

NAME

ReadUser -- Read a user's USER_DATA and Ram_File structures

SYNOPSIS

```
result = ReadUser(RamStruct,UserStruct,port)
                A0      A1      A2
BOOL ReadUser(struct Ram_File *,struct USER_DATA *,char *)
```

FUNCTION

Reads a user's USER_DATA and Ram_File structures.

INPUTS

```
RamStruct  -- Pointer to Ram_File structure (defined in user.h) to be
            filled in.

UserStruct -- Pointer to USER_DATA structure (defined in user.h) to be
            filled in.

port       -- Port the user is on.
```

RESULT

```
TRUE  if operation was successful
FALSE if function failed
```

EXAMPLE

```
if(!ReadUser(&Ram,&User,"TR0")) printf("Unable to read user data\n");
```

NOTES

BUGS

SEE ALSO

```
WriteUser()
,
WriteRam()
,
```

ReadRam()

1.109 dlgl.library/ReceiveFile

NAME

ReceiveFile -- Receives one or more file(s).

SYNOPSIS

```
result = ReceiveFile(path,protocol,header,UserStruct,RamStruct,port)
                A0  A1      A2      A3      D0      D1
```

```
BOOL ReceiveFile(char *, struct Protocol *, struct File_Header *,
                struct USER_DATA *, struct Ram_File *, char *)
```

FUNCTION

Receives one or more file(s).

INPUTS

path -- Upload path

protocol -- Pointer to a Protocol structure to be used (defined in file.h)

header -- Pointer to a file header structure with the filename filled in, if the protocol doesn't provide one (defined in file.h).

UserStruct -- Pointer to USER_DATA structure (defined in user.h).

RamStruct -- Pointer to Ram_File structure (defined in user.h).

port -- Port the user is on.

RESULT

TRUE if operation was successful

FALSE if function failed

EXAMPLE

```
result = ReceiveFile(path, protocol, header, &User, &Ram, port);
```

NOTES

Does a very basic file receive. Changes the current CLI directory to the upload path, translates the protocol's receive command, and then executes the receive command.

The application still has to insure that the upload path exists, prompt the user for the file description(s) and place the files in the proper area.

BUGS

SEE ALSO

SendFile()

1.110 dlg.library/ResourceMsg

NAME

ResourceMsg -- Low-level resource manager interface

SYNOPSIS

```
result = ResourceMsg(rmess)
                A0
LONG ResourceMsg(struct RMessage *)
```

FUNCTION

Provides a low-level interface to the resource manager.

INPUTS

rmess -- RMessage structure (defined in resman.h).

RESULT

Error message as defined in resman.h.

EXAMPLE

```
error = ResourceMsg(mymsg);
```

NOTES

Should not be called directly.

BUGS

SEE ALSO

1.111 dlg.library/ResumeTime

NAME

ResumeTime -- Resume the online clock for a port

SYNOPSIS

```
ResumeTime(Ram, port)
                A0  A1
VOID ResumeTime(struct Ram_File *,char *)
```

FUNCTION

Resume the online clock for a port with the number of minutes the port had left when suspended or up to the port shutdown time.

INPUTS

Ram -- Ram_File structure (defined in user.h).

port -- Port the action is occurring on.

RESULT

none

EXAMPLE

```
ResumeTime(&Ram, "TR0");
```

NOTES

Must be preceded by a call to
 SuspendTime()
 . DO NOT ResumeTime()
without having done a
 SuspendTime()
 .

BUGS

SEE ALSO

SuspendTime()

1.112 dlg.library/ScreenBuffer

NAME

ScreenBuffer -- Filter a buffer for objectionable language

SYNOPSIS

```
result = ScreenBuffer(inbuf,outbuf,maxsize,screenfile)
                        A0   A1   D0   A2
BOOL ScreenBuffer(char *,char *,ULONG,char *)
```

FUNCTION

Filters a buffer for bad language.

INPUTS

inbuf -- Buffer to be screened.

outbuf -- Buffer to put screened output into (note that this buffer
 may have to be bigger than inbuf).

maxsize -- Maximum size of the translated buffer (to avoid putting
 too much in outbuf).

screenfile -- Filename of screen.dat file to be used.

RESULT

TRUE if operation was successful
FALSE if function failed.

EXAMPLE

```
ScreenBuffer(inbuf,outbuf,1024,"screen.dat");
```

NOTES

BUGS

SEE ALSO

ScreenMsg()

1.113 dlg.library/ScreenMsg

NAME

ScreenMsg -- Filter a message for bad language

SYNOPSIS

```
result = ScreenMsg(filename,headerfile,msgtype,area)
                A0          A1          D0          D1
BOOL ScreenMsg(char *,char *,UBYTE,USHORT)
```

FUNCTION

Screens a message for bad language.

INPUTS

```
filename  -- Filename of the body text of the message (no message
           header should yet be in the file).

headerfile -- Filename of the fidonet-style header of the message.

msgtype   -- Type of the message (as defined in msg.h).

area      -- Number of the area the message is going to be placed in
           (for getting the correct "screen.dat" file).
```

RESULT

TRUE if operation was successful
FALSE if function failed.

EXAMPLE

```
ScreenMsg("T:TL0.msg","T:TL0.header",PUB_MSG,20);
```

NOTES

BUGS

SEE ALSO

ScreenBuffer()

1.114 dlg.library/ScreenPath

NAME

ScreenPath -- Screen a filename for invalid characters

SYNOPSIS

```
ScreenPath(filename)
                A0
void ScreenPath(char *)
```

FUNCTION

Screens a filename for invalid characters. The characters ':', '/', '#', '?', '*', '<', '>' and space are replaced with '_' (underscore).

```
INPUTS
  filename -- Filename to be screened.

RESULT
  none

EXAMPLE
  ScreenPath(filename);

NOTES

BUGS

SEE ALSO
```

1.115 dlg.library/SDraw_Line

```
NAME
  SDraw_Line -- Draw a line of dashes ('-') into a buffer.

SYNOPSIS
  SDraw_Line(buffer, size)
             A0      D0
  void SDraw_Line(char *, UBYTE)

FUNCTION
  Draws a line followed by a newline and a null to a string buffer.

INPUTS
  buffer -- buffer to draw line into

  size   -- size of the line including the newline

RESULT
  none

EXAMPLE
  SDraw_Line(buffer, 20);

NOTES

BUGS

SEE ALSO
```

```
Draw_Line()
```

1.116 dlg.library/SearchEnd

```
NAME
  SearchEnd -- End a file search
```

SYNOPSIS

```
SearchEnd(sc)
    A0
void SearchEnd(struct SearchCookie *)
```

FUNCTION

```
Ends a file search begun with
    SearchStart()
    INPUTS
sc -- SearchCookie returned by
    SearchStart()
    RESULT
none
```

EXAMPLE

```
SearchEnd(sc);
```

NOTES

BUGS

SEE ALSO

```
SearchStart()
,
SearchNext()
```

1.117 dlg.library/SearchNext

NAME

```
SearchNext -- Find the next file
```

SYNOPSIS

```
result = SearchNext(sc)
    A0
char *SearchNext(struct SearchCookie *)
```

FUNCTION

```
Finds the next file in a disk search.
```

INPUTS

```
sc -- SearchCookie returned by
    SearchStart()
.
```

RESULT

```
Pointer to a filename, or NULL if no more files are found.
```

EXAMPLE

```
sc = SearchStart("USER:Joe_Smith", "*");
while(filename = SearchNext(sc)) printf("[%s]\n", filename);
```

NOTES

BUGS

SEE ALSO

```
SearchStart()  
,  
SearchEnd()
```

1.118 dlg.library/SearchStart

NAME

SearchStart -- Begin a disk search, with pattern matching

SYNOPSIS

```
result = SearchStart(dir,pat)  
                A0 A1  
struct SearchCookie *SearchStart(char *,char *)
```

FUNCTION

Begins a search for files on disk.

INPUTS

dir -- Directory to search in.

pat -- Pattern to match ('*' and '?' wildcards supported).

RESULT

SearchCookie structure or
NULL if operation failed

EXAMPLE

```
sc = SearchStart("T:", "*.user");
```

NOTES

SearchStart() only gets things going,
SearchNext()
actually retrieves
the useful information. See
SearchNext()
for a useful example.

BUGS

SEE ALSO

```
SearchNext()  
,  
SearchEnd()
```

1.119 dlg.library/SendBulletin

NAME

SendBulletin -- Place a bulletin on the system

SYNOPSIS

```

result = SendBulletin(header,body,pswd)
                A0      A1      A2
LONG SendBulletin(struct Bulletin *,char *,char *)

```

FUNCTION

Places a bulletin on the system.

