

DOpusSDK

COLLABORATORS

	<i>TITLE :</i> DOpusSDK	
<i>ACTION</i>	<i>NAME</i>	<i>DATE</i>
WRITTEN BY		May 31, 2022
<i>SIGNATURE</i>		

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

1	DOpusSDK	1
1.1	Global table of contents	1
1.2	Copyrights	2
1.3	Contact and Support	3
1.4	Example files	4
1.5	Headers etc	4
1.6	TypeDefs etc	5
1.7	#defines (a-d)	5
1.8	#defines (e-f)	6
1.9	#defines (g-i)	7
1.10	#defines (j-o)	8
1.11	#defines (p-r)	8
1.12	#defines (s-w)	9
1.13	Module_Definition	10
1.14	AppXXX_routines	18
1.15	AllocAppMessage()	21
1.16	AppWindowData()	22
1.17	ChangeAppIcon()	23
1.18	CheckAppMessage()	24
1.19	FindAppWindow()	24
1.20	FreeAppMessage()	25
1.21	GetWBArgPath()	26
1.22	ReplyAppMessage()	26
1.23	SetAppIconMenuState()	27
1.24	SetWBArg()	27
1.25	Arg_Routines	28
1.26	ParseArgs()	28
1.27	DisposeArgs()	29
1.28	BOOPSI_gadgets	30
1.29	dopusbuttongclass	30

1.30	dopuscheckgclass	31
1.31	dopusframeclass	31
1.32	dopusiclass	32
1.33	dopuslistviewgclass	32
1.34	dopuspalettegclass	37
1.35	dopusstrgclass	38
1.36	dopusviewgclass	39
1.37	BufIO_Routines	39
1.38	CloseBuf()	39
1.39	ExamineBuf()	40
1.40	FHFFromBuf()	40
1.41	FlushBuf()	41
1.42	OpenBuf()	42
1.43	ReadBuf()	42
1.44	SeekBuf()	43
1.45	WriteBuf()	43
1.46	Clipboard_Routines	44
1.47	CloseClipBoard()	44
1.48	OpenClipBoard()	45
1.49	ReadClipString()	45
1.50	WriteClipString()	46
1.51	DiskIO_Routines	46
1.52	OpenDisk()	47
1.53	CloseDisk()	48
1.54	DOS_Routines	48
1.55	DateFromStrings()	49
1.56	DeviceFromHandler()	49
1.57	DeviceFromLock()	50
1.58	DevNameFromLock()	50
1.59	FreeDosPathList()	51
1.60	GetDosPathList()	52
1.61	GetFileVersion()	52
1.62	LaunchCLI()	53
1.63	LaunchWB()	54
1.64	ParseDateStrings()	55
1.65	SearchFile()	55
1.66	SetEnv()	56
1.67	Drag_Routines	57
1.68	FreeDragInfo()	58

1.69	GetDragImage()	59
1.70	GetDragInfo()	59
1.71	GetDragMask()	60
1.72	HideDragImage()	60
1.73	ShowDragImage()	61
1.74	StampDragImage()	61
1.75	Edit_Hook	62
1.76	FreeEditHook()	63
1.77	GetEditHook()	63
1.78	GetSecureString()	64
1.79	GUI_Routines	65
1.80	ActivateStrGad()	66
1.81	AddScrollBars()	66
1.82	BOOPSIFree()	67
1.83	DisposeBitMap()	68
1.84	DrawBox()	68
1.85	DrawFieldBox()	69
1.86	FindBOOPSIGadget()	69
1.87	GetPalette32()	70
1.88	LoadPalette32()	70
1.89	NewBitMap()	71
1.90	ScreenInfo()	72
1.91	FindPubScreen()	72
1.92	SetBusyPointer()	73
1.93	FreeCachedDiskObject()	73
1.94	GetCachedDefDiskObject()	74
1.95	GetCachedDiskObject()	74
1.96	GetCachedDiskObjectNew()	75
1.97	GetIconFlags()	76
1.98	GetIconPosition()	76
1.99	SetIconFlags()	77
1.100	SetIconPosition()	78
1.101	CopyFileIcon()	78
1.102	IFF_Routines	79
1.103	IFFChunkID()	80
1.104	IFFChunkRemain()	80
1.105	IFFChunkSize()	81
1.106	IFFClose()	81
1.107	IFFFailure()	82

1.108IFFGetForm()	82
1.109IFFNextChunk()	83
1.110IFFOpen()	84
1.111IFFPopChunk()	85
1.112IFFPushChunk()	85
1.113IFFReadChunkBytes()	86
1.114IFFWriteChunkBytes()	87
1.115IFFWriteChunk()	87
1.116Image_Routines	88
1.117CloseImage()	89
1.118CopyImage()	89
1.119FreeImageRemap()	90
1.120FreeRemapImage()	90
1.121GetImageAttrs()	91
1.122GetImagePalette()	91
1.123OpenImage()	92
1.124RemapImage()	93
1.125RenderImage()	94
1.126IPC_Routines	96
1.127IPC_Command()	96
1.128IPC_FindProc()	97
1.129IPC_Flush()	98
1.130IPC_Free()	98
1.131IPC_Launch()	99
1.132IPC_ListCommand()	100
1.133IPC_ProcStartup()	101
1.134IPC_Reply()	102
1.135Layout_Routines	103
1.136AddObjectList()	114
1.137AddWindowMenus()	115
1.138BoundsCheckGadget()	116
1.139BuildMenuStrip()	117
1.140CheckObjectArea()	118
1.141ClearWindowBusy()	118
1.142CloseConfigWindow()	118
1.143DisableObject()	119
1.144DisplayObject()	120
1.145EndRefreshConfigWindow()	120
1.146FindMenuItem()	121

1.147FreeObjectList()	121
1.148FreeWindowMenus()	122
1.149GetGadgetValue()	123
1.150GetObject()	124
1.151GetObjectRect()	124
1.152GetWindowAppPort()	125
1.153GetWindowID()	125
1.154GetWindowMsg()	126
1.155LayoutResize()	126
1.156OpenConfigWindow()	127
1.157ReplyWindowMsg()	127
1.158SetConfigWindowLimits()	128
1.159SetGadgetChoices()	128
1.160SetGadgetValue()	129
1.161SetWindowBusy()	130
1.162SetWindowID()	131
1.163StartRefreshConfigWindow()	132
1.164List_Routines	132
1.165AddSorted()	133
1.166Att_ChangeNodeName()	134
1.167Att_FindNode()	134
1.168Att_FindNodeData()	135
1.169Att_FindNodeNumber()	135
1.170Att_NewList()	136
1.171Att_NewNode()	137
1.172Att_NodeCount()	138
1.173Att_NodeDataNumber()	138
1.174Att_NodeName()	139
1.175Att_NodeNumber()	140
1.176Att_PosNode()	140
1.177Att_RemList()	141
1.178Att_RemNode()	142
1.179FindNameI()	142
1.180GetSemaphore()	143
1.181InitListLock()	143
1.182IsListLockEmpty()	144
1.183LockAttList()	144
1.184SwapListNodes()	145
1.185UnlockAttList()	145

1.186Locale_Routines	146
1.187DOpusGetString()	146
1.188Memory_Routines	147
1.189AllocMemH()	147
1.190ClearMemHandle()	148
1.191FreeMemH()	149
1.192FreeMemHandle()	149
1.193NewMemHandle()	150
1.194Misc_Routines	151
1.195Atoh()	152
1.196BtoCStr()	152
1.197BuildKeyString()	152
1.198BytesToString()	153
1.199ConvertRawKey()	154
1.200DivideToString()	154
1.201DivideU()	155
1.202Itoa()	155
1.203ItoaU()	156
1.204QualValid()	156
1.205Random()	157
1.206StrCombine()	157
1.207StrConcat()	158
1.208Seed()	158
1.209Notify_Routines	159
1.210AddNotifyRequest()	159
1.211RemoveNotifyRequest()	161
1.212ReplyFreeMsg()	162
1.213SetNotifyRequest()	162
1.214Popup_Routines	163
1.215DoPopUpMenu()	163
1.216GetPopUpItem()	165
1.217Progress_Routines	166
1.218CheckProgressAbort()	166
1.219CloseProgressWindow()	167
1.220GetProgressWindow()	167
1.221HideProgressWindow()	168
1.222OpenProgressWindow()	168
1.223SetProgressWindow()	170
1.224ShowProgressWindow()	171

1.225Requester_Routines	171
1.226AsyncRequest()	172
1.227OpenStatusWindow()	174
1.228SelectionList()	175
1.229SetStatusText()	176
1.230Timer_Routines	176
1.231AllocTimer()	177
1.232CheckTimer()	177
1.233FreeTimer()	178
1.234GetTimerBase()	178
1.235StartTimer()	179
1.236StopTimer()	180
1.237TimerActive()	180
1.238Index	181

Chapter 1

DOpusSDK

1.1 Global table of contents

Directory Opus 5.5
Software Development Kit 1.0
(c) 1996 Jonathan Potter & GPSoftware 1996

The Opus SDK kit allows you to access the functions in the `dopus5.library`, and create your own modules, applications or other programs that use the power of Directory Opus.

The contents of the SDK is as follows :

```
docs - Documentation for the dopus5.library
AGDocs - Documentation in AmigaGuide format for the dopus5.library
include - Include files
lib - Linker files

example
- Example source code including:-
source - Source to the modinit.o module
```

The include and linker files are all designed for use with a C compiler (the example source code is set up to compile under SAS/C).

`dopus5.library` AutoDocs Index

Writing an Opus 5.5 Module

Header files - structures

Header files - typedefs and defines

The `dopus5.library` contains many useful functions. For convenience ↔
, they

have been grouped into meaningful areas:

AppIcon/AppWindow support

BOOPSI Gadgets

Buffered I/O

Clipboard string handling

Custom string Edit Hook

Disk I/O

DOS support

Drag and Drop routines

GUI layout routines

GUI support

IFF reading/writing

Image handling

Inter-Process Communication

List management

Locale (language) support

Memory handling/pooling

Miscellaneous

Opus Notification

Popup Menus

Progress Indicator

Requesters

String argument parsing

Timer handling

Please read the following :

Copyrights

Contact Details

1.2 Copyrights

Copyrights and Notices

Directory Opus 5 is (c) Jonathan Potter and GPSoftware 1995-1996.

This collection of developer materials is (c) GPSoftware but may be distributed free of charge to owners of Opus 5 to assist in the development of supporting modules and programs to be run with Dopus 5.5 providing this archive is distributed in its entirety. No part of this archive may be reproduced separately in any form whatsoever without written permission from GPSoftware.

Although we have taken all care in assembling these development resources, the information is provided 'as is' without any guarantee or warranty as to the performance etc etc. Neither GPSoftware, Jonathan Potter nor Dr Greg Perry accept liability for the accuracy or the use of these materials.

--

Dr Greg Perry, GPSoftware, September 9th 1996
PO Box 570, Ashgrove, Qld Australia 4060 Ph/fax +61 7 33661402
Internet Email: zzgerry@mailbox.uq.oz.au
WWW : <http://www.livewire.com.au/gpsoft/>

1.3 Contact and Support

Contact and Support

As well as the WWW pages located at

<http://www.livewire.com.au/gpsoft/>

we maintain a number of mailing lists for Directory Opus users. These are designed to provide general comment and limited support for registered users of Opus 5.5.

A) General Mailing list: dopus5

This is a mailing list for general discussion relating to general use and comments for Opus 5. To subscribe to this list, send mail to listserv@lss.com.au with :

```
subscribe dopus5 <Your Name>
```

in the message body. You will be automatically sent a brief welcome message, with instructions on how to post to the list.

B) Developer Mailing list: dopus5-dev

There is a mailing list for the discussion of programming issues relating to Opus and the Opus SDK. To subscribe to this list, send mail to listserv@lss.com.au with :

```
subscribe dopus5-dev <Your Name>
```

in the message body. You will be automatically sent a brief welcome

message, with instructions on how to post to the list.

1.4 Example files

The 'example' directory contains the following example source code:-

module

A basic module that adds one command to Opus and opens a requester. Shows a simple example of creating an Opus module.

envoymodule

A module that lets you set network information for files with Envoy. Shows an example of a simple user interface, using the Opus callback function and using a progress indicator.

iconclock

The source to the icon clock module. Shows how to write a module that is called on startup and remains resident. Also has an example of the new AppIcon features of Opus.

viewfont

The source to the ViewFont program. Shows how to create a more complex, resizable user interface, and how to access menus.