INPUTS

```

header -- Bulletin structure (defined in bulletin.h).

body   -- Null-terminated block of text that makes up the body of the
         message. This text should be in standard, fidonet format
         as specified by FTS-0001.

pswd   -- Password to lock the bulletin area with while the bulletin
         is being written.

```

RESULT

The number the bulletin was assigned, or FALSE if the operation failed

EXAMPLE

```

num = SendBulletin(&header,bodytext,"Sending Bulletin");

```

NOTES

BUGS

SEE ALSO

1.120 dlg.library/SendFile

NAME

SendFile -- Send one or more file(s).

SYNOPSIS

```

result = SendFile(protocol, path, batch, UserStruct, RamStruct, port)
                A0      A1      A2      A3          D0          D1
BOOL ReceiveFile(struct Protocol *, char *, char *,
                struct USER_DATA *, struct Ram_File *, char *)

```

FUNCTION

Receives one or more file(s).

INPUTS

```

protocol -- Pointer to a protocol structure to be used

path     -- Full path/filename of file to send

batch    -- If doing a batch send, this is the Full path/filename
           of the batch file. For a single file send, pass as NULL.

UserStruct -- Pointer to USER_DATA structure (defined in user.h).

```

```
RamStruct  -- Pointer to Ram_File structure (defined in user.h).

port       -- Port the user is on.
```

RESULT

```
TRUE  if operation was successful
FALSE if function failed
```

EXAMPLE

```
result = SendFile(protocol, path, NULL, &User, &Ram, port);
```

NOTES

Does a very basic file send. Translates the protocol's single send command or batch send command, and then executes the send command.

BUGS

SEE ALSO

ReceiveFile()

1.121 dlgl.library/SendCtlMsg

NAME

SendCtlMsg -- Low-level handler interface

SYNOPSIS

```
result = SendCtlMsg(mod, aux_stat, port)
                D0  D1      A0
LONG SendCtlMsg(LONG, LONG, char *)
```

FUNCTION

Provides a low-level interface to the handler. Should not be called directly.

INPUTS

```
mod       -- Handler command (defined in devices/tpt.h).

aux_stat  -- Command argument

port      -- Three-character port name
```

RESULT

Error message as defined in devices/tpt.h.

EXAMPLE

```
error = SendCtlMsg(T_ECHO, NULL, "TL0");
```

NOTES

BUGS

SEE ALSO

1.122 dlg.library/SendPrivateMsg

NAME

SendPrivateMsg -- Send a private message

SYNOPSIS

```
result = SendPrivateMsg(header,body,msgtype,pswd,port)
                        A0    A1  D0    A2  A3
```

```
LONG SendPrivateMsg(struct Msg_Header *,char *,USHORT,char *,char *)
```

FUNCTION

Sends a private message to a user on the system.

INPUTS

header -- Msg_Header structure (see msg.h for details).

body -- Null-terminated block of text that makes up the body of the message. This text should be in standard, fidonet format as specified by FTS-0001.

msgtype -- Type of message (as defined in msg.h).

pswd -- Password to lock the area with while the message is being written.

port -- Port the application is running on (or NULL, if not applicable).

RESULT

The number the message was assigned in the area, or FALSE if the operation failed

EXAMPLE

```
num = SendPrivateMsg(&header,bodytext,PRI_MSG,"Sending Private",NULL);
```

NOTES

BUGS

At the moment, this function will crash if you don't allocate enough memory for DLG to append the origin and tearline to the end of your message, if sent in a Fido or UUCP message base. To be safe, allocate twice the size of the body file. This will eventually get fixed in a future version of the library.

SEE ALSO

```
SendPublicMsg()
,
KillMsg()
,
ImportPublicMsg()
,
SendRawMsg()
```

1.123 dlg.library/SendPublicMsg

NAME

SendPublicMsg -- Send a public message

SYNOPSIS

```
result = SendPublicMsg(ms, fido, pswd, port)
                        A0 A1  A2  A3
```

```
LONG SendPublicMsg(struct MsgStruct *, struct fido *, char *, char *)
```

FUNCTION

Sends a public message in a DLG message area. If the area is a fidonet area, tear and origin lines will be placed on the message.

INPUTS

```
ms    -- MsgStruct structure. This structure (defined in msg.h) is
      as follows:

      struct Msg_Header *header    -- Fidonet message header structure
                                   (see msg.h for details).

      struct Msg_Header *repheader -- Header of the message this is a reply
                                   to (or NULL if message is not a
                                   reply).

      unsigned char *body          -- Null-terminated block of text that
                                   makes up the body of the message.
                                   This text should be in standard,
                                   fidonet format as specified by
                                   FTS-0001.

      USHORT replyto              -- Number of the message this message is
                                   a reply to (or 0 if not a reply).

      struct Msg_Area *areainfo    -- Msg_Area structure of the area to
                                   place the message in. This
                                   structure is obtained by using

                                   ReadArea()
                                   .

      long flags                   -- MSG_NOORIGIN if the body text already
                                   has a tear and origin line appended.

fido  -- fido structure (defined in misc.h) obtained from the file
      "dlgconfig:port/FidoNet.Settings". The fido structure is only
      required if the message is being placed in a fidonet area,
      otherwise NULL.

pswd  -- Password to lock the area with while the message is being
      written.

port  -- Port the application is running on (or NULL if not applicable).
```

RESULT

The number the message was assigned in the area, or FALSE if the

operation failed

EXAMPLE

```
num = SendPublicMsg(&ms,&fido,"Sending Public","TR0");
```

NOTES

BUGS

At the moment, this function will crash if you don't allocate enough memory for DLG to append the origin and tearline to the end of your message, if sent in a Fido or UUCP message base. To be safe, allocate twice the size of the body file. This will eventually get fixed in a future version of the library.

SEE ALSO

```
SendPrivateMsg()
,
KillMsg()
,
ImportPublicMsg()
,
SendRawMsg()
```

1.124 dlg.library/SendRawMsg

NAME

SendRawMsg -- Low-level message sending routine

SYNOPSIS

```
result = SendRawMsg(ms,toname,pswd)
           A0 A1      A2
LONG SendRawMsg(struct MsgStruct *,char *,char *)
```

FUNCTION

INPUTS

```
ms      -- MsgStruct (see
           SendPublicMsg()
           for details).

toname  -- Name of user if message is to be placed in a user's private
           directory.

pswd    -- Password to lock area with while the message is being
           written.
```

RESULT

The number the message was assigned in the area, or FALSE if the operation failed

EXAMPLE

```
num = SendRawMsg(&ms,NULL,"Sending message");
```

NOTES

BUGS

At the moment, this function will crash if you don't allocate enough memory for DLG to append the origin and tearline to the end of your message, if sent in a Fido or UUCP message base. To be safe, allocate twice the size of the body file. This will eventually get fixed in a future version of the library.

SEE ALSO

```
SendPublicMsg()  
,  
KillMsg()  
,  
SendPrivateMsg()  
,  
ImportPublicMsg()
```

1.125 dlg.library/SmartRename

NAME

SmartRename -- Renames a file in a smart manner.

SYNOPSIS

```
SmartRename(source, dest)  
           A0      A1  
void SmartRename(char *, char *)
```

FUNCTION

If the destination is the same drive as the source, the file is renamed. Otherwise, copies all of a file into another file and then deletes the source file. Should the destination drive become full during the copy and the DLGConfig:Batch/DriveIsFull.batch exists, it will be executed.

INPUTS

```
source -- Path/Name of source file.  
  
dest   -- Path/Name of destination file.
```

RESULT

```
0 = successful  
-1 = destination file exists  
-2 = error copying file
```

EXAMPLE

```
SmartRename("T:File1", "T:File2");
```

NOTES

BUGS

SEE ALSO

Copy ()

1.126 dlg.library/SMDate

NAME

SMDate -- Make a timestamp

SYNOPSIS

```
SMDate (cur_time, string)
        D0      A0
void SMDate (ULONG, char *)
```

FUNCTION

Makes a timestamp string for the time specified.

INPUTS

cur_time -- Time, as returned by
AmigaTime ()

.

string -- Pointer to a buffer to place the timestamp in. An
example timestamp would be "Mon 3 May 93 1:22". The
buffer must be 20 characters long (19 characters plus
null-termination).

RESULT

none

EXAMPLE

```
SMDate (
        AmigaTime ()
        , mytimestamp);
```

NOTES

BUGS

SEE ALSO

```
MDate ()
,
UnpackTime ()
,
AmigaTime ()
```

1.127 dlg.library/Stricmp

NAME

Stricmp -- Case insensitive string compare

SYNOPSIS

```
result = Stricmp(str1,str2)
          A0  A1
LONG Stricmp(char *,char *)
```

FUNCTION

Does a case insensitive string compare

INPUTS

str1 -- First string.

str2 -- Second string.