1.5 Headers etc

Opus 5.5 Header files

dopus/appicon.h	dopus/args.h	dopus/bufferedio.h
dopus/clipboard.h	dopus/diskio.h	dopus/dopusbase.h
dopus/dos.h	dopus/drag.h	dopus/edithook.h
dopus/gui.h	dopus/icon.h	dopus/iff.h
dopus/images.h	dopus/ipc.h	dopus/layout.h
dopus/lists.h	dopus/locale.h	dopus/memory.h
dopus/misc.h	dopus/modules.h	dopus/notify.h
dopus/popup.h	dopus/progress.h	dopus/requesters.h
dopus/timer.h	pragmas/dopus_pragmas.h	

Opus 5.5 Structs/unions

_Att_List	_Att_Node	_DOpusAppMessage
_DragInfo	_GL_Object	_IPC
_ObjectList	addfile_packet	AppSnapshotMsg
command_packet	delfile_packet	DOpusLocale

DOpusScreenData	endentry_packet	function_entry
gpResize	ListLock	loadfile_packet
path_node	replacereq_packet	TimerHandle

1.6 TypeDefs etc

Opus 5.5 Typedefs

Att_List	Att_Node	ConfigWindow
DiskHandle	DOpusAppMessage	DOpusNotify
DragInfo	DragInfoExtra	FuncArgs
GL_Object	ImageRemap	IPCData
IPCMessage	ListViewDraw	MenuData
ModuleFunction	ModuleInfo	NewConfigWindow
ObjectDef	ObjectList	OpenImageInfo
PopUpItem	PopUpMenu	TimerHandle
WindowData	WindowID	

Opus 5.5 #defines

a-d

e-f

g-i

j-o

p-r

s-w

1.7 #defines (a-d)

Opus 5.5 #defines (a-d)

ADDNODEF_EXCLUSIVE	ADDNODEF_NUMSORT
ADDNODEF_PRI	ADDNODEF_SORT
APPSNAPF_CLOSE	APPSNAPF_HELP
APPSNAPF_INFO	APPSNAPF_MENU
APPSNAPF_UNSNAPSHOT	APPSNAPF_WINDOW_POS
AR_Buffer	AR_BufLen
AR_Button	AR_ButtonCode
AR_CheckMark	AR_CheckPtr
AR_Flags	AR_History
AR_Message	AR_Requester
AR_Screen	AR_Title
AR_Window	AREA()
AREAFLAG_ERASE	AREAFLAG_ICON
AREAFLAG_LINE	AREAFLAG_NOFILL

AREAFLAG_OPTIM	AREAFLAG_RAISED
AREAFLAG_RECESSED	AREAFLAG_THIN
AREAFLAG_TITLE	BUTTONFLAG_CANCEL_BUTTON
BUTTONFLAG_OKAY_BUTTON	BUTTONFLAG_THIN_BORDERS
BUTTONFLAG_TOGGLE_SELECT	CAIF_BUSY
CAIF_LOCKED	CAIF_RENDER
CAIF_SELECT	CAIF_SET
CAIF_TITLE	CAIF_UNBUSY
CFGDATA()	COMMANDF_RESULT
DAE_Background	DAE_Close
DAE_Info	DAE_Local
DAE_Locked	DAE_Menu
DAE_MenuBase	DAE_SnapShot
DAE_ToggleMenu	DAE_ToggleMenuSel
DAPPF_ICON_DROP	DATA()
DFB_DefPath	DIA_FrontPen
DIA_Type	DIR_BUTTON_KIND
DIR_GLASS_KIND	DLV_Check
DLV_DoubleClick	DLV_DragNotify
DLV_DrawLine	DLV_Flags
DLV_GetLine	DLV_Highlight
DLV_Labels	DLV_Lines
DLV_MakeVisible	DLV_MultiSelect
DLV_NoScroller	DLV_Object
DLV_ReadOnly	DLV_RightJustify
DLV_ScrollDown	DLV_ScrollUp
DLV_ScrollWidth	DLV_Selected
DLV_SelectNext	DLV_SelectPrevious
DLV_ShowChecks	DLV_ShowFileNames
DLV_ShowSelected	DLV_TextAttr
DLV_Top	DLV_TopJustify
DN_APP_ICON_LIST	DN_APP_MENU_LIST
DN_CLOSE_WORKBENCH	DN_DISKCHANGE
DN_DOS_ACTION	DN_OPEN_WORKBENCH
DN_OPUS_HIDE	DN_OPUS_QUIT
DN_OPUS_SHOW	DN_OPUS_START
DN_RESET_WORKBENCH	DN_REXX_UP
DN_WRITE_ICON	DNF_DOS_CLOSE
DNF_DOS_CREATE	DNF_DOS_CREATEDIR
DNF_DOS_DELETEFILE	DNF_DOS_RELABEL
DNF_DOS_RENAME	DNF_DOS_SETCOMMENT
DNF_DOS_SETFILEDATE	DNF_DOS_SETPROTECTION
DNF_ICON_CHANGED	DNF_ICON_REMOVED
DPG_Pen	DPG_SelectNext
DPG_SelectPrevious	DRAGF_DONE_GELS
DRAGF_NO_LOCK	DRAGF_OPAQUE
DRAGF_TRANSPARENT	DRAGF_VALID
DRAWINFO()	

1.8 #defines (e-f)

Opus 5.5 #defines (e-f)

EDITF_NO_SELECT_NEXT	EDITF_PATH_FILTER
EDITF_SECURE	EH_ChangeSigBit

EH_ChangeSigTask	EH_History
EXT_FUNC ()	EXTCMD_ADD_FILE
EXTCMD_CHECK_ABORT	EXTCMD_DEL_FILE
EXTCMD_DO_CHANGES	EXTCMD_END_DEST
EXTCMD_END_ENTRY	EXTCMD_END_SOURCE
EXTCMD_ENTRY_COUNT	EXTCMD_FREE_SCREENDATA
EXTCMD_GET_DEST	EXTCMD_GET_ENTRY
EXTCMD_GET_HELP	EXTCMD_GET_PORT
EXTCMD_GET_SCREEN	EXTCMD_GET_SCREENDATA
EXTCMD_GET_SOURCE	EXTCMD_GET_WINDOW
EXTCMD_LOAD_FILE	EXTCMD_NEXT_SOURCE
EXTCMD_RELOAD_ENTRY	EXTCMD_REMOVE_ENTRY
EXTCMD_REPLACE_REQ	EXTCMD_SEND_COMMAND
EXTCMD_UNLOCK_SOURCE	FIELD_KIND
FILE_BUTTON_KIND	FILE_GLASS_KIND
FILEBUTFLAG_SAVE	FONT_BUTTON_KIND
FPOS_TEXT_OFFSET	FRAME_KIND
FUNCF_CAN_DO_ICONS	FUNCF_NEED_DEST
FUNCF_NEED_DIRS	FUNCF_NEED_ENTRIES
FUNCF_NEED_FILES	FUNCF_NEED_SOURCE
FUNCF_PRIVATE	FUNCF_SINGLE_DEST
FUNCF_SINGLE_SOURCE	FUNCF_WANT_DEST
FUNCF_WANT_ENTRIES	FUNCF_WANT_SOURCE
FUNCID_STARTUP	

1.9 #defines (g-i)

Opus 5.5 #defines (g-i)

GAD_ID_ICONIFY	GADGET ()
GADGET_NUMBER ()	GADGET_SEL ()
GADGET_SPECIAL ()	GADGET_STRING ()
GADNUMBER ()	GADSEL ()
GADSPECIAL ()	GADSTRING ()
GM_RESIZE	GTCustom_Bold
GTCustom_Borderless	GTCustom_CallBack
GTCustom_ChangeSigBit	GTCustom_ChangeSigTask
GTCustom_Control	GTCustom_CopyTags
GTCustom_FontPenCount	GTCustom_FontPens
GTCustom_FontPenTable	GTCustom_FrameFlags
GTCustom_History	GTCustom_Image
GTCustom_Integer	GTCustom_Justify
GTCustom_LayoutPos	GTCustom_LayoutRel
GTCustom_LocaleKey	GTCustom_LocaleLabels
GTCustom_MinMax	GTCustom_NoGhost
GTCustom_NoSelectNext	GTCustom_PathFilter
GTCustom_Secure	GTCustom_Style
GTCustom_TextAttr	GTCustom_TextPlacement
GTCustom_ThinBorders	HOOKTYPE_STANDARD
HOTKEY_KIND	ICONF_NO_BORDER
ICONF_NO_LABEL	ICONF_POSITION_OK
ID_AFS_MULTI	ID_AFS_PRO
ID_AFS_USER	ID_PFS_FLOPPY
ID_PFS_HARD	IDCMP_FUNC ()
IFF_CLIP	IFF_CLIP_READ

IFF_CLIP_WRITE	IFF_READ
IFF_SAFE	IFF_WRITE
IM_ARROW_DOWN	IM_ARROW_UP
IM_BBOX	IM_BORDER_BOX
IM_CHECK	IM_ClipBoundary
IM_CROSS	IM_Depth
IM_DRAWER	IM_Erase
IM_Height	IM_ICONIFY
IM_LOCK	IM_Mask
IM_NoDrawInvalid	IM_NoIconRemap
IM_Rectangle	IM_State
IM_Width	IPCDATA()
IPCF_GETPATH	IPCM_STACK()
IPCSIG_HIDE	IPCSIG_QUIT
IPCSIG_SHOW	IRF_PRECISION_EXACT
IRF_PRECISION_GUI	IRF_PRECISION_ICON
IRF_REMAP_COLO	IS_GADTOOLS()

1.10 #defines (j-o)

Opus 5.5 #defines (j-o)

JUSTIFY_CENTER	JUSTIFY_LEFT
JUSTIFY_RIGHT	LAYOUTF_BOTTOM_ALIGN
LAYOUTF_LEFT_ALIGN	LAYOUTF_RIGHT_ALIGN
LAYOUTF_SAME_HEIGHT	LAYOUTF_SAME_WIDTH
LAYOUTF_TOP_ALIGN	LFF_ICON
LISTF_LOCK	LISTF_POOL
LISTVIEWFLAG_CURSOR_KEYS	lve_Flags
lve_Pen	LVEF_SELECTED
LVEF_TEMP	LVEF_USE_PEN
MENUFLAG_AUTO_MUTEX	MENUFLAG_COMM_SEQ
MENUFLAG_GET_SEQ()	MENUFLAG_MAKE_SEQ()
MENUFLAG_TEXT_STRING	MENUFLAG_USE_SEQ
MENUID()	MODULEF_CALL_STARTUP
MODULEF_STARTUP_SYNC	MTYPE_APPSNAAPSHOT
NM_BAR_LABEL	NM_NEXT
NT_DOPUS_NOTIFY	OBJECTF_HOTKEY
OBJECTF_INTEGER	OBJECTF_NO_SELECT_NEXT
OBJECTF_PATH_FILTER	OBJECTF_READ_ONLY
OBJECTF_SECURE	OBJECTFLAG_DRAWN
OBJLIST()	OD_AREA
OD_END	OD_GADGET
OD_IMAGE	OD_SKIP
OD_TEXT	OPEN_SHRUNK
OPEN_SHRUNK_HORIZ	OPEN_SHRUNK_VERT
OPEN_USED_DEFAULT	OPEN_USED_TOPAZ
OPUS_LISTVIEW_KIND	

1.11 #defines (p-r)

Opus 5.5 #defines (p-r)

POPUP_BARLABEL	POPUP_HELPFLAG
POPUPF_CHECKED	POPUPF_CHECKIT
POPUPF_DISABLED	POPUPF_LOCALE
POPUPF_SUB	POPUPMF_ABOVE
POPUPMF_HELP	POPUPMF_REFRESH
POS_CENTER	POS_MOUSE_CENTER
POS_MOUSE_REL	POS_PROPORTION
POS_REL_RIGHT	POS_RIGHT_JUSTIFY
POS_SQUARE	POSFLAG_ADJUST_POS_X
POSFLAG_ADJUST_POS_Y	POSFLAG_ALIGN_POS_X
POSFLAG_ALIGN_POS_Y	PW_FileCount
PW_FileDone	PW_FileName
PW_FileNum	PW_FileSize
PW_Flags	PW_Info
PW_Screen	PW_SigBit
PW_SigTask	PW_Title
PW_Window	PWF_ABORT
PWF_FILENAME	PWF_FILESIZE
PWF_GRAPH	PWF_INFO
PWF_INVISIBLE	PWF_NOABORT
PWF_SWAP	RANGE_AFTER
RANGE_BETWEEN	RANGE_WEIRD
RECTHEIGHT()	RECTWIDTH()
REF_CALLBACK()	REMLISTF_FREEDATA
REMLISTF_SAVELIST	REPLACE_ABORT
REPLACE_LEAVE	REPLACE_REPLACE
REPLACEF_ALL	REPLY_NO_PORT
REPLY_NO_PORT_IPC	

1.12 #defines (s-w)

Opus 5.5 #defines (s-w)

SCRI_LORES	SCROLL_HORIZ
SCROLL_NOIDCMP	SCROLL_VERT
SEARCH_NOCASE	SEARCH_ONLYWORDS
SEARCH_WILDCARD	SEMF_ATTEMPT
SEMF_EXCLUSIVE	SEMF_SHARED
SET_IPCDATA()	SET_WINDOW_ID()
SIZE_MAX_LESS	SIZE_MAXIMUM
SIF_DIR_FIELD	SRF_CENTJUST
SRF_CHECKMARK	SRF_HISTORY
SRF_LONGINT	SRF_MOUSE_POS
SRF_PATH_FILTER	SRF_RIGHTJUST
SRF_SECURE	TEXTFLAG_ADJUST_TEXT
TEXTFLAG_CENTER	TEXTFLAG_NO_USCORE
TEXTFLAG_RIGHT_JUSTIFY	TEXTFLAG_TEXT_STRING
TYPE_EXT()	VISINFO()
WINDOW_AUTO_KEYS	WINDOW_AUTO_REFRESH
WINDOW_BACKDROP	WINDOW_BUTTONS
WINDOW_FUNCTION	WINDOW_GROUP
WINDOW_ICONIFY	WINDOW_LAYOUT_ADJUST

WINDOW_LISTER	WINDOW_LISTER_ICONS
WINDOW_MAGIC	WINDOW_NO_ACTIVATE
WINDOW_NO_BORDER	WINDOW_NO_CLOSE
WINDOW_OBJECT_PARENT	WINDOW_POPUP_MENU
WINDOW_REQ_FILL	WINDOW_SCREEN_PARENT
WINDOW_SIMPLE	WINDOW_SIZE_BOTTOM
WINDOW_SIZE_RIGHT	WINDOW_START
WINDOW_TEXT_VIEWER	WINDOW_UNDEFINED
WINDOW_UNKNOWN	WINDOW_USER
WINDOW_VISITOR	WINFLAG()
WINMEMORY()	WINREQUESTER()

1.13 Module_Definition

External_Module_Definition

PURPOSE

1. Introduction

Directory Opus 5 supports two types of "external modules". This document describes the main type, an AmigaDos library file. The other type, ARexx modules, are described in the main Opus 5 documentation.

Library-based modules are located in the DOpus5:Modules/ directory, and are identified by the ".module" suffix. They are standard AmigaDos libraries, with two compulsory entry points. They main contain any other entry points you want, but the first two are required to work with Opus.

The example code supplied with the SDK illustrates how to create an Opus module. While the supplied code is designed for SAS/C, it would be easy to adapt it to any other C compiler.

2. Main module entry point

The main entry point to the module is a function called `Module_Entry()`. The prototype for this function is as follows:

```
long Module_Entry( register __a0 char *args,
                  register __a1 struct Screen *screen,
                  register __a2 IPCData *ipc,
                  register __a3 IPCData *mainipc,
                  register __d0 ULONG mod_id,
                  register __d1 EXT_FUNC(callback) );
```

This function must be at offset `-0x1e` in the library, and take parameters in the specified registers.

You should never call this function yourself; it is called by Opus when the user runs one of the commands in your module. The parameters to the `Module_Entry()` function are as follows:

`args` - null-terminated argument string, contains any arguments the user supplied for the command

screen - main Opus screen pointer, you should open any requesters on this screen

ipc - a pointer to your IPCData structure - Opus launches each module command as a new process

mainipc - a pointer to the main IPCData structure for Opus

mod_id - the ID code of the command selected by the user

callback - address of Opus callback function (see below)

3. Second module entry point

The second entry point is a function that the module uses to identify itself. When Opus starts up it scans the contents of the modules directory, and calls this function in each of the modules.

The standard modinit.o module startup code supplies this function for you. It is not recommended that you change it. If you need to supply your own function, the prototype is as follows :

```
APTR Module_Identify( register __d0 long num );
```

This function must be at offset -0x24 in the library. "num" is the command ID number that Opus is enquiring about. If "num" is equal to -1, you must return a pointer to the ModuleInfo structure for the module. If "num" is equal to a valid command ID code, you must return a pointer to a description string for that command. If "num" is an invalid value, you must return 0.

4. Module identification

The contents of the module are identified with a ModuleInfo structure. All fields of the ModuleInfo structure must be initialised. The meaning of the fields is as follows:

ver - Module version number (for your own use)

name - pointer to module name, including ".module" suffix

locale_name - name of locale catalog for the module. This is opened automatically by the standard modinit.o startup code, which you should use.

flags - Module flags, see below

function_count - The number of functions in this module

function - The definition of the first function

Module flag values (for the "flags" field) are as follows:

MODULEF_CALL_STARTUP - If this flag is specified, Opus will run your module automatically on startup, with the special "mod_id" value of FUNCID_STARTUP

MODULEF_STARTUP_SYNC - If MODULEF_CALL_STARTUP is also specified, this flag causes Opus to wait for your

module to return from the startup call before continuing

The ModuleInfo structure contains room for only one function definition. If your module contains more than one function, the additional ModuleFunction structures MUST follow the ModuleInfo structure in memory. You must provide as many ModuleFunction structures as were specified in the "function_count" field of the ModuleInfo structure. For example,

```
// Module definition, includes first function
ModuleInfo
module_info={
    1,          // Version
    "example.module", // Module name
    "example.catalog", // Catalog name
    0,          // Flags
    2,          // Number of functions
    {0, "Example1", MSG_EXAMPLE1_DESC, 0, 0}};

// Second function definition follows immediately on
ModuleFunction
module_func_2={
    1, "Example2", MSG_EXAMPLE2_DESC, 0, 0};
```

5. Function definitions

The ModuleFunction structure is used to define each command that the module provides. The first function is defined with a ModuleFunction structure embedded in the ModuleInfo; additional commands must be defined after that. All fields of the ModuleFunction structure must be initialised, as follows:

id - command ID code. This value is passed as the "mod_id" parameter to the Module_Entry() function

name - name of the function. This is the actual command name that will be used to invoke this command

desc - locale string ID for the function description. This is the ID of the string in the catalog for this module that is used to describe this command in the popup command list.

flags - command flags, see below

template - command template (in ReadArgs() format). This string is displayed in the popup argument list in Opus function editors, but is not actually parsed by Opus. You will need to use the ParseArgs() routine on the "args" parameter in the Module_Entry() function.

ModuleFunction flags are as follows:

FUNCF_NEED_SOURCE - set if your module requires a valid source

directory - if one is not available, your command will not be launched

FUNCF_NEED_DEST - set if your module requires a valid destination directory

FUNCF_NEED_FILES - set if you need there to be selected files

FUNCF_NEED_DIRS - set if you need selected directories

FUNCF_CAN_DO_ICONS - set if you can operate on icons as well as normal files/directories

FUNCF_SINGLE_SOURCE - set if you can only operate on a single source lister

FUNCF_SINGLE_DEST - set if you can only operate on a single destination lister

FUNCF_WANT_DEST - set if you want a destination directory, but don't require one

FUNCF_WANT_SOURCE - set if you want a source directory, but don't require one

FUNCF_WANT_ENTRIES - set in conjunction with FUNCF_NEED_FILES or FUNCF_NEED_DIRS, to specify that you want those items but don't require them

FUNCF_PRIVATE - the function is private, it won't show up in the popup command list

6. Standard startup code

It is highly recommended that you link with the standard module startup code (modinit.o) when creating modules. This code contains the Module_Identify() function, and automatically initialises several library bases which you may need. See the <dopus/modules.h> file for more information on this file.

7. Module callback function

The "callback" parameter to the Module_Entry() function provides the address of a callback function within Opus. This function allows you to access information that your module command may need. The callback function is defined as follows:

```
ULONG callback(    register __d0 ULONG command,
                  register __a0 APTR handle,
                  register __a1 APTR packet    );
```

command - the callback command, see below for the list

handle - the callback handle. You must pass the value of IPCDATA(ipc) for this parameter ("ipc" is the argument

passed to the Module_Entry() function)

packet - a command-specific packet

Following is the list of callback commands. Each command takes a packet specific to it.

Command : EXTCMD_GET_SOURCE

Purpose : Returns the current source path

Packet : char path[256]

Returns : struct path_node *path

Notes : The packet is a pointer to a 256 character buffer, into which the current source path will be copied. The return value is a pointer to a path_node structure, which can be used with other callback commands. This structure is READ ONLY!

Command : EXTCMD_END_SOURCE

Purpose : Finishes and cleans up current source path

Packet : Set to 0

Returns : <none>

Notes : Call this command if you are aborting early and do not wish to process further source paths.

Command : EXTCMD_NEXT_SOURCE

Purpose : Gets the next source path

Packet : char path[256]

Returns : struct path_node *path

Notes : Call this command when you have finished with the first source path and want to move onto the next one. The return value is NULL if there are no more source paths.

Command : EXTCMD_UNLOCK_SOURCE

Purpose : Unlock source listers

Packet : <none>

Returns : <none>

Notes : When your module command is called, any source listers are locked automatically. Call this command when you want to unlock them (they are unlocked automatically when your module returns).

Command : EXTCMD_GET_DEST

Purpose : Returns the next destination path

Packet : char path[256]

Returns : struct path_node *path

Notes : The packet is a pointer to a 256 character buffer, into which the current destination path will be copied. The return value is a handle to the path, which can be used with other callback commands. Call this command repeatedly to move through the destination paths. When all the destination paths have been used, this command will return NULL. If you call this command again, it will start again with the first destination path.

Command : EXTCMD_END_DEST

Purpose : Ends the current destination path

Packet : FALSE to abort, TRUE to continue
Returns : <none>
Notes : You must call this command when you have finished with one destination path, prior to calling EXTCMD_GET_DEST.

Command : EXTCMD_GET_ENTRY
Purpose : Get the next entry to work with
Packet : <none>
Returns : struct function_entry {
 struct MinNode node; // Node
 char *name; // File name
 APTR entry; // Not used
 short type; // Type; <0 = file, >0 = dir
 short flags; // Not used
};

Notes : This returns a pointer to the next entry in the current source path. This structure is READ ONLY! Use the "name" field to get the entry name.

Command : EXTCMD_END_ENTRY
Purpose : Finish with specific entry
Packet : struct endentry_packet {
 struct function_entry *entry; // Entry to end
 BOOL deselect; // TRUE for deselect
};

Returns : <none>
Notes : Call this command when you have finished working with one entry and wish to move on to the next. "entry" must be set to the pointer that was returned by EXTCMD_GET_ENTRY. Set "deselect" to TRUE to have the entry deselected in the lister.

Command : EXTCMD_RELOAD_ENTRY
Purpose : Marks an entry to be reloaded
Packet : struct function_entry *entry;
Returns : <none>
Notes : This command marks the specified entry to be reloaded. When the function finishes, the entry will be reloaded to update any changes that your module might have made to it.

Command : EXTCMD_REMOVE_ENTRY
Purpose : Marks an entry to be reloaded
Packet : struct function_entry *entry;
Returns : <none>
Notes : This command marks the specified entry to be removed. When the function finishes, the entry will be removed from the lister it is in.

Command : EXTCMD_ENTRY_COUNT
Purpose : Returns total count of entries
Packet : <none>
Returns : long entry_count;
Notes : Returns the number of selected entries for the function.

Command : EXTCMD_ADD_FILE
Purpose : Adds a file to a lister

```
Packet : struct addfile_packet {
    char      *path;      // Path to add file to
    struct FileInfoBlock *fib;    // FileInfoBlock to add
    APTR      lister;    // Lister pointer
};
```

Returns : <none>

Notes : Allows you to add a file or directory to a lister. The path field points to the full path of the file to add. fib is an initialised FileInfoBlock which is used to create the file entry. The lister pointer is found in the path_node structure, which is obtained via a call to EXTCMD_GET_SOURCE or EXTCMD_GET_DEST. The display is not updated until you call EXTCMD_DO_CHANGES, or your function returns.

Command : EXTCMD_DEL_FILE

Purpose : Delete a file from a lister

```
Packet : struct delfile_packet {
    char      *path;      // Path file is in
    char      *name;      // Filename to delete
    APTR      lister;    // Lister pointer
};
```

Returns : <none>

Notes : This removes the specified file from any listers it is current shown in. The file itself is not deleted, only the display of it in the lister. The display is not updated until you call EXTCMD_DO_CHANGES, or your function returns.

Command : EXTCMD_LOAD_FILE

Purpose : Load a new file in a lister

```
Packet : struct loadfile_packet {
    char      *path;      // Path file is in
    char      *name;      // Name of file
    short     flags;      // Flags
    short     reload;     // Reload existing file
};
```

Returns : <none>

Notes : This command is similar to EXTCMD_ADD_FILE except that it Examines() the file and supplies the FileInfoBlock automatically. 'path' is the full path of the file and 'name' is the file name. The only valid flag at this time is LFF_ICON, which indicates that the icon (.info) of the supplied file is to be loaded instead of the file itself. If 'reload' is set to TRUE, an existing file will be reloaded (ie the old entry in the lister will be removed).

Command : EXTCMD_DO_CHANGES

Purpose : Perform file changes in listers

Packet : <none>

Returns : <none>

Notes : This command causes any changes made to listers by the EXTCMD_ADD_FILE, EXTCMD_DEL_FILE and EXTCMD_LOAD_FILE commands to be displayed. If your function returns without calling this command, the changes are displayed automatically.

Command : EXTCMD_CHECK_ABORT

Purpose : Check abort status in lister

Packet : <undefined>
Returns : BOOL
Notes : This command returns TRUE if your 'function' has been aborted by the user. This could have occurred because the user pressed escape or clicked the close button on a lister, or quit the program.

Command : EXTCMD_GET_WINDOW
Purpose : Get a lister's window pointer
Packet : struct path_node *path
Returns : struct Window *window
Notes : Returns a pointer to the Window for the lister specified by the path_node structure. This is useful if you want to open a requester over a lister window.

Command : EXTCMD_GET_HELP
Purpose : Get help on a topic
Packet : char *topic
Returns : <none>
Notes : This command causes Opus to open the AmigaGuide help file and search for the named topic.

Command : EXTCMD_GET_PORT
Purpose : Get ARexx port name
Packet : char name[40]
Returns : <none>
Notes : This command copies the name of the Opus ARexx port into the supplied buffer.

Command : EXTCMD_GET_SCREEN
Purpose : Get public screen name
Packet : char name[40]
Returns : <none>
Notes : This command copies the name of the Opus public screen into the supplied buffer.

Command : EXTCMD_REPLACE_REQ
Purpose : Shows a "file exists - replace?" requester
Packet : struct replacereq_packet {
 struct Window *window; // Window to open over
 struct Screen *screen; // Screen to open on
 IPCData *ipc; // Process IPC pointer
 struct FileInfoBlock *file1; // First file information
 struct FileInfoBlock *file2; // Second file information
 short flags; // Set to 0 for now
};
Returns : Result of requester; REPLACE_ABORT for abort, REPLACE_LEAVE for skip or REPLACE_REPLACE for replace. If the REPLACEF_ALL flag is set, it indicates an "All" gadget (eg Skip All, Replace All)

Command : EXTCMD_GET_SCREENDATA
Purpose : Get information about the Opus display
Packet : <none>
Returns : struct DOpusScreenData {
 struct Screen *screen; // Pointer to Opus screen
 struct DrawInfo *draw_info; // DrawInfo structure

```

    USHORT    depth;        // Depth of screen
    USHORT    pen_alloc;    // Pen allocation flag
    USHORT    pen_array[16]; // User pen array
    USHORT    pen_count;    // Number of pens;
};

```

Notes : Returns a structure with useful information about the Opus screen. This structure is READ ONLY!
Call EXTCMD_FREE_SCREENDATA to free it.