RESULT

<0 if str1 is alphanumerically smaller than str2

0 if str1 is identical to str2

>0 if str1 is alphanumerically greater than str2

EXAMPLE

```
if (Stricmp("THIS","this")) printf("Hmmm, they should be equal\n");
```

NOTES

Using this function instead of the SAS/C stricmp() will save a lot of excessive code, 1-4K depending on other functions used.

BUGS**SEE ALSO**

Strnicmp()

1.128 dlg.library/StripPath

NAME

StripPath -- Get a root filename

SYNOPSIS

```
result = StripPath(path)
```

A0

```
char *StripPath(char *)
```

FUNCTION

Gets a root filename from a path/filename.

INPUTS

path -- Path/filename to find root of.

RESULT

Pointer to the filename part of the given path.

EXAMPLE

```
filename = StripPath("work:games/blazemonger");
```

NOTES**BUGS**

SEE ALSO

1.129 dlg.library/StripSpaces

NAME

StripSpaces -- Removes leading and trailing spaces from a string

SYNOPSIS

```
StripSpaces(string)
           A0
void StripSpaces(char *)
```

FUNCTION

Removes leading and trailing spaces from a string

INPUTS

string -- string to be stripped

RESULT

EXAMPLE

```
StripSpaces(name);
```

NOTES

BUGS

SEE ALSO

1.130 dlg.library/Strnicmp

NAME

Strnicmp -- Case insensitive, fixed-length string compare

SYNOPSIS

```
result = Strnicmp(str1, str2, n)
           A0  A1  D0
LONG Strnicmp(char *, char *, USHORT)
```

FUNCTION

Does a case insensitive, fixed-length string compare

INPUTS

str1 -- First string.

str2 -- Second string.

n -- Number of characters to compare

RESULT

<0 if str1 is alphanumerically smaller than str2

```

    0 if str1 is identical to str2
    >0 if str1 is alphanumerically greater than str2

```

EXAMPLE

```
if(Stricmp("THIS?", "this!", 4)) printf("Hmmm, they should be equal\n");
```

NOTES

Using this instead of the SAS/C `strnicmp()` function will generally shave 1-4K off your program, depending on other functions utilized.

BUGS

SEE ALSO

`Stricmp()`

1.131 dlgl.library/Substitute

NAME

Substitute -- Substitute the translated value for a single '%' switch

SYNOPSIS

```

result = Substitute(cstr, result, User, Ram, port)
           A0   A1     A2   A3  D0
BOOL Substitute(char *, char *, struct USER_DATA *,
               struct Ram_File *, char *)

```

FUNCTION

Substitutes the translated value for a single '%' switch

INPUTS

```

cstr    -- String to be translated.

result  -- String to put translation into.

User    -- USER_DATA structure (defined in user.h).

Ram     -- Ram_File structure (defined in user.h).

port    -- Port the translation is occurring on.

```

RESULT

```

TRUE if operation was successful
FALSE if function failed

```

EXAMPLE

```
Substitute("UNAME", result, &User, &Ram, "TR0");
```

NOTES

Note the example above. Do NOT include the "%" with the switch text, just its name.

BUGS

SEE ALSO

TranslateBuffer()

1.132 dlg.library/SuspendTime

NAME

SuspendTime -- Suspend the online clock for a port

SYNOPSIS

```
SuspendTime(Ram, port)
             A0  A1
VOID SuspendTime(struct Ram_File *,char *)
```

FUNCTION

Suspends the online clock for a port up to the port shutdown time.

INPUTS

```
Ram    -- Ram_File structure (defined in user.h).

port   -- Port the translation is occurring on.
```

RESULT

none

EXAMPLE

```
SuspendTime(&Ram, "TR0");
```

NOTES

BUGS

SEE ALSO

ResumeTime()

1.133 dlg.library/TBaud

NAME

TBaud -- Set the baud rate for a port

SYNOPSIS

```
result = TBaud(baud,port)
             D0  A0
LONG TBaud(LONG,char *)
```

FUNCTION

Sets the baud rate for a port.

INPUTS

```
baud -- Baud rate.

port -- Port.
```

RESULT
0 if successful
negative if an error occurred

EXAMPLE
error = TBaud(19200,"TR0");

NOTES

BUGS

SEE ALSO

1.134 dlg.library/TCheckCarrier

NAME
TCheckCarrier -- Checks for the presence of a carrier

SYNOPSIS
result = TCheckCarrier(port)
 A0
LONG TCheckCarrier(char *)

FUNCTION
Checks for the presence of a carrier on a port.

INPUTS
port -- The port to check

RESULT
TRUE if carrier present
FALSE if no carrier present

EXAMPLE
Carrier = TCheckCarrier("TR0");

NOTES

BUGS

SEE ALSO

1.135 dlg.library/TColors

NAME
TColors -- Change the colors for a port

SYNOPSIS
result = TColors(colortable,port)
 A0 A1
LONG TColors(USHORT *,char *)

```
FUNCTION
    Changes the colors for a port.

INPUTS
    colortable -- Color table suitable for passing to the graphics.library
                LoadRGB4() routine.

    port      -- Port.

RESULT
    0 if successful
    negative if an error occurred

EXAMPLE
    error = TColors(colors, "TR0");

NOTES

BUGS

SEE ALSO
```

1.136 dlg.library/TCont

```
NAME
    TCont -- UnFreeze a port

SYNOPSIS
    result = TCont(port)
                A0
    LONG TCont(char *)

FUNCTION
    UnFreezes a port frozen with
        TFreeze()
    .

INPUTS
    port -- Port.

RESULT
    0 if successful
    negative if an error occurred

EXAMPLE
    TCont("TR0");

NOTES

BUGS

SEE ALSO
```

TFreeze()

1.137 dlg.library/TDevQuery

NAME

TDevQuery -- Get information about a port

SYNOPSIS

```
result = TDevQuery(devstruct,port)
                A0      A1
LONG TDevQuery(struct tdev_info *,char *)
```

FUNCTION

Gets information about a port.

INPUTS

devstruct -- tdev_info structure. The format of this structure is as follows:

```
char devname[21]  -- Name of serial device being used.
unsigned char unit -- Unit number of serial device being used.
long serflags     -- Serial flags being used.

port             -- Port.
```

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TDevQuery(&ds,"TR0");
```

NOTES

BUGS

SEE ALSO

1.138 dlg.library/TFreeze

NAME

TFreeze -- Cause port to suspend all I/O

SYNOPSIS

```
result = TFreeze(port)
                A0
LONG TFreeze(char *)
```

FUNCTION

Causes port to suspend all I/O.

INPUTS
port -- Port.

RESULT
0 if successful
negative if an error occurred

EXAMPLE
TFreeze("TR0");

NOTES

BUGS

SEE ALSO

TCont()

1.139 dlg.library/TGetSer

NAME
TGetSer -- Get serial informaiton for a port

SYNOPSIS
result = TGetSer(serstruct,port)
 A0 A1
LONG TGetSer(struct TPTSerStuff *,char *)

FUNCTION
Gets serial information for a port.

INPUTS
serstruct -- TPTSerStuff structure to be filled in. This structure has
the following format:

 struct IOExtSer *read -- IOMessage for reading.

 struct IOExtSer *write -- IOMessage for writing.

port -- Port.

RESULT
0 if successful
negative if an error occurred

EXAMPLE
error = TGetSer(&ss,"TR0");

NOTES

BUGS

SEE ALSO

1.140 dlg.library/TGetTitle

NAME

TGetTitle -- Get the screen/window title for a port

SYNOPSIS

```
result = TGetTitle(title,port)
                A0    A1
LONG TGetTitle(char *,char *)
```

FUNCTION

Gets the screen/window title for a port.

INPUTS

```
title -- String to place title in.

port  -- Port.
```

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TGetTitle(title,"TR0");
```

NOTES

BUGS

SEE ALSO

TTitle()

1.141 dlg.library/TimeUntilShutdown

NAME

TimeUntilShutdown -- Get the number of minutes until a port is shutdown

SYNOPSIS

```
result = TimeUntilShutdown(port)
                A0
LONG TimeUntilShutdown(char *)
```

FUNCTION

Returns the number of minutes until the port shuts down. If there is no shutdown event scheduled for the port then 1440 minutes (24 hours) is returned.

INPUTS

```
port -- DLG Port
```

RESULT

Minutes till port shutdown or 1440

EXAMPLE

```
minutes = TimeUntilShutdown("TR0");
```

NOTES

BUGS

SEE ALSO

```
SuspendTime()  
,  
ResumeTime()
```

1.142 dlg.library/TInTrans

NAME

TInTrans -- Set the input translation table for a port

SYNOPSIS

```
result = TInTrans(trans, port)  
          A0      A1  
LONG TInTrans(char *, char *)
```

FUNCTION

Sets the input translation table for a port.