Command : EXTCMD_FREE_SCREENDATA
Purpose : Free a DOpusScreenData structure
Packet : struct DOpusScreenData *
Returns : <none>
Notes : Frees the result of an EXTCMD_GET_SCREENDATA call

```

Command : EXTCMD_SEND_COMMAND
Purpose : Send an ARexx command to DOpus
Packet : struct command_packet {
    char    *command;        // Command to send
    ULONG   flags;          // Command flags
    char    *result;        // Result pointer
    ULONG   rc;             // Result code
};

```

Returns : TRUE if the message was sent
Notes : This command allows you to send an ARexx instruction directly to the Opus ARexx port. Set the COMMANDF_RESULT flag if you want a result string returned; if one is, the 'result' field of the packet will contain a pointer to it. You MUST call FreeVec() on this pointer when you have finished with the result.

1.14 AppXXX_routines

AppXXX routines

AllocAppMessage()

AppWindowData()

ChangeAppIcon()

CheckAppMessage()

FindAppWindow()

FreeAppMessage()

GetWBArgPath()

ReplyAppMessage()

SetAppIconMenuState()

SetWBArg()

PURPOSE

The dopus5.library installs patches into the system when it loads to intercept calls to the workbench.library AddAppXXX() functions. This allows DOpus to show AppIcons, AppMenuItems, and support drag and drop onto AppWindows.

The emulation is transparent as far as a third-party application is concerned, but it is possible to access additional features that DOpus provides (especially for AppIcons). Obviously these are only available through DOpus, and not through Workbench.

Using tags with the AddAppIcon() function, it is possible to control the DOpus-only features of AppIcons. Workbench ignores these tags, and so if Workbench is running as well it will just see the plain AppIcon itself.

The tags are as follows :

DAE_Local

This tag causes the icon to only be added to DOpus. If Workbench is running as well, it will not see this icon. Supplying this tag is probably a good idea if your AppIcon depends on some of the other DOpus-specific functions.

DAE_Snapshot

Indicates that this icon can support the Snapshot function. If this tag is specified, the Snapshot item in the icon popup menu will be enabled, and the Snapshot item in the main Icon menu will work. Use of this tag generates AppSnapshotMsgs, see below for more information.

DAE_Close

Turns the 'Open' item in the icon popup menu into a 'Close' item. APPSNAPF_CLOSE will be set in the AppSnapshotMsg that is generated.

DAE_Info

Indicates that this icon can support the Information function. This is similar in operation to the DAE_Snapshot tag. The APPSNAPF_INFO flag will be set in the AppSnapshotMsg.

DAE_Menu

This tag can be used several times for one icon. It allows you to specify additional entries for the icon popup menu. ti_Data points to a string that is displayed for the menu item. The order these tags are supplied specifies the order they are displayed, and also controls the ID that is returned in AppSnapshotMsgs.

DAE_ToggleMenu

Similar to DAE_Menu, this allows you to specify a menu item for

the icon popup menu. The only difference is that the menu item is a toggle-select item (analogous to CHECKIT for Intuition menus).

DAE_ToggleMenuSel

The same as DAE_ToggleMenu, but specifies that the item is to be selected by default (analogous to CHECKED for Intuition menus).

DAE_MenuBase

This tag allows you to specify a base ID for menu IDs that are generated via the DAE_Menu, DAE_ToggleMenu and DAE_ToggleMenuSel tags. Menu IDs usually start at 0 for the first menu and increase from there. If you specify the DAE_MenuBase tag, the menu IDs will start from your supplied value.

DAE_Background

This allows you to specify a pen to use to render the background colour of the icon. If not supplied, the default is pen 0.

DAE_Locked

This tag specifies that the icon position is to be locked. That is, the user will be unable to reposition the icon from the initial coordinates. This flag can be changed later using the

ChangeAppIcon()
function.

Using the new tags can cause special messages to be sent to your message port. These are an extension of the normal AppMessages, and can be identified by an am_Type of MTYPE_APPSNAPSHOT.

The events you will get special messages for are :

Snapshot

If the DAE_Snapshot tag was specified and the user snapshots your icon, you will receive a message containing the icon position (AppSnapshotMsg->position_x and AppSnapshotMsg->position_y). It is your responsibility to save these values, and use them when adding the AppIcon in the future.

If the APPSNAPF_WINDOW_POS flag is set in the AppSnapshotMsg->flags field, the position in AppSnapshotMsg->window_pos is also valid.

Un-Snapshot

If the DAE_Snapshot tag was specified and the user unsnapshots your icon, you will receive a message with the APPSNAPF_UNSNAPSHOT flag set.

Close

If DAE_Close was specified, and the user selects the Close item

in the icon popup menu, you will receive a message with the APPSNAPF_CLOSE flag set.

Information

You will receive a message with the APPSNAPF_INFO flag set if you specified the DAE_Info tag, and the user selects Information on your icon.

Menu

If menus were added with the DAE_Menu tags, you will receive a message with the APPSNAPF_MENU flag set when the user selects one of your menu items. The AppSnapshotMsg->id field contains the item ID. If the user pressed the help key on one of the items, the APPSNAPF_HELP flag will also be set.

Directory Opus also sends additional information to AppWindows. If you receive an AppMessage of type MTYPE_APPWINDOW, you should check to see if it is an Opus message using the

```
CheckAppMessage()  
function. If so,
```

the message is a DOpusAppMessage, which contains additional information. The extra fields are :

da_DropPos

This field contains an array of Point structures. Each structure gives the offset from the origin of each file in the message. This allows you to maintain the relative positions of all icons dropped in a multiple-file operation.

da_DragOffset

This Point structure gives you the offset of the primary icon from the mouse pointer. That is, if the user clicked on the primary icon in the top-left corner, this offset would be 0,0. If they picked up the icon from the bottom-right corner, it might be 32,18.

da_Flags

The only flag supported so far is DAPPF_ICON_DROP. This indicates that the files dropped were in fact icons (ie from an icon mode lister).

The da_DropPos and da_DragOffset fields enable you to calculate the exact position that the user dropped the files on. Normal AppMessages only provide the position of the mouse pointer, which is useless if you want to maintain the relative and correct positions of the icons.

1.15 AllocAppMessage()

NAME

AllocAppMessage - allocate a DOpusAppMessage

SYNOPSIS

```
AllocAppMessage(memory, port, count)
           A0      A1      D0
```

```
DOpusAppMessage *AllocAppMessage(APTR, struct MsgPort *, short);
```

FUNCTION

This function allows you to create a DOpusAppMessage (an extended AppMessage) easily.

INPUTS

memory - memory handle or NULL (see memory.doc)
 port - address of reply port
 count - number of arguments

RESULT

Allocates a DOpusAppMessage, including space for count arguments (both da_Msg.am_ArgList and da_DropPos will be initialised).

NOTES

Unless you actually want to send an AppMessage to a DOpus window with relative icon positions, you don't really need this function. It does provide a convenient way to allocate an AppMessage, though, and there's no reason you can't use DOpusAppMessages totally in place of AppMessages if you want to.

SEE ALSO

```
FreeAppMessage()
,
SetWBArg()
```

1.16 AppWindowData()

NAME

AppWindowData - extract data from an AppWindow

SYNOPSIS

```
AppWindowData(appwindow, idptr, userdataptr)
           A0 A1      A2
```

```
struct MsgPort *AppWindowData(APTR, ULONG *, ULONG *);
```

FUNCTION

This function returns the ID, Userdata and Message port associated with the specified AppWindow. These are the parameters that are supplied in the call to AddAppWindow.

INPUTS

appwindow - AppWindow handle
 idptr - pointer to ULONG to contain the ID
 userdataptr - pointer to ULONG to contain the Userdata

RESULT

The AppWindow ID and Userdata are stored in the variables supplied, and the address of the AppWindow's message port is returned.

SEE ALSO

```
FindAppWindow()
, workbench.library/AddAppWindow()
```

1.17 ChangeAppIcon()

NAME

ChangeAppIcon - make changes to an AppIcon

SYNOPSIS

```
ChangeAppIcon(icon, render, select, label, flags)
             A0      A1      A2      A3      D0
```

```
void ChangeAppIcon
  (APTR, struct Image *, struct Image *, char *,
   ULONG);
```

FUNCTION

This function allows you to make changes to an AppIcon that was previously added to DOpus. It has no effect on the icon on Workbench, so you should use the DAE_Local tag when adding the icon if your program depends on this function.

You are able to change both frames of the icon's image and the icon's label. You can also lock or unlock the icon's position, and make it busy.

INPUTS

icon - icon to act on, as returned by AddAppIcon()
 render - new main image for the icon
 select - new select image for the icon
 label - new label for the icon
 flags - control flags. The available flags are :

```
CAIF_RENDER - change the main image
CAIF_SELECT - change the select image
CAIF_TITLE  - change the label
CAIF_LOCKED - change the 'locked' flag
CAIF_SET    - use with CAIF_LOCKED
CAIF_BUSY   - make icon busy
CAIF_UNBUSY - make icon unbusy
```

NOTES

To lock an icon, pass CAIF_LOCKED|CAIF_SET. To unlock it, pass CAIF_LOCKED by itself. The render, select and label parameters are ignored unless their corresponding flags are set. You can specify any combination of these flags at once. To reduce the visible effects, you should make as many changes with the one call as possible.

SEE ALSO

```
SetAppIconMenuState()  
, workbench.library/AddAppIcon
```

1.18 CheckAppMessage()

NAME

CheckAppMessage - check if an AppMessage is from DOpus

SYNOPSIS

```
CheckAppMessage(msg)  
A0
```

```
BOOL CheckAppMessage(DOpusAppMessage *);
```

FUNCTION

This function allows you to discover whether an AppMessage is actually an extended DOpusAppMessage.

INPUTS

msg - AppMessage to test

RESULT

Returns TRUE if the message is a DOpusAppMessage.

NOTES

You MUST only pass AppMessages (or DOpusAppMessages) to this function. Passing other types of messages (eg IntuiMessages) results in undefined behaviour.

SEE ALSO

```
AllocAppMessage()
```

1.19 FindAppWindow()

NAME

FindAppWindow - test to see if a window is an AppWindow

SYNOPSIS

```
FindAppWindow(window)  
A0
```

```
APTR FindAppWindow(struct Window *);
```

FUNCTION

This routine allows you to discover whether a Window is in fact an AppWindow.

INPUTS

window - pointer to the window to test

RESULT

Returns the AppWindow handle if it is an AppWindow, or NULL if not.

NOTES

You should only use the returned value within a Forbid()/Permit(), as the window in question could disappear at any time. Also note that the system patches are not installed until the dopus5.library is loaded. Any AppWindows added to the system before the patches are installed are undetectable.

SEE ALSO

AppWindowData()
, workbench.library/AddAppWindow()

1.20 FreeAppMessage()

NAME

FreeAppMessage - frees a DOpusAppMessage

SYNOPSIS

FreeAppMessage(msg)
A0

```
void FreeAppMessage(DOpusAppMessage *);
```

FUNCTION

This function frees the supplied DOpusAppMessage. It is only designed for messages allocated with
AllocAppMessage

.

INPUTS

msg - message to free

NOTES

You should not use this routine for AppMessages you receive (ie are sent by another process). You should ReplyMsg() those messages as normal. This function is used to free DOpusAppMessages that YOU create, usually when they are replied to by another task.

SEE ALSO

AllocAppMessage()

1.21 GetWBArgPath()

NAME

GetWBArgPath - extract pathname from WBArg

SYNOPSIS

```
GetWBArgPath(wbarg, buffer, size)
             A0      A1      D0
```

```
BOOL GetWBArgPath(struct WBArg *, char *, long);
```

FUNCTION

This function is provided as a convenient method of extracting the pathname of a file/directory from a WBArg structure (usually within an AppMessage).

INPUTS

wbarg - pointer to the WBArg structure
 buffer - buffer to write pathname to
 size - size of buffer

RESULT

The full path and name of the object referred to by the WBArg structure is copied to the supplied buffer. This routine returns TRUE if it was successful.

1.22 ReplyAppMessage()

NAME

ReplyAppMessage - reply to an AppMessage

SYNOPSIS

```
ReplyAppMessage(msg)
             A0
```

```
void ReplyAppMessage(DOpusAppMessage *);
```

FUNCTION

This function is the best way to reply to a DOpusAppMessage. Its operation is quite straightforward - if the message has a reply port set, it calls ReplyMsg() as normal. Otherwise, it calls

FreeAppMessage

. This allows messages to be sent with no reply needed. Directory Opus will never send an AppMessage without a reply port, but you might want to use this routine among your own processes.

INPUTS

msg - message to reply to

RESULT

The message is replied or freed.

SEE ALSO

FreeAppMessage()

1.23 SetAppIconMenuState()

NAME

SetAppIconMenuState - change the state of an icon popup menu

SYNOPSIS

```
SetAppIconMenuState(icon, item, state)
    A0    D0    D1
```

```
long SetAppIconMenuState(APTR, long, long);
```

FUNCTION

This allows you to set the state of a toggle-select menu item in the icon popup menu of AppIcons. These menu items would have been added with the DAE_ToggleMenu and DAE_ToggleMenuSel tags.

INPUTS

icon - icon to act on, as returned by AddAppIcon()
 item - number of item to change (in the order they were added)
 state - new state for the item (TRUE=selected)

RESULT

Returns the old selection state of the item.

NOTES

This routine uses 0 as a base ID for the menu items, even if you specified a new base with DAE_MenuBase.

SEE ALSO

```
ChangeAppIcon()
, workbench.library/AddAppIcon
```

1.24 SetWBArg()

NAME

SetWBArg - fill out a WBArg entry in a DOpusAppMessage

SYNOPSIS

```
SetWBArg(msg, item, lock, name, memory)
    A0    D0    D1    A1    A2
```

```
BOOL SetWBArg(DOpusAppMessage *, short, BPTR, char *, APTR);
```

FUNCTION

This routine makes it easy to initialise the WBArg structures in an AppMessage (or a DOpusAppMessage).

INPUTS

msg - AppMessage to initialise
item - item to initialise (starting at 0)
lock - lock on parent directory
name - name of file
memory - memory handle or NULL

RESULT

The specified WBArg in the AppMessage is initialised with the lock and name specified. This routine returns TRUE if it was successful.

NOTES

'lock' is the lock of the item's parent directory in the case of files, or on the item itself in the case of directories. For files, 'name' is the name of the file. 'name' is null for directories. The lock and name you supply are both copied, so they do not need to remain valid once this call is complete.

1.25 Arg_Routines

Argument Routines

ParseArgs()

DisposeArgs()

1.26 ParseArgs()

NAME

ParseArgs - easier interface to ReadArgs()

SYNOPSIS

ParseArgs(template, args)
 A0 A1

FuncArgs *ParseArgs(char *, char *);

FUNCTION

This routine makes it much more straightforward to use ReadArgs() to parse an argument string. Using ReadArgs to parse a string requires you to allocate and initialise a RArgs structure and argument array structure, and also requires the argument string to have a newline character. This function automates this process for you.

INPUTS

template - pointer to ReadArgs template string
 args - pointer to argument string (need not have a newline)

RESULT

If successful, this function returns a FuncArgs structure. This structure has several fields, but the useful ones are :

FA_Arguments

This is the argument array you should use. It is an array of long *, each member of which points to the argument result for the corresponding template entry. If you need to modify any of the values in this array you can, as it is just a copy of the real array.

FA_Count

This contains the number of arguments in the template. Opus counts the arguments in the template and initialises the argument array accordingly.

The strings supplied to this function are not needed once the function has returned.

NOTES

You should use this routine when parsing arguments supplied to your Opus modules.

SEE ALSO

DisposeArgs()
 , dos.library/ReadArgs()

1.27 DisposeArgs()

NAME

DisposeArgs - free a FuncArgs structure

SYNOPSIS

DisposeArgs(funcargs)
 A0

```
void DisposeArgs(FuncArgs *);
```

FUNCTION

This function frees the FuncArgs structure returned by ParseArgs()

.

INPUTS

funcargs - pointer to FuncArgs structure to free

RESULT

The structure is free. Once you have freed it, none of the arguments

remain valid, so you should make local copies of anything you need to refer to.

SEE ALSO

```
ParseArgs()  
, dos.library/ReadArgs()
```

1.28 BOOPSI_gadgets

BOOPSI Gadgets

CLASSES

dopusbuttonclass

dopuscheckgclass

dopusframeclass

dopusicclass

dopuslistviewgclass

dopuspalettegclass

dopusstrgclass

dopusviewgclass

PURPOSE

The `dopus5.library` makes several BOOPSI gadgets available globally. These gadgets can be accessed globally without even opening the `dopus5.library`, although it is a good idea to open it to make sure the library is present in the system.

The gadgets are all sub-classes of standard BOOPSI gadgets, and so take all the standard tags (`GA_Left`, `GA_Top`, etc..). Often they are based heavily on GadTools gadgets and will support equivalent GadTools tags. They also have their own set of tags, which is described below.

1.29 dopusbuttonclass

`dopusbuttonclass`

The `dopusbuttonclass` provides a standard pushbutton gadget. It is similar to a standard `buttonclass` gadget, but provides some additional functionality. This is via the following tags:

GTCustom_TextAttr (struct TextAttr *) (I) - used to specify a font for the gadget label. (default is the window font).

GTCustom_ThinBorders (BOOL) (I) - if set to TRUE, the gadget will be rendered with single-pixel borders (default FALSE).

GTCustom_Borderless (BOOL) (I) - if set to TRUE, the gadget will be rendered with no border (default FALSE).

GTCustom_Bold (BOOL) (I) - is set to TRUE, the gadget label will be rendered in bold (default FALSE).

GTCustom_Style (ULONG) (I) - use this tag to control the text style of the gadget label. Valid flags are FSF_BOLD and FSF_ITALIC (default FSF_NORMAL).

GTCustom_NoGhost (BOOL) (I) - if set to TRUE, the gadget imagery will not 'ghost' when the gadget is disabled (default FALSE).

GTCustom_TextPlacement (WORD) (I) - Lets you select the position of the label relative to the gadget. Valid values are:

```
PLACETEXT_IN (default)
PLACETEXT_LEFT
PLACETEXT_RIGHT
PLACETEXT_ABOVE
```

1.30 dopuscheckgclass

dopuscheckgclass

The dopuscheckgclass provides a replacement for GadTools checkbox gadgets. As a BOOPSI class, it allows you to have a checkbox without using GadTools. This class uses the same basic code as the dopusbuttonclass, and as such supports the same tags. The class also supports the GTCB_Checked flag (defined in libraries/gadtools.h) to set or get the current state of the gadget.

1.31 dopusframeclass

dopusframeclass

The dopusframeclass is a BOOPSI class for a frame gadget. A frame gadget does not respond to user input; its only purpose is to draw a frame (usually around some other gadgets). This class uses the same basic code as the dopusbuttonclass, and as such supports the same tags. The class also supports the GTCustom_FrameFlags tag, to specify flags for the frame. Currently, the only defined flag is AREAFLAG_RECESSED, which causes the frame to be drawn as recessed.

1.32 dopusiclass

dopusiclass

This class allows you to access several predefined images. The image you receive is controlled by the following tags:

DIA_Type

This sets the image type. Current valid types are:

IM_ARROW_UP - an up arrow
 IM_ARROW_DOWN - a down arrow
 IM_CHECK - a check mark
 IM_DRAWER - a "folder" image
 IM_BBOX - a filled box with a border
 IM_BORDER_BOX - a filled box
 IM_ICONIFY - an iconify gadget image
 IM_LOCK - a lock gadget image

DIA_FrontPen

This sets the front pen for the image. Currently, only the IM_CHECK image supports this tag.

This class is a sub-class of imageclass, and so supports the standard IM_Width, IM_Height, etc, tags. Images are scaled to the supplied sizes.

1.33 dopuslistviewgclass

dopuslistviewgclass

This boopsi gadget is a replacement for the gadtools LISTVIEW_KIND gadgets. It has been designed to "drop-in" as easily as possible, and uses many of the same tags as the gadtools equivalent. It is however much more flexible than the gadtools gadget.

The gadget duplicates most of the tags provided by gadtools' listview gadget. It also offers some powerful additions not available under gadtools. These include :

- o Current selection indicated by highlight bar, checkmark or text colour
- o Multiple-selection of items with checkmarks
- o Items can be rendered in different colours
- o Simple text formatting in the lister
- o Scroller can be optionally removed
- o Supports drag notification
- o Automatic double-click notification
- o Supports resizing via OM_SET

It also does not suffer from the gadtools problem of resizing itself to an

integral multiple of the item height (ie, the size you specify is the size you get). It is controlled by the following tags:

- DLV_Top (WORD) (ISG) - Top item visible in the listview. This value will be made reasonable if out-of-range (defaults to 0).
- DLV_MakeVisible (WORD) (IS) - Number of an item that should be forced within the visible area of the listview by doing minimal scrolling. This tag overrides DLV_Top.
- DLV_Labels (struct List * or Att_List *) (ISG) - List of nodes whose ln_Name fields are to be displayed in the listview. Calling SetGadgetAttrs() and specifying 0 will remove the current list. Specifying ~0 will remove the list but will not disturb the display, allowing you to make changes to the contents and selection status.
- DLV_ReadOnly (BOOL) (I) - If TRUE, then listview is read-only (defaults to FALSE).
- DLV_ScrollWidth (UWORD) (I) - Width of scroll bar for listview. Must be greater than zero (defaults to 16).
- DLV_ShowSelected (void) (I) - Specify this tag to have the currently selected item displayed with a highlight bar (or another method). Note that this tag does not support the automatic copying to a string gadget that gadtools does. You should specify ti_Data as 0 for future compatibility.
- DLV_Selected (UWORD) (ISG) - Ordinal number of currently selected item, or ~0 to have no current selection (defaults to ~0).
- DLV_TextAttr (struct TextAttr *) (I) - Allows you to specify a font to use in the lister. Must have previously been opened.
- DLV_MultiSelect (BOOL) (I) - If TRUE, the listview allows multiple-selection of items (see below for details).
- DLV_Check (BOOL) (I) - If TRUE, and DLV_ShowSelected is TRUE, the current selection will be indicated with a checkmark. Note that this tag has no meaning in conjunction with DLV_MultiSelect.
- DLV_ShowChecks (ULONG) (I) - If set to something other than zero, checkmarks will be shown for selected items (see below for details), but the user will not be able to alter their state.
- If set to 1, selected items will be rendered in the highlight pen colour. If set to 2, they will be rendered in the normal text colour.
- DLV_Highlight (BOOL) (I) - If TRUE, and DLV_ShowSelected is TRUE, the current selection will be displayed in a different colour.
- DLV_NoScroller (BOOL) (I) - If TRUE, the lister will not have a scroller attached. The gadget will still support scrolling by "dragging" the selection highlight.
- DLV_TopJustify (BOOL) (I) - If TRUE, items displayed in the lister will
-

be aligned to the top of the gadget, rather than being centred vertically.

DLV_Flags (ULONG) (I) - Allows you to specify layout flags for the lister. Currently the only flags supported are :

PLACETEXT_ABOVE - display title above gadget (default)
PLACETEXT_LEFT - display title at top-left of gadget

DLV_RightJustify (BOOL) (I) - If TRUE, items displayed in the lister will be aligned to the right of the gadget, rather than to the left.

DLV_ShowFileNames (BOOL) (I) - If TRUE, items in the lister are taken to be pathnames to files, and only the filename component (ie the result of a FilePart() call) is displayed. This allows you to keep the full pathname in ln_Name but only display the filename.

DLV_DragNotify (ULONG) (I) - If this is set to something other than zero, the gadget will notify you when the user tries to drag an item out of it. See the section on DragNotify below.

DLV_ScrollUp (void) (S) - Use this tag with SetGadgetAttrs() to make the lister scroll up one line.

DLV_ScrollDown (void) (S) - Use this tag with SetGadgetAttrs() to make the lister scroll down one line.

DLV_SelectPrevious (void) (S) - Use this tag with SetGadgetAttrs() to make the previous entry become selected.

DLV_SelectNext (void) (S) - Use this tag with SetGadgetAttrs() to make the next entry become selected.

DLV_Lines (void) (G) - returns number of visible lines displayed in lister.

DLV_Object (void) (G) - returns the address of the Object * structure.

DLV_GetLine (void) (G) - this allows you to get the line number in the lister from window-relative mouse coordinates. StoragePtr should be initialised to the mouse coordinates ((x<<16)|y).

DLV_DrawLine (void) (G) - this allows you to render a line of the listview into your own RastPort. See the section on DragNotify below for more information.

The gadget is a subclass of gadgetclass and as such supports the standard gadgetclass tags (including GA_Disabled). The title of the gadget can be specified with GA_Text (GA_IntuiText and GA_LabelImage are not supported).

MULTIPLE SELECTION

The dopuslistviewgclass gadget supports multiple-selection of items. This feature is enabled by passing {DLV_MultiSelect,TRUE} on creation. The ln_Type field of each of the node structures is used to

indicate whether an item is selected or not. For convenience, this field has been renamed `lve_Flags`.

To see whether an item is selected, test the `LVEF_SELECTED` flag in the `lve_Flags` field. Similarly, you can set an item's selection status by changing the value of this flag.

CUSTOM PEN COLOURS

You can specify the individual pen colours of each of the items in the list. The `ln_Pri` field of each of the node structures is used for this purpose. For convenience, this field has been renamed `lve_Pen`.

To specify that an item is to be rendered in other than the default pen colour, set `lve_Pen` to the appropriate value and set the `LVEF_USE_PEN` flag in the `lve_Flags` field.

TEXT FORMATTING

The gadget supports simple text-formatting for item display. This allows you to have columns and right-justified text in the lister.

If the text for an entry (`ln_Name`) contains a `\t` (tab character), the text following that character will be right-justified in the lister.

You can specify column positions using the `\a` (alert) character. The character immediately following the `\a` provides the position for the start of the next column. This is specified in character spaces. You should be aware that characters in proportional fonts are often wider than the nominal width of the font.

For example, if the following items were supplied to the gadget :

```
Bloggs\a\xa Fred\a\x1a 1-Sep-65\tPaid
Hall\a\xa Jane\a\x1a 9-Aug-68\tNot paid
Hubbard\a\xa Bill\a\x1a 7-Mar-18\tPaid
```

The display you would see would be something like this :

```
Bloggs      Fred      1-Sep-65    Paid
Hall        Jane      9-Aug-68    Not paid
Hubbard     Bill      7-Mar-18    Paid
```

DRAG NOTIFICATION

To enable drag notification, pass `{DLV_DragNotify,1}` on creation. You will then be sent an extended taglist via the `IDCMP_IDCMPUPDATE` message when the user attempts to drag an item out of the list.

If you pass `{DLV_DragNotify,2}` the user will only be able to drag out of the list sideways; dragging up or down will scroll the list as usual.

The tags you are sent on an attempted drag are as follows :

Tag	Data
GA_ID	gadget ID
GA_Left	
GA_Top	window-relative item coordinates
GA_Width	
GA_Height	size of the item as displayed
GA_RelRight	
GA_RelBottom	offset mouse position in item
DLV_Top	top item number
DLV_DragNotify	ordinal number of item dragged

To see if an IDCMP_IDCMPUDPATE message is from a drag, just test for the presence of the DLV_DragNotify tag in the taglist.