INPUTS

trans -- Array of 256 characters. Input character x will be mapped to trans[x].

port -- Port.

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TInTrans(trans, "TR0");
```

NOTES

BUGS

SEE ALSO

```
TOutTrans()
```

1.143 dlg.library/TKill

NAME

TKill -- Kill a port

SYNOPSIS

```
result = TKill(port)
          A0
LONG TKill(char *)
```

FUNCTION

Kills a port. The user will be hung up on.

INPUTS

port -- Port.

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TKill("TR0");
```

NOTES

BUGS

SEE ALSO

TRecover()

1.144 dlg.library/TOutTrans

NAME

TOutTrans -- Set the output translation table for a port

SYNOPSIS

```
result = TOutTrans(trans,port)
          A0    A1
LONG TOutTrans(char *,char *)
```

FUNCTION

Sets the output translation table for a port.

INPUTS

trans -- Array of 256 characters. Character x will be output is trans[x].

port -- Port.

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TOutTrans(trans,"TR0");
```

NOTES

BUGS

SEE ALSO

TInTrans()

1.145 dlgl.library/TransferPortLock

NAME

TransferPortLock -- Change the lock on a port

SYNOPSIS

```
result = TransferPortLock(port,passwd,newpasswd,reason,pri,bc)
                        A0  A1      A2      A3      D0  D1
LONG TransferPortLock(char *,char *,char *,char *,char, char *)
```

FUNCTION

Changes the status of a lock on a port.

INPUTS

```
port      -- Port to be transferred.

passwd    -- Password port was previously locked with.

newpasswd -- New password to lock port with.

reason    -- Reason for new lock.

pri       -- Priority of new lock.

bc        -- Background command for new lock (see LockPort for
            more info).
```

RESULT

Error message as defined in resman.h.

EXAMPLE

```
error = TransferPortLock("TR0", "OldPasswd", "NewPasswd",
                        "New Reason", 0, "");
```

NOTES

BUGS

SEE ALSO

```
LockPort()
,
FreePort()
,
ImmedLockPort()
```

1.146 dlgl.library/TranslateBuffer

NAME

TranslateBuffer -- Translate '%' switches in a buffer

SYNOPSIS

```
result = TranslateBuffer(inbuffer,outbuffer,maxsize,User,Ram,port)
                        A0      A1      D0      A2  A3  D1
LONG TranslateBuffer(char *,char *,ULONG,struct USER_DATA *,
                    struct Ram_File *,char *)
```

FUNCTION

Translates the '%' switches in a buffer.

INPUTS

inbuffer -- Buffer to be translated.

outbuffer -- Buffer to put translation into (may have to be bigger than inbuffer).

maxsize -- Maximum size of translated buffer (to avoid putting too many characters in outbuffer).

User -- USER_DATA structure.

Ram -- Ram_File structure.

port -- Port the user is on.

RESULT

Number of characters placed in outbuffer.

EXAMPLE

```
numchars = Translate(inbuf,outbuf,1024,&User,&Ram,"TR0");
```

NOTES

BUGS

SEE ALSO

Substitute()

1.147 dlg.library/TRecover

NAME

TRecover -- Recover a killed port

SYNOPSIS

```
result = TRecover(port)
                        A0
LONG TRecover(char *)
```

FUNCTION

Recovers a killed port if for some reason it couldn't shut down properly


```
port      -- Port.
```

RESULT

```
0 if successful
negative if an error occurred
```

EXAMPLE

```
error = TScreen(1, &scr, colors);
```

NOTES**BUGS****SEE ALSO**

```
TWindow()
```

1.149 dlg.library/TSendBreak

NAME

```
TSendBreak -- Send immediate break to a port.
```

SYNOPSIS

```
result = TSendBreak(port)
                A0
LONG TSendBreak(char *)
```

FUNCTION

```
Immediately sends a break to the indicated port.
```

INPUTS

```
port -- Port to send break to.
```

RESULT

```
0 if successful
negative if an error occurred
```

EXAMPLE

```
result = TSendBreak("TR0");
```

NOTES

```
Any writes that are in progress to the port are aborted and the break
is sent immediately.
```

BUGS**SEE ALSO**

1.150 dlg.library/TSetFlags

NAME

```
TSetFlags -- Set handler flags
```

SYNOPSIS

```
result = TSetFlags(flags,port)
                D0    A0
LONG TSetFlags(ULONG,char *)
```

FUNCTION

Sets various handler flags.

INPUTS

flags -- As follows:

1	T_ECHO	-	1	-- Enable echoing of characters.
2	T_CRLF	-	2	-- Enable CR/LF conversion.
4	T_RAW	-	4	-- Enable RAW moded.
8	T_RPEND	-	8	-- Read pending. (Set *only* by TPT-Handler)
10	T_WAIT_FOR	-	16	-- Wait for input. (Set *only* by TPT-Handler)
20	T_TYPEAHEAD_FULL	-	32	-- TypeAhead full. (Set *only* by TPT-Handler)
40	T_BREAK	-	64	-- Pass through user-typed ^C signals.
80	T_WINDOW	-	128	-- Window is opened. (Set *only* by TPT-Handler)
100	T_KILL_ENABLE	-	256	-- Allow handler to send ^C kill signals.
200	T_DO_PEND	-	512	-- Keep track of pending kills.
400	T_KILL_PEND	-	1024	-- Control-C sent. (Set *only* by TPT-Handler)
800	T_SER_TIMEOUT	-	2048	-- Serial Timeout. (Set *only* by TPT-Handler)
1000	T_DO_TIMEOUT	-	4096	-- Enable inactivity timeouts.
2000	T_CTLD	-	8192	-- Pass ^D characters through.
4000	T_PAUSE	-	16384	-- Enable ^S^Q pausing.
8000	T_PAUSED	-	32768	-- Port Paused. (Set *only* by TPT-Handler)
10000	T_KILLED	-	65536	-- Port Killed. (Set *only* by TPT-Handler)
20000	T_SCREEN	-	131072	-- Screen is opened. (Set *only* by TPT-Handler)
40000	T_PASS_THRU	-	262144	-- Enable 'passthru' mode.
80000	T_VERB_PAUSE	-	524288	-- Display verbose "[PAUSED]" message.
100000	T_CWRITE_PEND	-	1048576	-- Console write pend. (Set *only* by TPT-Handler)
200000	T_LINEFREEZE	-	2097152	-- Freeze output when user starts typing in line mode.
400000	T_FROZEN	-	4194304	-- Port Frozen. (Set *only* by TPT-Handler)
800000	T_WRITE_PEND	-	8388608	-- Serial write pend. (Set *only* by TPT-Handler)

port -- Port.

RESULT

new handler flags.

EXAMPLE

```
flags = TSetFlags(T_ECHO|T_RAW,"TR0");
```

NOTES

In order to get the current flag settings do:

```
flags = TSetFlags(0, port);
```

the flags returned will be the current flags.

BUGS

SEE ALSO

```
TUnSetFlags()
```

1.151 dlg.library/TString

NAME

TString -- Pretend a user typed a string

SYNOPSIS

```
result = TString(string,port)
           A0      A1
LONG TString(char *,char *)
```

FUNCTION

Take a string as if it was typed as input by a user on a port.

INPUTS

string -- String.

port -- Port.

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
error = TString("Pretend user typed this","TR0");
```

NOTES

BUGS

SEE ALSO

1.152 dlg.library/TTimeDelay

NAME

TTimeDelay -- Set the timeout delay for a port

SYNOPSIS

```
result = TTimeDelay(delay,port)
```

```

                                D0    A0
LONG TTimeDelay(LONG, char *)

FUNCTION
    Sets the timeout delay for a port.

INPUTS
    delay -- Timeout delay (in 5-second intervals).

    port  -- Port.

RESULT
    0 if successful
    negative if an error occurred

EXAMPLE
    error = TTimeDelay(6, "TR0");

NOTES

BUGS

SEE ALSO
```

1.153 dlg.library/TTitle

```

                                NAME
TTitle -- Change the screen/window title for a port

SYNOPSIS
    result = TTitle(title, port)
                                A0    A1
LONG TTitle(char *, char *)

FUNCTION
    Changes the screen/window title for a port.

INPUTS
    title -- New title.

    port  -- Port.

RESULT
    0 if successful
    negative if an error occurred

EXAMPLE
    error = TTitle("New title", "TR0");

NOTES

BUGS

SEE ALSO
```

TSetTitle()

1.154 dlg.library/TUnSetFlags

NAME

TUnSetFlags -- Unset handler flags for a port

SYNOPSIS

```
result = TUnSetFlags(flags,port)
                D0      A0
LONG TUnSetFlags(ULONG,char *)
```

FUNCTION

Unsets various handler flags for a port.