Once you get a drag notification, the actual dragging of the item is your responsibility. The easiest way is using the drag routines provided by the dopus5.library. Create a DragInfo large enough for the item (GA_Width and GA_Height in the taglist). There are two ways to get the image for the bitmap.

The first way is to use the GA_Left and GA_Top coordinates in the taglist and just ClipBlit() from your window into the drag rastport. This is the easiest way, but will also copy the checkmark if there is one, and you may not want that.

The second way is to use the DLV_DrawLine tag with the GetAttr() call, and have the listview render the item into your bitmap for you.

To do this, you need to initialise a ListViewDraw structure :

```
lvdraw.rp      RastPort to render into
lvdraw.drawinfo DrawInfo for the screen
lvdraw.node    List node to render
lvdraw.line    Set to 0
lvdraw.box.Left Set to 0
lvdraw.box.Top Set to 0
lvdraw.box.Width Width of BitMap
lvdraw.box.Height Height of BitMap
```

Then you pass the address of the ListViewDraw structure as the StoragePtr for the GetAttr call. Eg,

```
ULONG *ptr=(ULONG)&lvdraw;
Object *obj=GetTagData(DLV_Object,0,tags);

GetAttr(DLV_DrawLine,obj,&ptr);
```

The GA_RelRight and GA_RelBottom tags are used to indicate where in the item the user clicked. When you display the drag image on the screen, you should offset its position by these values.

DOUBLE-CLICK NOTIFICATION

If you get an IDCMP_IDCMPUPDATE message from the gadget, and the DLV_DragNotify tag is not set, it is a normal selection message. An additional tag is sent in this situation; DLV_DoubleClick. The ti_Data field is a boolean indicating whether the selection is a double-click or a normal single click.

The tags now sent for this message are :

Tag	Data
GA_ID	Gadget ID
DLV_Selected	Ordinal number of selection
DLV_DoubleClick	BOOL

RESIZING

To resize the gadget, pass the new coordinates via GA_Left, GA_Top, GA_Width and GA_Height in a SetGadgetAttrs() call. You will then need to refresh the display yourself, usually by clearing the window and calling RefreshGLList(). You may also need to call RefreshWindowFrame(), if the window has been resized smaller, as the gadget may have overwritten the window border before it was resized.

1.34 dopuspalettegclass

dopuspalettegclass

The dopuspalettegclass provides a replacement for GadTools PALETTE_KIND gadgets. As a BOOPSI class, it allows you to have a palette gadget without using GadTools. This class supports the following tags:

GTCustom_TextAttr (struct TextAttr *) (I) - used to specify a font for the gadget label. (default is the window font).

GTCustom_ThinBorders (BOOL) (I) - if set to TRUE, the gadget will be rendered with single-pixel borders (default FALSE).

GTPA_Color (UBYTE) (ISG) - the currently selected colour of the palette. This number is a pen number, and not the ordinal colour number within the palette gadget itself (default 1).

GTPA_Depth (UWORD) (IS) - Number of bitplanes in the palette (default 1).

GTPA_ColorTable (UBYTE *) (IS) - Pointer to a table of pen numbers indicating which colours should be used and edited by the palette gadget. This array must contain as many entries as there are colours displayed in the palette gadget. The array provided with this tag must remain valid for the life of the gadget, or until a new table is provided. (default is NULL, which causes a 1-to-1 mapping of pen numbers).

GTPA_NumColors (UWORD) (IS) - Number of colours to display in the palette gadget. This overrides GTPA_Depth and allows numbers which aren't powers of 2. (defaults to 2)

DPG_Pen (UWORD) (ISG) - the currently selected colour of the palette. This is similar to GTPA_Color but refers to the ordinal colour number and not the pen number itself.

DPG_SelectNext (void) (S) - use this tag with SetGadgetAttrs() to cause the next colour in the gadget to be selected.

DPG_SelectPrevious (void) (S) - use this tag with SetGadgetAttrs() to cause the previous colour in the gadget to be selected.

1.35 dopusstrgclass

dopusstrgclass

This dopusstrgclass provides a replacement for GadTools STRING_KIND gadgets. It is basically a standard string gadget with an automatic border, but also supports additional features. This class is based on the dopusbuttongclass, and as such supports all the tags of that class. It is also a subclass of strgclass and supports the standard string gadget tags of that class (with some important changes, listed below). The control tags supported by this class are as follows:

STRINGA_Buffer (char *) (I) - Specify the main buffer for the gadget. If this is not supplied, a buffer will be allocated automatically (this does not suffer from the maximum 128 bytes limitation of the standard BOOPSI string gadget class).

STRINGA_UndoBuffer (char *) (I) - Specify the undo buffer for the gadget. Again, one will be allocated automatically if you do not supply one.

STRINGA_WorkBuffer (char *) (I) - Specify the work buffer for the gadget. This will also be automatically allocated if you do not supply it.

STRINGA_MaxChars (long) (I) - Specify the maximum length of the string editable by this gadget. If buffers are allocated automatically, they will be this size. GTST_MaxChars and GTIN_MaxChars are also synonyms for this tag. (defaults to 40).

STRINGA_Font (struct TextFont *) (I) - Specify the font to use for this gadget.

GTCustom_ChangeSigTask (struct Task *) (I) - Specify a task that is to be signalled whenever the contents of this gadget change. (defaults to NULL).

GTCustom_ChangeSigBit (BYTE) (I) - Specify the signal bit that is used to signal a task whenever the contents of this gadget change. (defaults to 0).

STRINGA_TextVal (char *) (IS) - Set the contents of the string gadget. The supplied string is copied to the buffer. GTST_String, GTTX_Text, GTIN_Number and GTNM_Number are valid synonyms for this tag.

To use the `dopus5.library` edit hook with a string gadget, you should call

```
GetEditHook()  
and pass the results with the STRINGA_EditHook tag.
```

1.36 dopusviewgclass

`dopusviewgclass`

This class provides a simple view gadget, similar to GadTools `TEXT_KIND` and `NUMBER_KIND` gadgets. It is a subclass of `dopusbuttonclass`, and so supports all the tags of that class. To set the contents of the view gadget, use the `GTTX_Text` or `GTNM_Number` tags (a view gadget can be used to display either text or a number interchangeably).

1.37 BufIO_Routines

Buffered I/O Routines

`CloseBuf()`

`ExamineBuf()`

`FHFromBuf()`

`FlushBuf()`

`OpenBuf()`

`ReadBuf()`

`SeekBuf()`

`WriteBuf()`

1.38 CloseBuf()

NAME

`CloseBuf` - close a buffered file

SYNOPSIS

`CloseBuf(file)`

A0

```
void CloseBuf(APTR);
```

FUNCTION
Closes a file opened with
 OpenBuf()
 .

INPUTS
file - file to close

RESULT
Any write data in the buffer is written to disk and the
file is closed.

SEE ALSO
 OpenBuf()

1.39 ExamineBuf()

NAME
ExamineBuf - Examine an open file

SYNOPSIS
ExamineBuf(file, fib)
 A0 A1

long ExamineBuf(APTR, struct FileInfoBlock *);

FUNCTION
This function calls ExamineFH() on the underlying DOS file handle.

INPUTS
file - file to examine
fib - FileInfoBlock structure, must be longword aligned

RESULT
Returns DOSTRUE if successful. The FileInfoBlock will contain
information about the open file.

BUGS
If the file is open for writing, the file size reported by this
function may not be accurate.

SEE ALSO
 OpenBuf()
 , dos.library/ExamineFH()

1.40 FHFromBuf()

NAME

FHFromBuf - get DOS file handle

SYNOPSIS

FHFromBuf(file)

A0

BPTR FHFromBuf(APTR);

FUNCTION

This function returns the underlying DOS file handle for a buffered IO handle.

INPUTS

file - buffered IO file handle

RESULT

Returns the file handle.

SEE ALSO

OpenBuf()

1.41 FlushBuf()

NAME

FlushBuf - flush file buffer

SYNOPSIS

FlushBuf(file)

A0

void FlushBuf(APTR);

FUNCTION

This function flushes the buffer of a buffered IO file. If there is any write data in the buffer, it is written to disk.

INPUTS

file - file handle to flush

RESULT

The buffer is flushed.

NOTES

In practice, you never need to call this function.

SEE ALSO

OpenBuf()

,

WriteBuf()

,

ReadBuf()

1.42 OpenBuf()

NAME

OpenBuf - open a file for buffered I/O

SYNOPSIS

OpenBuf(name, mode, bufsize)
A0 D0 D1

APTR OpenBuf(char *, long, long);

FUNCTION

This function opens a file for use with the buffered I/O routines.

INPUTS

name - name of the file to open
mode - mode to use
bufsize - size of the buffer to use

RESULT

Returns a buffered file handle if successful, or NULL. This is not a standard DOS file handle, and can only be used with the other buffered IO functions.

SEE ALSO

CloseBuf()
, dos.library/Open()

1.43 ReadBuf()

NAME

ReadBuf - read data from a buffered file

SYNOPSIS

ReadBuf(file, buffer, size)
A0 A1 D0

long ReadBuf(APTR, char *, long);

FUNCTION

This function reads data from a buffered IO file.

INPUTS

file - buffered IO file handle
buffer - buffer to place data in
size - size to read

RESULT

This function returns the size of the data actually read, or -1 if an error occurred.

SEE ALSO

OpenBuf()
, dos.library/Read()

1.44 SeekBuf()

NAME

SeekBuf - seek within a buffered IO file

SYNOPSIS

SeekBuf(file, offset, mode)
A0 D0 D1

long SeekBuf(APTR, long, long);

FUNCTION

This function sets the read/write position for a buffered IO file. If the seek takes the position outside of the current buffer, the buffer will be flushed and re-read automatically.

INPUTS

file - file to seek within
offset - offset to seek
mode - type of seek (OFFSET_BEGINNING, OFFSET_CURRENT, OFFSET_END)

RESULT

Returns the previous file position.

SEE ALSO

OpenBuf()
, dos.library/Seek()

1.45 WriteBuf()

NAME

WriteBuf - write data to a buffered IO file

SYNOPSIS

WriteBuf(file, data, size)
A0 A1 D0

long WriteBuf(APTR, char *, long);

FUNCTION

Writes data to a file opened for buffered IO.

INPUTS

file - file handle
data - data to write
size - size to write (-1 works for a null-terminated string)

RESULT

Returns the number of bytes written, or -1 for an error.

SEE ALSO

OpenBuf()
, dos.library/Write()

1.46 Clipboard_Routines

Clipboard Routines

CloseClipboard()

OpenClipboard()

ReadClipString()

WriteClipString()

1.47 CloseClipboard()

NAME

CloseClipboard - close a clipboard handle

SYNOPSIS

CloseClipboard(handle)
A0

```
void CloseClipboard(APTR);
```

FUNCTION

Closes a handle to the clipboard opened with
OpenClipboard()
.

INPUTS

handle - clipboard handle

RESULT

The clipboard unit is closed.

SEE ALSO

OpenClipboard()

1.48 OpenClipboard()

NAME

OpenClipboard - open clipboard for easy access

SYNOPSIS

```
OpenClipboard(unit)
    D0
```

```
APTR OpenClipboard(ULONG);
```

FUNCTION

This function opens a specified unit of the clipboard.device. Used with the other clipboard functions, it provides an easy method to manipulate text strings with the clipboard.

INPUTS

unit - unit number to open (usually 0)

RESULT

Returns clipboard handle.

SEE ALSO

CloseClipboard()

1.49 ReadClipString()

NAME

ReadClipString - read a text string from the clipboard

SYNOPSIS

```
ReadClipString(handle, buffer, size)
    A0      A1      D0
```

```
long ReadClipString(APTR, char *, long);
```

FUNCTION

This function reads a string of text from the clipboard handle.

INPUTS

handle - clipboard handle
buffer - buffer to store string
size - size of buffer

RESULT

Returns the length of the string. If there was no valid FTXT string in the clipboard, it returns 0.

SEE ALSO

```
OpenClipBoard()
,
WriteClipString()
```

1.50 WriteClipString()

NAME

WriteClipString - write a text string to the clipboard

SYNOPSIS

```
WriteClipString(handle, buffer, size)
    A0      A1      D0
```

```
BOOL WriteClipString(APTR, char *, long);
```

FUNCTION

This function writes a string of text to the clipboard handle.

INPUTS

handle - clipboard handle
buffer - buffer containing string
size - length of string

RESULT

Returns TRUE if it succeeded, FALSE otherwise. The string is stored in standard FTXT format, readable by

```
ReadClipString()
and most
```

other programs that access the clipboard.

SEE ALSO

```
OpenClipBoard()
,
ReadClipString()
```

1.51 DiskIO_Routines

Disk I/O Routines

```
OpenDisk()
```

```
CloseDisk()
```

1.52 OpenDisk()

NAME

OpenDisk - open a disk for direct I/O

SYNOPSIS

```
OpenDisk(disk, port)
    A0    A1
```

```
DiskHandle *OpenDisk(char *, struct MsgPort *);
```

FUNCTION

This routine makes it easy to access the underlying device for direct I/O. It allows you to open any filesystem (that supports direct I/O) with just the device name.

INPUTS

disk - name of disk to open, eg DF0:, HD1:
port - message port to use, or NULL

RESULT

If this function succeeds, it returns a DiskHandle structure, which contains all the information you need to access the device directly. The structure fields are :

dh_Port

If you did not supply a message port to use, one is created automatically and its address is stored here. Usually you will want a port created for you, but if you are working with multiple devices at once you might want them all to share the same message port.

dh_IO

This is a pointer to an IOExtTD structure, which you can use to perform I/O on the device.

dh_Startup

A pointer to the FileSysStartupMsg of the device.

dh_Geo

A pointer to the DosEnvec structure of the device.

dh_Device

Full device name (without a colon)

dh_Info/dh_Result

If dh_Result is TRUE, dh_Info is valid, and contains current information about the disk.

dh_Root/dh_BlockSize

These give the block number of the disk's root block, and the block size.