INPUTS

```
flags -- See
        TSetFlags()
        .

port  -- Port.
```

RESULT

0 if successful
negative if an error occurred

EXAMPLE

```
TUnSetFlags(T_ECHO|T_RAW,"TR0");
```

NOTES

BUGS

SEE ALSO

TSetFlags()

1.155 dlg.library/TWindow

NAME

TWindow -- Open/close a window on a port

SYNOPSIS

```
result = TWindow(onoff,winstruct,port)
                D0      A0      A1
LONG TWindow(LONG,struct WinStruct *,char *)
```

FUNCTION

Opens/closes a window on a port.

INPUTS

```
onoff  -- 0 to close, 1 to open.
```

```
winstruct -- WinStruct structure. The format of this structure
          (defined in portconfig.h) is as follows:

short x, y          -- x and y position of upper left corner of
                    window.

short width, height -- Width and height of window.

char fontname[41]   -- Name of font to be used (case sensitive and
                    must include ".font").

UBYTE fontsize     -- Point size of font.

UBYTE flags        -- DISP_BKGRND if window should be opened up
                    behind all other windows.

port              -- Port.
```

```
RESULT
  0 if successful
  negative if an error occurred
```

```
EXAMPLE
  error = TWindow(1, &ws, "TR0");
```

NOTES

BUGS

SEE ALSO

TScreen()

1.156 dlg.library/TWinHeight

```
NAME
  TWinHeight -- Change the height of the window on a port
```

```
SYNOPSIS
  result = TWinHeight(height, port)
                    A0    A1
  LONG TWinHeight(char *, char *)
```

```
FUNCTION
  Changes the height of the window on a port.
```

```
INPUTS
  height -- New height of window.
```

```
port -- Port.
```

```
RESULT
  0 if successful
  negative if an error occurred
```

EXAMPLE

NOTES

BUGS

SEE ALSO

1.157 dlg.library/UnderScore

NAME

UnderScore -- Underscore a string

SYNOPSIS

```
UnderScore(string)
           A0
void UnderScore(char *)
```

FUNCTION

Replaces spaces with underscores '_' in a string. Useful when converting a username to a user's directory name.

INPUTS

string -- String to be underscored.

RESULT

none

EXAMPLE

```
UnderScore("John Doe");
```

NOTES

BUGS

SEE ALSO

DeScore ()

1.158 dlg.library/UnpackTime

NAME

UnpackTime -- Unpack a time value

SYNOPSIS

```
UnpackTime(secs, at)
           D0  A0
void UnpackTime(ULONG, struct ATime *)
```

FUNCTION

Unpacks a time value returned from

```
        AmigaTime()
        into a more accessible
structure.
```

INPUTS

```
secs -- Number of seconds returned by
        AmigaTime()
        .
```

```
at   -- Pointer to an ATime structure to be filled in.  See misc.h for
        the format of this structure.
```

RESULT

```
none
```

EXAMPLE

```
UnpackTime(
        AmigaTime()
        , timestruct);
```

NOTES

BUGS

SEE ALSO

```
        AmigaTime()
        ,
        SMDate()
        ,
        MDate()
```

1.159 dlg.library/Upper

NAME

```
Upper -- Convert a string to uppercase
```

SYNOPSIS

```
Upper(string)
        A0
void Upper(char *)
```

FUNCTION

```
Converts a string to uppercase.
```

INPUTS

```
string -- String to be converted
```

RESULT

```
none
```

EXAMPLE

```
Upper("uppercase this");
```

NOTES

BUGS

SEE ALSO

Capitalize()

1.160 dlg.library/WaitingMail

NAME

WaitingMail -- Add a message to a user's waiting mail list

SYNOPSIS

```
result=WaitingMail(toname,from,subject,areanum,areaname,messagenum,port)
                   A0      A1   A2      D0      A3      D1      D2
```

```
BOOL WaitingMail(char *,char *,char *,SHORT,char *,SHORT,char *)
```

FUNCTION

Adds a message to a user's waiting mail list and informs them about it.

INPUTS

toname -- User the mail is for.

from -- User the mail is from.

subject -- Subject of the message.

areanum -- Number of the area the message is in.

areaname -- Name of the area the message is in.

messagenum -- Number of the message.

port -- Port the application sending the waiting mail is on.

RESULT

TRUE if the operation was successful

FALSE if the function failed

EXAMPLE

```
WaitingMail("John Doe","Fred Doe","Hey Dude!",23,"General",457,NULL);
```

NOTES

BUGS

SEE ALSO

1.161 dlg.library/WhenEvent

```

NAME
  WhenEvent -- Check when an event will next happen

SYNOPSIS
  result = WhenEvent(string)
                A0
  LONG WhenEvent(char *)

FUNCTION
  Checks when a TPTCron event will next happen.

INPUTS
  string -- Pattern to search for ('*' and '?' wildcards supported).

RESULT
  Number of seconds until event
  -1 if operation failed

EXAMPLE
  secs = WhenEvent("*UU*");
  if(secs==-1) printf("WhenEvent failed\n");
  else printf("The next UUCP event will occur in %d seconds\n",secs);

NOTES

BUGS

SEE ALSO
  CronEvent()

```

1.162 dlg.library/WriteEvent

```

NAME
  WriteEvent -- Write a line to a user's event log

SYNOPSIS
  result = WriteEvent(name,buf)
                A0  A1
  BOOL WriteEvent(char *,char *)

FUNCTION
  Writes a line to a user's event log.

INPUTS
  name -- Name of user.

  buf  -- String to be written.

RESULT
  TRUE  if operation was successful
  FALSE if operation failed

EXAMPLE

```

```
if(!WriteEvent("John Doe","Something important happened"))
    printf("Failed to tell John something important\n");
```

NOTES

BUGS

SEE ALSO

Inform()

1.163 dlg.library/WriteLog

NAME

WriteLog -- Write a event to the system log

SYNOPSIS

```
result = WriteLog(code, person, port, info)
           D0   A0       A1   A2
BOOL WriteLog(UBYTE, char *, char *, char *)
```

FUNCTION

Writes an event to the system log.

INPUTS

code -- Event code number (defined in log.h, or user defined).

person -- User the event pertains to.

port -- Port event pertains to.

info -- Comment about the event.

RESULT

TRUE if operation was successful
FALSE if operation failed

EXAMPLE

```
WriteLog(PAGED_SYSOP, "John Doe", "TR0", "No Comment");
```

NOTES

BUGS

SEE ALSO

1.164 dlg.library/WriteRam

NAME

WriteRam -- Write a user's Ram_File structure

SYNOPSIS

```

result = WriteRam(RamStruct,port)
                A0      A1
BOOL WriteRam(struct Ram_File *,char *)

```

FUNCTION

INPUTS

```

RamStruct -- Pointer to Ram_File structure to be written.

port      -- Port the user is on.

```

RESULT

```

TRUE  if operation was successful
FALSE if function failed

```

EXAMPLE

```

WriteRam(&Ram, "TR0");

```

NOTES

BUGS

SEE ALSO

```

    ReadRam()
    ,
    WriteUser()
    ,
    ReadUser()

```

1.165 dlg.library/WriteUser

NAME

```

WriteUser -- Write a user's USER_DATA structure.

```

SYNOPSIS

```

result = WriteUser(name,UserStruct)
                A0      A1
BOOL WriteUser(char *,struct USER_DATA *)

```

FUNCTION

```

Writes a user's USER_DATA structure.

```

INPUTS

```

name      -- User's name.

UserStruct -- Pointer to USER_DATA structure to be written.

```

RESULT

```

TRUE  if operation was successful
FALSE if function failed

```

EXAMPLE

```

WriteUser("John Doe",&User);

```

NOTES

BUGS

SEE ALSO

```

    ReadUser()
    ,
    WriteRam()
    ,
    ReadRam()

```

1.166 dlg.library/XAFPrintf

NAME

XAFPrintf -- Send formatted output to a file

SYNOPSIS

```

result = XAFPrintf(User, fh, fmt, argptr)
           A0  A1 A2  A3
LONG XAFPrintf(struct USER_DATA *,BPTR, char *,void *)

```

FUNCTION

Does standard 'C'-style formatting to a file. Should not be called directly. See

```

    AFPrintf()
    .

```

INPUTS

```

User    -- Optional USER_DATA structure (used for ansi color).

fh      -- AmigaDOS file handle to send output to.

fmt     -- Format string containing text and switches (see any printf()
          documentation for examples of the switches).

argptr  -- Pointer to a memory area (usually the stack) that contains
          the arguments to the formatting statements. Note that all
          arguments must be long values.

```

RESULT

The result is the number of characters output.

EXAMPLE

```

See the file '
    format.c
' included with this archive for an interface
function to be used with this routine.

```

NOTES

Compatible with most printf() format strings. If the User structure is passed, the format string may include DLG %a and %b color codes. There is no floating point support nor is %x formatting supported.

BUGS

When using

```
format.c
's
AFPrintf()
```

to interface to this function, all arguments are converted to LONGs. A format of "%hd" should not be used and will cause invalid results, use "%d" instead. If you call this function directly, you should use "%hd" for SHORTs and will get the proper results.

SEE ALSO

```
XASPrintf()
,
AFormat()
,
AFPrintf()
,
ASPrintf()
```

1.167 dlg.library/XASPrintf

NAME

XASPrintf -- Put formatted output into a string

SYNOPSIS

```
result = XASPrintf(User,buf,fmt,argptr)
           A0  A1  A2  A3
LONG XASPrintf(struct USER_DATA *,char *,char *,void *)
```

FUNCTION

Does standard 'C'-style formatting to a file. Should not be called directly. See

```
ASPrintf()
.
```

INPUTS

```
User    -- Optional USER_DATA structure (used for ansi color).

buf     -- Buffer to send output to.

fmt     -- Format tring containing text and switches (see any printf()
          documentation for examples of the switches).

argptr  -- Pointer to a memory area (usually the stack) that contains
          the arguments to the formatting statements. Note that all
          arguments must be long values.
```

RESULT

The result is the number of characters output.

EXAMPLE

```
See the file '
format.c
' included with this archive for an interface
```

function to be used with this routine.

NOTES

Compatible with most printf() format strings. If the User structure is passed, the format string may include DLG %a and %b color codes. There is no floating point support nor is %x formatting supported.

BUGS

When using

```
format.c
's
ASPrintf()
```

to interface to this function, all arguments are converted to LONGs. A format of "%hd" should not be used and will cause invalid results, use "%d" instead. If you call this function directly, you should use "%hd" for SHORTs and will get the proper results.

SEE ALSO

```
AFormat()
,
XAFPrintf()
,
ASPrintf()
,
AFPrintf()
```

1.168 General STRUCTURE functions

These functions streamline the manipulation of structured data objects -- structures. They have many applications in many areas of DLG, and can be used to manipulate your own custom data items, too.

```
~~~~~AddStruct()
-- Add a structure to a file

~~~~~BinPos()
-- Binary search for a structure in a file

~~DeleteStruct()
-- Delete a structure from a file

~~DLGBinSearch()
-- Search for a structure in a sorted array

~~~~~DLGSearch()
-- Search for a structure in an array

GetFirstStruct()
-- Get the first structure from a file

~~~~~GetStruct()
```

```
-- Get a structure from a file
```

1.169 Formatted I/O functions

These I/O functions perform interaction and (in some cases) interaction with files, as well, similar to the standard C functions like printf, putchar, etc. ←

```
~~~~~AFormat()
-- * Low level I/O routine

~~~~~AFPrintf()
-- Send formatted output to a file

~~~~~ASPrintf()
-- Send formatted output to a string

~~~~~BoolQuery()
-- Asks a yes-no (Y/N) question

~~~~~Clr()
-- Clears the screen

~~~~~DispBuffer()
-- Display the contents of a buffer

~~~~~DispForm()
-- Display a file with DLG '~' switches

~~~~~DLGQuery()
-- Get input from the user intelligently

~~~~~Draw_Line()
-- Draw a line of dashes (-)

~~~~~GetChar()
-- Read a character from a user

~~~~~IntQuery()
-- Get an integer value from the user

~~~~~More()
-- Print a "More (Y/n/=)" prompt

~~~~~Pause()
-- Print a "Press Return" prompt

~~~~~PrintSpace()
-- Print spaces intelligently

~~~~~PutChar()
-- Output a character

~~~~~ReadChar()
```

```

-- Wait for a character

~~~~~SDraw_Line()
-- Draw a line of dashes (-) into a buffer

~~~~~Substitute()
-- Substitute a single "%" switch for its translated
   value

TranslateBuffer()
-- Translate % switches in a buffer

~~~~~XAFPrintf()
-- * Send formatted output to a file

~~~~~XASPrintf()
-- * Put formatted output into a string

```

Functions marked with a * should not be used directly -- use the higher level functions that call them.

1.170 Time Functions

These functions facilitate the reading, writing, and use of time

```

~~~~~AmigaTime()
-- Get the current time in seconds

~~~~~CronEvent()
-- Send a message to TpTCron

~~~~~MDate()
-- Make a timestamp of the current time

~~~~~ResumeTime()
-- resume the online clock for a port

~~~~~SMDate()
-- Make a timestamp of a specified time

~~~~~SuspendTime()
-- Suspend the online clock for a port

TimeUntilShutdown()
-- Get the number of minutes until port is shut down

~~~~~UnpackTime()
-- unpack a time value

~~~~~WhenEvent()
-- Number of seconds until specified cron event will
   occur

```


1.171 File Manipulation Functions

File manipulation functions exist for some rather specialized actions, but there are also some rather good general purpose functions as well. ↔

```
~~~~~Cat()
-- Specialized concatenate of two files

~~~~~CD()
-- Change directory

~~~~~Copy()
-- Copies one file to another

~~~~~DelDir()
-- Completely delete dir and its subdirectories

~~~~DirSize()
-- Number of bytes in a directory

~~~~Exists()
-- Check if file or dir exists

~~~FileCopy()
-- Copy all or part of a file

~~~FileSize()
-- Get the size of a file

~GetComment()
-- Get the file's comment

GetFileDate()
-- Get the date of a file

~~~~~GetPath()
-- Get the path of a file or file area

~~SearchEnd()
-- End a file search

~SearchNext()
-- Find the next file

SearchStart()
-- Start a disk search

SmartRename()
-- Rename a file intelligently

~~StripPath()
-- Get a root filename
```

1.172 LOGGING functions

These functions are used in logging actions in various ways. Some ←
are
evident on the BBS, others are not so evident

```
~AppendFile()  
  -- Append a timestamped line to a file.  
  
~~~~~DB()  
  -- Output a debugging string  
  
~~~~~Inform()  
  -- Inform a user of something  
  
WaitingMail()  
  -- Add a message to a user's waiting mail list  
  
~WriteEvent()  
  -- Write a line to a user's event log  
  
~~~WriteLog()  
  -- Write an event to the system log
```

1.173 UTILITY functions

These functions (mostly string manipulation) are of great general ←
use.

```
~~~~~ArgParse()  
  -- Parse a string into an array of words  
  
~~~~~Capitalize()  
  -- Capitalize a string  
  
~~~~~DeScore()  
  -- De-underscores a string  
  
DLGPatternMatch()  
  -- Check if a string matches a pattern  
  
~~~ScreenBuffer()  
  -- Screen a buffer for inappropriate language  
  
~~~~~ScreenPath()  
  -- screen a path for invalid characters  
  
~~~~~Stricmp()  
  -- Case insensitive string compare  
  
~~~~~StripSpaces()  
  -- Removes leading and trailing spaces from a string
```

```

~~~~~Strnicmp()
-- Case insensitive, fixed-length string compare

~~~~UnderScore()
-- Underscore a string

~~~~~Upper()
-- Convert a string to uppercase

```

1.174 BROADCAST Functions

These functions are associated with the Broadcaster, or TpTBC. ←
 Most of these are low-level and should not be used casually. Those functions are marked with an asterisk.

```

~~~~~BCGet()
-- Get a broadcast message from the resource manager

~~~~~BCMsg()
-- * Low-level broadcast routine

~~~~~BCPend()
-- Pend (suspend) automatic printing of broadcast
  messages

~~~~BCResume()
-- Resume printing of broadcast messages

~~~Broadcast()
-- Broadcast message

HandleBCMsgs()
-- Display all pending broadcast messages

```

1.175 AREA functions

These functions revolve around the use of message and file areas, ←
 and include several much-used functions. Functions marked with a * should not be casually used.