SEE ALSO

CloseDisk()

, trackdisk.doc

1.53 CloseDisk()

NAME

CloseDisk - close a DiskHandle structure

SYNOPSIS

CloseDisk(handle)
A0

```
void CloseDisk(DiskHandle *);
```

FUNCTION

This function cleans up and closes a DiskHandle structure opened with the

OpenDisk()
routine.

INPUTS

handle - DiskHandle to close

SEE ALSO

OpenDisk()

1.54 DOS_Routines

DOS Routines

DateFromStrings()

DeviceFromHandler()

DeviceFromLock()

DevNameFromLock()

FreeDosPathList()

GetDosPathList()

GetFileVersion()

LaunchCLI()

LaunchWB()

ParseDateStrings()

```
SearchFile()
```

```
SetEnv()
```

1.55 DateFromStrings()

NAME

DateFromStrings - convert date and time strings to a datestamp

SYNOPSIS

```
DateFromStrings(date, time, ds)
    A0    A1    A2
```

```
BOOL DateFromStrings(char *, char *, struct DateStamp *);
```

FUNCTION

This routine takes a date string and a time string and converts them to a DOS DateStamp. The DOS StrToDate() routine is used to perform this conversion, so it is sensitive to the current locale. If the time string contains an 'a' or a 'p' to signify am or pm, it is automatically converted to 24 hour time for the DOS call.

INPUTS

date - date string to convert
time - time string to convert
ds - DateStamp to store result

RESULT

Returns TRUE if successful.

SEE ALSO

```
ParseDateStrings()
, dos.library/StrToDate()
```

1.56 DeviceFromHandler()

NAME

DeviceFromHandler - returns device name from handler

SYNOPSIS

```
DeviceFromHandler(handler, buffer)
    A0    A1
```

```
struct DosList *DeviceFromHandler(struct MsgPort *, char *);
```

FUNCTION

This function takes a pointer to a filesystem's handler (message port) and returns the associated device name.

INPUTS

handler - pointer to handler message port
 buffer - buffer to store device name (must be >=34 bytes)

RESULT

If the port supplied is a valid filesystem handler, the name of the device is stored in the supplied buffer, and a pointer to the DosList entry for that device is returned.

SEE ALSO

```
DeviceFromLock()
,
DevNameFromLock()
```

1.57 DeviceFromLock()

NAME

DeviceFromLock - returns device name from a filelock

SYNOPSIS

```
DeviceFromLock(lock, buffer)
    A0      A1
```

```
struct DosList *DeviceFromLock(BPTR, char *);
```

FUNCTION

This function takes a filelock and returns the name of the device that lock resides on.

INPUTS

lock - pointer to lock
 buffer - buffer to store device name (must be >=34 bytes)

RESULT

The name of the device is stored in the supplied buffer, and a pointer to the DosList entry for that device is returned.

SEE ALSO

```
DeviceFromHandler()
,
DevNameFromLock()
```

1.58 DevNameFromLock()

NAME

DevNameFromLock - return the full pathname of a file

SYNOPSIS

```
DevNameFromLock(lock, buffer, size)
    D1      D2      D3
```

```
BOOL DevNameFromLock(BPTR, char *, long);
```

FUNCTION

Returns a fully qualified path for the lock. The only difference between this function and the equivalent DOS library routine is that the device name of the disk is returned, rather than the volume name.

For example, if the NameFromLock() routine returned :

```
Workbench:S/startup-sequence
```

The DevNameFromLock() routine would return :

```
DH0:S/startup-sequence
```

INPUTS

lock - filelock to obtain the path for
buffer - buffer to store path
size - size of buffer

RESULT

This function returns TRUE if it succeeds.

SEE ALSO

```
DeviceFromLock()
, dos.library/NameFromLock()
```

1.59 FreeDosPathList()

NAME

FreeDosPathList - free a DOS path list

SYNOPSIS

```
FreeDosPathList(list)
    A0
```

```
void FreeDosPathList(BPTR);
```

FUNCTION

This function frees a standard DOS path list, by unlocking each lock and FreeVec()ing each entry.

INPUTS

list - pointer to head of list

RESULT

The list is freed.

SEE ALSO

GetDosPathList()

1.60 GetDosPathList()

NAME

GetDosPathList - get a copy of a DOS path list

SYNOPSIS

GetDosPathList(list)
A0

BPTR GetDosPathList(BPTR);

FUNCTION

This routine has two uses. The first is to copy an existing DOS path list that you supply. The second is to attempt to find and copy the system path list.

INPUTS

list - path list to copy or NULL

RESULT

If you supply a path list, it will be copied and the address of the first entry of the new list will be returned.

If you pass NULL, this routine attempts to find a system path list to copy. The Amiga has no definitive path list, so the only way to obtain one is to copy it from another process. This routine looks for the following processes (in order) : Workbench, Initial CLI, Shell Process, New_WShell and Background CLI. If one of these processes is found and it has a valid path list, that list is copied and returned to you.

NOTES

If Workbench is not running, Opus creates a dummy task called 'Workbench', purely to provide a path list for programs that use this method.

SEE ALSO

FreeDosPathList()

1.61 GetFileVersion()

NAME

GetFileVersion - get a file's version information

SYNOPSIS

```
GetFileVersion(name, verptr, revptr, date, progress)
    A0      D0      D1      A1      A2
```

```
BOOL GetFileVersion(char *, short *, short *,
    struct DateStamp *, APTR);
```

FUNCTION

This routine examines the given file and returns the file's version number and revision, and creation date if available. It looks primarily for a \$VER string, but also understands the format of libraries, devices, etc, and can extract the version from the Romtag structure in the file. You can also supply a Progress handle if you want to use a progress indicator while looking for the version information.

INPUTS

name - full pathname of file to examine
 verptr - pointer to short to receive the version number
 revptr - pointer to short to receive the revision number
 date - pointer to DateStamp structure (NULL if no date needed)
 progress - pointer to progress indicator (or NULL)

RESULT

Returns TRUE if a valid version number was found (this does not necessarily mean that a date was found too).

1.62 LaunchCLI()

NAME

LaunchCLI - launch a program as a CLI process

SYNOPSIS

```
LaunchCLI(name, screen, curdir, input, output, wait)
    A0      A1      D0      D1      D2      D3
```

```
BOOL LaunchCLI(char *, struct Screen *, BPTR, BPTR, BPTR, short);
```

FUNCTION

This routine makes it easy to launch a program as a CLI process. The launched process will have a full path list and copy of local environment variables. You can have the process launched synchronously, which means this function would not return until the process quit. The stack size is fixed to 4096 bytes.

INPUTS

name - name of the program to launch, including any arguments
 screen - a screen for errors to appear on (or NULL for default)
 curdir - lock for current directory, or NULL for default
 input - file handle for standard input, or NULL
 output - file handle for standard output, or NULL
 wait - set to TRUE if you want to wait for the process to return

RESULT

Returns TRUE if the process was launched successfully. If the 'wait' parameter was set to TRUE, will not return until the child

process does.

NOTES

This function will search the current path list for your program if you do not specify the full path.

SEE ALSO

LaunchWB
, dos.library/SystemTagList

1.63 LaunchWB()

NAME

LaunchWB - launch a program as a Workbench process

SYNOPSIS

LaunchWB(name, screen, wait)
A0 A1 D0

BOOL LaunchWB(char *, struct Screen *, short);

FUNCTION

This routine makes it easy to launch a program as a Workbench process. Workbench processes expect to receive a startup message from the launching process, and ordinarily the launching process must wait until this message is replied to. Using this function relieves you of this - you can launch the process and then forget about it. The launched process will have a full path list and copy of local environment variables.

INPUTS

name - name of the program to launch, including any arguments
screen - a screen for errors to appear on (or NULL for default)
wait - set to TRUE if you want to wait for the process to return

RESULT

Returns TRUE if the process was launched successfully. If the 'wait' parameter was set to TRUE, will not return until the child process does. Otherwise, it will return immediately and you do not need to wait for a reply to the startup message.

NOTES

This function will search the current path list for your program if you do not specify the full path.

SEE ALSO

LaunchCLI
, dos.library/SystemTagList

1.64 ParseDateStrings()

NAME

ParseDateStrings - parse a date/time string into separate buffers

SYNOPSIS

```
ParseDateStrings(string, date, time, rangeptr)
                A0      A1      A2      A3
```

```
char *ParseDateStrings(char *, char *, char *, long *);
```

FUNCTION

This function takes a date/time string (eg "8-12-95 10:34:18") and splits the date and time elements into separate buffers. It also supports the use of the '>' character to indicate ranges. For example, "10-jan-94 > 15-jun-95" would indicate any date between those dates.

INPUTS

string - combined date/time string. This routine is smart enough to handle it if the time comes before the date, or vice versa, and it also deals reasonably well with different types of date inputs.
date - buffer to receive the date component (>=22 bytes)
time - buffer to receive the time component (>=22 bytes)
rangeptr - long pointer to receive the range code (or NULL)

RESULT

The return from this function is a pointer to the end of the parsed part of the input string. If the range returns RANGE_BETWEEN (to signify a range between two dates), you will need to call ParseDateStrings() again on the remainder of the string to get the next date and time.

SEE ALSO

DateFromStrings()

1.65 SearchFile()

NAME

SearchFile - search a file or buffer for a text string

SYNOPSIS

```
SearchFile(file, text, flags, buffer, bufsize)
                A0      A1      D0      A2      D1
```

```
long SearchFile(APTR, UBYTE *, ULONG, UBYTE *, ULONG);
```

FUNCTION

This routine searches a file, either on disk or in memory, for a specified text string. It supports hex or decimal ascii values,

and limited wildcard searching. To search for a hex string, the supplied search string should begin with a \$ and then consist of two-character hex codes. When searching for plain text, a decimal ascii value can be specified with a \ character (eg \127). A literal \ is given as \\. A question mark (?) is used as a single wildcard character in both hex and text searches.

INPUTS

file - buffered IO file handle
text - text string to search for
flags - Combination of the following flags :

SEARCH_NOCASE - not case sensitive
SEARCH_WILDCARD - support ? as a wildcard character
SEARCH_ONLYWORDS - only match whole words

buffer - memory buffer to search if no file specified
bufsize - size of memory buffer

RESULT

If the supplied string is found, the offset within the file/buffer of the first instance is returned. If no match is found or an error occurs, -1 is returned.

1.66 SetEnv()

NAME

SetEnv - set a global environment variable

SYNOPSIS

SetEnv(name, string, permanent)

A0 A1 D0

```
void SetEnv(char *, char *, BOOL);
```

FUNCTION

This routine sets the named environment variable to the supplied string value, and optionally saves it permanently.

INPUTS

name - name of the variable to set
string - text string to set the variable to (must be null-terminated)
permanent - set to TRUE if you want the variable saved

RESULT

The environment variable will be created if it does not exist. Any sub-directories that are needed will also be created. For example, if you set the variable "foo/bar/baz", the directories "env:foo" and "env:foo/bar" would be automatically created if they did not exist. If you set the 'permanent' flag to TRUE, the variable will also be created in the ENVARC: directory.

SEE ALSO

dos.library/GetVar

1.67 Drag_Routines

Drag Routines

FreeDragInfo()

GetDragImage()

GetDragInfo()

GetDragMask()

HideDragImage()

ShowDragImage()

StampDragImage()

PURPOSE

No, this isn't a new form of software-based cabaret act. The drag routines in the dopus5.library make it easy for you to implement your own drag and drop system.

The DragInfo structure is the key of this system. Calling the

GetDragInfo()

function will create one of these structures, and you use it in all subsequent calls.

The important fields of the DragInfo structure are :

flags You can set flags to modify the behaviour of dragged images.

DRAGF_OPAQUE indicates that the drag image should be opaque; that is, colour 0 does not allow the background to show through.

DRAGF_NO_LOCK indicates that the drag routines should not lock the screen layers themselves.

DRAGF_TRANSPARENT indicates that the drag image should be transparent. Used in conjunction with DRAGF_OPAQUE, it allows you to create irregular shaped images.

drag_rp This is a RastPort that you can use to draw into the drag image.

The other fields of the DragInfo structure can be used by you, but normally they should be left alone.

The usual process of dragging an image is :

1.
 - GetDragInfo()
 2. Either render into di->drag_rp or call GetDragImage()
 3. Call GetDragMask() if you rendered directly
4. If you set DRAGF_NO_LOCK, LockLayers()
5. Multiple calls to ShowDragImage() to make the image visible and move it around (in response to mouse movements)
6.
 - FreeDragInfo() to remove the image
7. If DRAGF_NO_LOCK was set, UnlockLayers()

The Amiga OS has a bug which can cause a deadlock if another task attempts to call LockLayers() while you have them locked. If you are dragging over the entire screen rather than an individual window, you will need to take additional steps to prevent this deadlock.

You need to set up a timer event, roughly every half second or so (the dopus5.library timer routines are ideal for this purpose). You also need to have the IDCMP_INTUITICKS flag set for your window. You then must keep a count of the number of IDCMP_INTUITICKS messages received. Every time your periodic timer event comes around, you must examine this count to see if it has changed. As INTUITICKS are sent roughly every 10th of a second, one or more should have been received between each of your timer events. If no INTUITICKS were received, it's a fair bet that Intuition has deadlocked itself, and you should immediately call UnlockLayers() (or HideDragImage()) to unfreeze the system.