```

~~~~~BorrowArea()
-- Short term lock on an area

~~~~~DispMsg()
-- Displays the contents of a message

~~~~~EnterArea()

```

```
-- Enter an area (increase user count by 1)

~~~~~FreeArea()
-- Free a lock on an area

~~~~FreeAreaInfo()
-- Free AreaInfo structure

~~~~~GetAreaInfo()
-- Get information about an area

GetHiLowFPointers()
-- Get the high and low pointers for a file area

~GetHiLowPointers()
-- Get the high and low pointers for a message area

~~~~~GetOrigin()
-- Get the origin address of a message

~~ImportPublicMsg()
-- Import a message into a DLG message base

~~~~~KillMsg()
-- Delete a message from an area

~~~~~LeaveArea()
-- Leave (decrease user count by 1) an area

~~~~~ListAreas()
-- Display a list of available areas

~~~~~ListSIGS()
-- Display a list of available SIGs

~~~~~LockArea()
-- Lock an area for an extended period of time

PutHiLowFPointers()
-- Write the high and low pointers for a file area

~PutHiLowPointers()
-- Write the high and low pointers for a message
  area

~~~~~ReadArea()
-- Get information about an area

~~~~~ReceiveFile()
-- Receive one or more files

~~~~~ScreenMsg()
-- Screen a message for undesirable language

~~~~~SendBulletin()
-- Place a bulletin on the system
```

```

~~~~~SendFile()
-- Send one or more files

~~~SendPrivateMsg()
-- Send a private message

~~~~SendPublicMsg()
-- Send a public message

~~~~~SendRawMsg()
-- Low-level message sending routine

```

1.176 EXEC functions

These functions fire off external programs or help external programs interface with DLG. ↔

```

~~~~~CallEditor()
-- Call the user's editor and edit a file

~~ChainProgram()
-- Sets up DLG to execute another program

~~~DialogBatch()
-- Execute a DLG batch file

OverlayProgram()
-- Execute another program using current CLI

~~~ResourceMsg()
-- * Low-level resource manager interface

~~~~~SendCtlMsg()
-- * Low level handler interface

```

1.177 SERIAL functions

Take control and manipulate the serial port at your own risk... if you know ↔ what you're doing, these functions will prove extremely handy. If you're unfamiliar with serial I/O, these functions may prove extremely deadly. :-)

```

~~~~~ClearLine()
-- Flush all characters from the input line

~~~~~DLGGetSer()
-- Take control of a port's serial device

DLGProtoStatus()

```

```

-- Update the status of a transfer

~DLGReleaseSer()
-- Releas a hold on the serial device

```

1.178 RESOURCE functions

These functions deal with various resources and such, including ↔ menus. The ones with an asterisk shouldn't be used casually.

```

~~~~~FreeMenu()
-- * Free a menu

~FreeResource()
-- Free a misc resource

FreeResReport()
-- Free a resource report

~~~~~GetLang()
-- Get the language information for a port

~GetResReport()
-- Get information about many resources

~~~~~LoadLang()
-- Load a language

~~~~~LockMenu()
-- * Lock a menu

~LockResource()
-- Get a lock on a misc resource

~~~~PurgeMenu()
-- * Remove a menu from use

```

1.179 AFPrintf()

NAME

AFPrintf -- Send formatted output to a file

SYNOPSIS

```

result = AFPrintf(User, fh, fmt, argptr)
           A0  A1 A2  A3
LONG AFPrintf(struct USER_DATA *, BPTR, char *, void *)

```

FUNCTION

Does standard 'C'-style formatting to a file.

INPUTS

User -- Optional USER_DATA structure (used for ansi color).

fh -- AmigaDOS file handle to send output to.

fmt -- Format string containing text and switches (see any printf() documentation for examples of the switches).

argptr -- Pointer to a memory area (usually the stack) that contains the arguments to the formatting statements. Note that all arguments must be long values.

RESULT

The result is the number of characters output.

EXAMPLE

```
numchars = AFPrintf(out,NULL,"This is to file %s\n",outfilename);
numchars = AFPrintf(Output(),&userdata,"This is to user %s\n",
                    username);
```

NOTES

Compatible with most printf() format strings. If the User structure is passed, the format string may include DLG %a and %b color codes. There is no floating point support nor is %x formatting supported.

BUGS

Arguments are converted to LONGs. A format of "%hd" should not be used and will cause invalid results, use "%d" instead. If you call this function directly, you should use "%hd" for SHORTs and will get the proper results.

SEE ALSO

```
XAFPrintf()
,
XASPrintf()
,
AFormat()
,
ASPrintf()
```

1.180 format.c

```
/* *****
 *
 *  © copyright 1995-96 by DLG Development
 *  All rights reserved
 *
 * *****
 *
 *  /* Interface routines for calling the XAFPrintf() and XASPrintf() routines
 *  ** in dlgl.library. You can compile this with your program or copy the
 *  ** following routines into your code directly.
 *  */
```

```

#include <exec/types.h>
#include <dos/dosextens.h>
#include <dialog/user.h>

#include <proto/dos.h>
#include <proto/dlg.h>

extern struct Library *DLGBase;

LONG __stdargs AFPrintf(struct USER_DATA *User,BPTR fh,char *fmt,...)
{
    return(XAFPrintf(User,fh,fmt,(char *)(&fmt+1)));
}

LONG __stdargs ASPrintf(struct USER_DATA *User,char *str,char *fmt,...)
{
    return(XASPrintf(User,str,fmt,(char *)(&fmt+1)));
}

```

1.181 ASPrintf()

NAME

ASPrintf -- Put formatted output into a string

SYNOPSIS

```

result = ASPrintf(User,buf,fmt,argptr)
           A0  A1  A2  A3
LONG ASPrintf(struct USER_DATA *,char *,char *,void *)

```

FUNCTION

Does standard 'C'-style formatting to a file.

INPUTS

User -- Optional USER_DATA structure (used for ansi color).

buf -- Buffer to send output to.

fmt -- Format tring containing text and switches (see any printf() documentation for examples of the switches).

argptr -- Pointer to a memory area (usually the stack) that contains the arguments to the formatting statements. Note that all arguments must be long values.

RESULT

The result is the number of characters output.

EXAMPLE

```

numchars = ASPrintf(string,NULL,"This is with no user\n");
numchars = ASPrintf(string,userdat,"This is tailored for %s's
                    account\n",UserName);

```

NOTES

Compatible with most `printf()` format strings. If the User structure is passed, the format string may include DLG %a and %b color codes. There is no floating point support nor is %x formatting supported.

BUGS

Arguments are converted to LONGs. A format of "%hd" should not be used and will cause invalid results, use "%d" instead. If you call this function directly, you should use "%hd" for SHORTs and will get the proper results.

SEE ALSO

```
XASPrintf()  
,  
AFormat()  
,  
XAFPrintf()  
,  
AFPrintf()
```

1.182 Using this guide

Using the DLG 1.1 Programming Guide

Included with this guide, you will find a directory called INCLUDE which includes the header files necessary for developing DLG code correctly. If you assign INCLUDE: to that directory, various links in the documentation will take you directly to the specific data structures and flags mentioned in the autodocs. Alternatively, if you already have an Include: assignment, you can copy those directories over to that directory.

All functions in this guide require you to open the DLG library before using them. This is accomplished using the AmigaDOS `OpenLibrary()` function:

```
struct Library *DLGBase;  
  
DLGBase = OpenLibrary("dlg.library",2L);  
  
if(!DLGBase) exit(10);
```

The above example assumes that you are opening DLG.library version 2 and that you will exit with an error code of 10 if it fails.

Also, before your program shuts down, you should close the library. Your program will not crash if you do not, but it is good form to do so, and keeps the open count of the library accurate.

```
if(DLGBase) CloseLibrary(DLGBase);
```

Note that this accounts for the library being closed earlier and doesn't cause an error if the library is already closed.

Prototypes

Before using any functions, you need to have declared their prototype. All DLG functions are prototyped in `proto/dlg.h`. `#include` this file in your program and your prototyping needs are taken care of.

A note about the include files

The include files provided are for use in development of code for DLG utilities. The files that most third party developers will be primarily concerned with are in the `include:dialog/` directory.

Although they correctly reflect the structures, variables, and flags used by DLG, no absolute guarantee of their accuracy is made. We will make every effort to insure their accuracy and will publish updated files as needed.

With few exceptions, the include files may be used for DLG 1.0 development, as well, but do not assume that everything is the same. One area that is certainly different is the actual functions -- there are many new functions available in the DLG 1.1 library that are not available under DLG 1.0.

Any structure, variable, or flag marked "For internal use" should not be tampered with or used unless absolutely necessary -- and if they are, further compatibility is not guaranteed.

Any part of a structure marked as "filler" or "unused" should not be touched! It may be used in later versions of DLG, thus causing your program to fail or cause failures.

Final note

While later versions of DLG are available, we will continue to refer to DLG as version 1.1. No significant changes to the handler, resource manager, or library will take place within the same revision, thus, you can consider DLG 1.16 to be DLG 1.1 for the purposes of this document.

1.183 Distribution

Distribution

This archive may be distributed freely on any electronic bulletin board system or information service provided that the contents of the archive remain unchanged with no additions, deletions, or modifications of existing files.

This archive should be made available to anyone wishing to develop software for use with DLG Pro 1.1, regardless of whether or not they own the software.

1.184 Copyright

Copyright

The contents of this archive are copyright (c) 1995-1996 by DLG Development. All rights reserved.

Permission is given for use in the development of legitimate software for use with DLG Pro BBOS or in conjunction with it.

DLG Development reserves the right to suspend this permission on a case by case basis if needed to protect the interests of its customers.

1.185 Credits

Credits

This programming document would not have been possible without the work of many individuals, so please forgive me if I leave anyone out, it is purely unintentional.

- o Tom Conroy, Mike Oliphant, and James Hastings-Trew, the original developers of DLG, who got us to version 1.0 and gave us a great package!
- o Steve Lewis, next keeper of the source and developer of PDQMail and DLGMail, some very excellent mailer software even after all this time.
- o The DLG 1.1 beta team: Mike Moon, Glyn Hughes, Jon Godfrey, Jon Guidry, John Veldthius, Don Plesky; for thier long suffering and patience with scrambled hard drives, corrupt memory pointers, and a general pain in the nether regions.
- o Holly Sullivan, for infinite patience with a Significant Other who seems to be perpetually locked into Code Mode.

And last but not least, by a long shot:

- o Bob Stouder, who has performed with brilliance and grace under fire, stomped bugs with boots of iron, and shown the infinite patience of a Zen Master as he explained the intricate inner workings of DLG to the new kid on the block. May your stacks never overflow.

1.186 Contacts

Contacting DLG Development

Our preferred method of contact when dealing with DLG development is in the FidoNet echo DLG_DEV, available from any of our support sites (see below). This echo exists for any and all DLG third-party developers who need close

communication with the authors of the software.

Questions can also be directly mailed to the development team; Jeff Grimmett, at 1:202/720 either from your own system or routed from one of the support sites. One may also contact me as jeff_grimmett@elric.maximumaccess.com.

Support Sites

We have a number of support sites around the world. One of them is bound to be within reasonable reach of you, or can provide you with a pointer to a linked system that is close to you.

Please see the enclosed Support.Site document for more information.

Additional information as it happens is available on the DLG web page: <http://www.ald.net/dlg>.

1.187 Index

Alphabetical Index of DLG Functions in this guide

~

-- A --

~ActivatePort () ~~~~~

~AddArea () ~~~~~

~AddStruct () ~~~~~

~AFormat () ~~~~~

~AFPrintf () ~~~~~

~Age () ~~~~~

~AmigaTime () ~~~~~

~AppendFile () ~~~~~

~ArgParse () ~~~~~

~ASPrintf () ~~~~~

-- B --

~BCGet ~~~~~

~BCMsg () ~~~~~

~BCPend () ~~~~~

~BCResume () ~~~~~~
~BinPos () ~~~~~~
~BoolQuery () ~~~~~~
~BorrowArea () ~~~~~~
~BroadCast () ~~~~~~

-- C --

~CallEditor () ~~~~~~
~Capitalize () ~~~~~~
~Cat () ~~~~~~
~CD () ~~~~~~
~ChainProgram () ~~~~~~
~CheckUser () ~~~~~~
~ClearLine () ~~~~~~
~CloseGroup () ~~~~~~
~Clr () ~~~~~~
~Copy () ~~~~~~
~CronEvent () ~~~~~~

-- D --

~DB () ~~~~~~
~DeActivatePort () ~~~~
~DelArea () ~~~~~~
~DelDir () ~~~~~~
~DeleteStruct () ~~~~~~
~DeScore () ~~~~~~
~DialogBatch () ~~~~~~
~DirSize () ~~~~~~
~DispBuffer () ~~~~~~
~DispForm () ~~~~~~
~DispMsg () ~~~~~~

```
~DLGBinSearch () ~~~~~
~DLGGetSer () ~~~~~
~DLGPatternMatch () ~
~DLGProtoStatus () ~
~DLGQuery () ~~~~~
~DLGReleaseSer () ~
~DLGSearch () ~~~~~
~Draw_Line () ~~~~~
-- E --

~EnterArea () ~~~~~
~Exists () ~~~~~
~ExistsGlobalArea () ~
-- F --

~FileCopy () ~~~~~
~FileSize () ~~~~~
~FreeArea () ~~~~~
~FreeAreaInfo () ~
~FreeMenu () ~~~~~
~FreePort () ~~~~~
~FreePortInfo () ~
~FreeResource () ~
~FreeResReport () ~
-- G --

~GetAreaInfo () ~
~GetChar () ~~~~~
~GetComment () ~
~GetComputerType () ~
~GetDevName () ~
```

```
~GetFileDate () ~~~~~~
~GetFirstStruct () ~~~
~GetHiLowFPointers ()
~GetHiLowPointers () ~
~GetLang () ~~~~~~
~GetLevel () ~~~~~~
~GetOrigin () ~~~~~~
~GetPath () ~~~~~~
~GetPortInfo () ~~~~~~
~GetResReport () ~~~~~~
~GetStruct () ~~~~~~
-- H --

~HandleBCMsgs () ~~~~~~
-- I --

~ImmedLockPort () ~~~~
~ImportPublicMsg () ~~
~Inform () ~~~~~~
~IntQuery () ~~~~~~
-- J --

-- K --

~KillMsg () ~~~~~~
-- L --

~LeaveArea () ~~~~~~
~ListAreas () ~~~~~~
~ListPorts () ~~~~~~
~ListSIGS () ~~~~~~
~LoadLang () ~~~~~~
~LockArea () ~~~~~~
~LockMenu () ~~~~~~
```

```
~LockPort () ~~~~~~
~LockResource () ~~~~~
~Logout () ~~~~~~
-- M --

~MDate () ~~~~~~
~More () ~~~~~~
-- N --

~NextInGroup () ~~~~~
-- O --

~OpenGroup () ~~~~~~
~OverlayProgram () ~~~
-- P --

~Pause () ~~~~~~
~PrintSpace () ~~~~~~
~PurgeMenu () ~~~~~~
~PutChar () ~~~~~~
~PutHiLowFPointers ()
~PutHiLowPointers () ~
-- Q --

-- R --

~ReadArea () ~~~~~~
~ReadChar () ~~~~~~
~ReadRam () ~~~~~~
~ReadUser () ~~~~~~
~ReceiveFile () ~~~~~~
~ResourceMsg () ~~~~~~
~ResumeTime () ~~~~~~
-- S --
```

~ScreenBuffer () ~~~~~
~ScreenMsg () ~~~~~
~ScreenPath () ~~~~~
~SDraw_Line () ~~~~~
~SearchEnd () ~~~~~
~SearchNext () ~~~~~
~SearchStart () ~~~~~
~SendBulletin () ~~~~~
~SendCtlMsg () ~~~~~
~SendFile () ~~~~~
~SendPrivateMsg () ~~~
~SendPublicMsg () ~~~~
~SendRawMsg () ~~~~~
~SmartRename () ~~~~~
~SMDate () ~~~~~
~Stricmp () ~~~~~
~StripPath () ~~~~~
~StripSpaces () ~~~~~
~Strnicmp () ~~~~~
~Substitute () ~~~~~
~SuspendTime () ~~~~~

~TBaud () ~~~~~
~TCheckCarrier () ~~~~
~TColors () ~~~~~
~TCont () ~~~~~
~TDevQuery () ~~~~~
~TFreeze () ~~~~~
~TGetTitle () ~~~~~

-- T --

```
~TimeUntilShutdown()
~TInTrans() ~~~~~~
~TKill() ~~~~~~
~TOutTrans() ~~~~~~
~TransferPortLock() ~
~TranslateBuffer() ~
~TRecover() ~~~~~~
~TScreen() ~~~~~~
~TSendBreak() ~~~~~~
~TSetFlags() ~~~~~~
~TString() ~~~~~~
~TTimeDelay() ~~~~~~
~TTitle() ~~~~~~
~TUnSetFlags() ~~~~~~
~TWindow() ~~~~~~
~TWinHeight() ~~~~~~
-- U --

~UnderScore() ~~~~~~
~UnpackTime() ~~~~~~
~Upper() ~~~~~~
-- V --

-- W --

~WaitingMail() ~~~~~~
~WhenEvent() ~~~~~~
~WriteEvent() ~~~~~~
~WriteLog() ~~~~~~
~WriteRam() ~~~~~~
~WriteUser() ~~~~~~
-- X --
```

```
~XAFPrintf() ~~~~~
```

```
~XASPrintf() ~~~~~
```

```
-- Y --
```

```
-- Z --
```