1.68 FreeDragInfo()

NAME

FreeDragInfo - frees a DragInfo structure

SYNOPSIS

```
FreeDragInfo(drag)
    A0
```

```
void FreeDragInfo(DragInfo *);
```

FUNCTION

This function removes a drag image from the display if it is still visible, and frees the DragInfo structure.

INPUTS

drag - structure to free

SEE ALSO

GetDragInfo()

1.69 GetDragImage()

NAME

GetDragImage - pick up on-screen imagery to drag

SYNOPSIS

GetDragImage(drag, x, y)

A0 D0 D1

```
void GetDragImage(DragInfo *, long, long);
```

FUNCTION

This routine copies on-screen image data into the rastport of the specified DragInfo structure. If the drag image is visible when this routine is called, it is cleared before the data is copied.

INPUTS

drag - DragInfo structure

x - x-position on screen

y - y-position on screen

RESULT

The image data is copied from the Bitmap of the Window that was specified when the drag image was created. This routine calls

GetDragMask()
automatically.

SEE ALSO

GetDragInfo()

1.70 GetDragInfo()

NAME

GetDragInfo - create a DragInfo structure

SYNOPSIS

GetDragInfo(window, rastport, width, height, need_gels)

A0 A1 D0 D1 D2

```
DragInfo *GetDragInfo(struct Window *, struct RastPort *,
    long, long, long);
```

FUNCTION

Creates a DragInfo structure that is used to implement drag and drop. Drags are inherently attached to a particular RastPort (usually either

a screen's or a window's). The drag system is implemented using BOBs, which require a GelsInfo structure to be attached to the destination RastPort. This routine can do this for you if you desire.

INPUTS

window - Window to attach BOB to (or NULL if rastport is supplied)
rastport - RastPort to attach BOB to (if not a window)
width - width of drag image
height - height of drag image
need_gels - set to TRUE if you would like this routine to allocate and initialise a GelsInfo automatically

RESULT

If successful, a DragInfo structure is returned. Nothing is displayed on-screen; you must create the image and display it using the other library calls. Once you have the DragInfo structure, you can initialise the 'flags' field as described in the introduction.

SEE ALSO

FreeDragInfo()

1.71 GetDragMask()

NAME

GetDragMask - build mask for drag image

SYNOPSIS

GetDragMask(drag)
A0

```
void GetDragMask(DragInfo *);
```

FUNCTION

Once you have created the image you want to drag, you must call this function. This builds the shadow mask used to drag the image, and is necessary for the image to be displayed correctly.

INPUTS

drag - DragInfo structure to build mask for

SEE ALSO

GetDragInfo()

1.72 HideDragImage()

NAME

HideDragImage - remove a drag image from the display

SYNOPSIS

```
HideDragImage(drag)
    A0
```

```
void HideDragImage(DragInfo *);
```

FUNCTION

This routine removes a visible drag image from the display.

INPUTS

drag - DragInfo structure

SEE ALSO

ShowDragImage()

1.73 ShowDragImage()

NAME

ShowDragImage - display a drag image

SYNOPSIS

```
ShowDragImage(drag, x, y)
    A0 D0 D1
```

```
void ShowDragImage(DragInfo *, long, long);
```

FUNCTION

This routine displays a drag image at a given location. The image is displayed in the RastPort that was supplied to the

```
GetDragInfo()
call.
```

If the image was not already displayed, it is added to the display. If it was, it is removed from its current position and redisplayed in the new location. This is the main call used to move an image around the screen.

INPUTS

drag - DragInfo structure to display
x - x position in rastport
y - y position in rastport

SEE ALSO

```
GetDragInfo()
,
HideDragImage()
```

1.74 StampDragImage()

NAME

StampDragImage - stamp drag image onto the screen

SYNOPSIS

```
StampDragImage(drag, x, y)
```

```
A0 D0 D1
```

```
void StampDragImage(DragInfo *, long, long);
```

FUNCTION

This routine stamps the drag image onto the bitmap at the given location. Using this function would allow you to "paint" with the drag image.

INPUTS

drag - DragInfo structure

x - x position to stamp image at

y - y position to stamp image at

RESULT

The image is drawn into the RastPort that was supplied in the

```
GetDragInfo()
call.
```

SEE ALSO

```
GetDragInfo()
,
ShowDragImage()
```

1.75 Edit_Hook

```
Edit Hook
```

```
FreeEditHook()
```

```
GetEditHook()
```

```
GetSecureString()
```

PURPOSE

The dopus5.library provides an edit hook which you can attach to string gadgets, to take advantage of some additional features. The features provided by the edit hook include :

- Enhanced cursor movement using alt and shift
- History using up/down cursor
- Clipboard support (copy, cut and paste)
- Optional filtering of path characters
- Secure option where string is not displayed
- Return automatically activates the next string gadget

- Key press notification

To use the edit hook in your gadgets, you must obtain a Hook structure with the

```

    GetEditHook()
    function. Once you have finished with it, you
must free this with
    FreeEditHook()
    .

```

1.76 FreeEditHook()

NAME

FreeEditHook - free a Hook created with
GetEditHook()

SYNOPSIS

```
FreeEditHook(hook)
    A0
```

```
void FreeEditHook(struct Hook *);
```

FUNCTION

This routine frees a Hook that you obtained via a call to

```

    GetEditHook()
    .

```

INPUTS

hook - hook to free

SEE ALSO

```
GetEditHook()
```

1.77 GetEditHook()

NAME

GetEditHook - get a Hook pointer to access the edit hook

SYNOPSIS

```
GetEditHook(type, flags, tags)
    D0    D1    A0
```

```
struct Hook *GetEditHook(ULONG, ULONG, struct TagItem *);
```

```
struct Hook *GetEditHookTags(ULONG, ULONG, Tag, ...);
```

FUNCTION

This routine returns a Hook structure that you can use to give the additional capabilities to your string gadgets.

INPUTS

type - type of Hook (only HOOKTYPE_STANDARD is valid so far)
flags - a combination of the following flags :

EDITF_NO_SELECT_NEXT - stops the return key from automatically activating the next (or previous, with shift) string gadget.

EDITF_PATH_FILTER - filters path characters out of the string (/ and :)

EDITF_SECURE - secure password field, doesn't display the characters that are typed. If this flag is specified, the gadget's StringInfo->MaxChars value must be twice what it would ordinarily be. The gadget's StringInfo->Buffer must be this size plus an additional two bytes, eg (max_len*2)+2.

tags - the following tags are supported by the edit hook :

EH_History - supplies a pointer to an Att_List structure which contains the history list for this gadget. Each entry in the list is the name of a node, with the most recent node at the end of the list. The Att_List should be created using Semaphore locking.

EH_ChangeSigTask - allows you to specify a task that is to be signalled whenever the contents of the string gadget are changed by the user. This allows you to have a display dynamically updated as the user types.

EH_ChangeSigBit - the signal bit used to signal your task. This is the bit number, not a mask.

RESULT

Returns a pointer to a Hook structure which you can use for your gadget. It is not a good idea to share hooks between gadgets (although unless you are using the EH_History facility it should not really be a problem).

SEE ALSO

```
FreeEditHook()  
,  
GetSecureString()  
,  
Att_NewList()
```

1.78 GetSecureString()

NAME

GetSecureString - retrieve the real string from a secure field

SYNOPSIS

```
GetSecureString(gadget)
    A0
```

```
char *GetSecureString(struct Gadget *);
```

FUNCTION

The secure feature of the edit hook is implemented using a buffer for the gadget that is twice as large plus 2 bytes as it would ordinarily be. This is because the first half of the buffer is filled with * characters as the user types. The real text is stored in the second half of the buffer.

INPUTS

gadget - secure string gadget

RESULT

Returns a pointer to the real text for the gadget. The text is properly null-terminated.

SEE ALSO

GetEditHook()

1.79 GUI_Routines

GUI Routines

ActivateStrGad()

AddScrollBars()

BOOPSIFree()

DisposeBitMap()

DrawBox()

DrawFieldBox()

FindBOOPSIGadget()

GetPalette32()

LoadPalette32()

NewBitMap()

ScreenInfo()

FindPubScreen()

SetBusyPointer()

1.80 ActivateStrGad()

NAME

ActivateStrGad - Activate a string gadget.

SYNOPSIS

```
ActivateStrGad(gadget, window)
    A0      A1
```

```
void ActivateStrGad(struct Gadget *, struct Window *);
```

FUNCTION

This function activates a string gadget in a window. The difference between this and the standard ActivateGadget() call is that the cursor will be positioned at the end of the string.

INPUTS

gadget - string gadget you wish to activate
window - window the gadget is in

SEE ALSO

intuition.library/ActivateGadget()

1.81 AddScrollBars()

NAME

AddScrollBars - Add BOOPSI scrollers to a window

SYNOPSIS

```
AddScrollBars(window, list, drawinfo, flags)
    A0      A1      A2      D0
```

```
struct Gadget *AddScrollBars(struct Window *, struct List *,
    struct DrawInfo *, short);
```

FUNCTION

This function adds BOOPSI scrollers (proportional gadget and two arrows) to a window. It can add scrollers either horizontally, vertically or both.

INPUTS

window - window to add scrollers to. The window must have a border for the side(s) you wish to add scrollers to. For example, if you add a horizontal scroller, the window must have a bottom border (ie the WFLG_SIZEEBOTTOM flag is set).

list - this must point to an initialise List structure. The gadgets created by this routine will be added to this List, which can later be freed by

```
BOOPSIFree()
```

```
.
```

drawinfo - pointer to the screen's DrawInfo structure

flags - the type of scrollers you want :

```
SCROLL_VERT      - vertical scroller
SCROLL_HORIZ    - horizontal scroller
SCROLL_NOIDCMP  - this flag signified that the scrollers
                  are only to return the normal
                  IDCMP_GADGETUP, IDCMP_GADGETDOWN and
                  IDCMP_MOUSEMOVE messages. If not
                  specified, the gadgets also generate
                  IDCMP_IDCMPUPDATE messages.
```

RESULT

This routine returns 0 if it fails. If it succeeds, it returns a pointer to the last gadget created. This pointer is not particularly useful; it just signifies success. To actually add the gadgets to the window, call AddGList() on the first gadget in the supplied List structure, and then RefreshGList().

The gadgets created by this function have pre-defined IDs. If you use this routine you should make sure that your own gadgets do not conflict with these IDs. See the "dopus/gui.h" file for the ID values.

SEE ALSO

```
FindBOOPSIGadget()
```

```
,
```

```
BOOPSIFree()
```

```
,
```

```
intuition.library/GetScreenDrawInfo(), intuition.library/AddGList(),
intuition.library/RefreshGList()
```

1.82 BOOPSIFree()

NAME

BOOPSIFree - free a list of BOOPSI gadgets

SYNOPSIS

```
BOOPSIFree(list)
```

```
    A0
```

```
void BOOPSIFree(struct List *);
```

FUNCTION

This routine iterates through a list of BOOPSI gadgets, calling DisposeObject on each one in turn. It then reinitialises the list.

INPUTS

list - List of gadgets to free

SEE ALSO

```
AddScrollBars()
, intuition.library/DisposeObject()
```

1.83 DisposeBitMap()

NAME

DisposeBitMap - free a bitmap created with
NewBitMap()

SYNOPSIS

```
DisposeBitMap(bm)
A0
```

```
void DisposeBitMap(struct BitMap *);
```

FUNCTION

This routine frees a bitmap and the associated bitplanes that was allocated with the

```
NewBitMap()
```

function. Under OS39 it simply passes the bitmap through to the graphics.library FreeBitMap() call. Under OS37 it frees the bitmap and bitplanes manually.

INPUTS

bm - BitMap to free

RESULT

The BitMap is freed.

SEE ALSO

```
NewBitMap()
, graphics.library/FreeBitMap()
```

1.84 DrawBox()

NAME

DrawBox - draw a 3D box

SYNOPSIS

```
DrawBox(rp, rect, info, recessed)
A0 A1 A2 D0
```

```
void DrawBox(struct RastPort *, struct Rectangle *,
struct DrawInfo *, BOOL);
```

FUNCTION

This routine draws a single-pixel 3D box, using the current pen

settings in the DrawInfo you supply.

INPUTS

rp - RastPort to draw into
 rect - Rectangle to draw
 info - Screen's DrawInfo
 recessed - set to TRUE if you want a recessed box

SEE ALSO

intuition.library/GetScreenDrawInfo()

1.85 DrawFieldBox()

NAME

DrawFieldBox - draw a 3D field box

SYNOPSIS

```
DrawFieldBox(rp, rect, info)
             A0  A1  A2
```

```
void DrawFieldBox(struct RastPort *, struct Rectangle *,
                  struct DrawInfo *);
```

FUNCTION

Draws a 3D field box (eg the path field in DOpus listers).

INPUTS

rp - RastPort to draw into
 rect - Rectangle to draw
 info - Screen's DrawInfo

SEE ALSO

intuition.library/GetScreenDrawInfo()

1.86 FindBOOPSIGadget()

NAME

FindBOOPSIGadget - find a gadget by ID in a BOOPSI list

SYNOPSIS

```
FindBOOPSIGadget(list, id)
                 A0  D0
```

```
struct Gadget *FindBOOPSIGadget(struct List *, USHORT);
```

FUNCTION

This routine iterates through the supplied list of gadgets looking for one with the supplied ID. If found, it returns it. This routine can be used to find a specific gadget in the List created by

```
AddScrollBars()
```

. eg,

```
vert_scroll=FindBOOPSIGadget (&list,GAD_VERT_SCROLLER);
```

INPUTS

list - List of BOOPSI gadgets

id - ID to look for

RESULT

If the gadget is found, it is returned, otherwise NULL.

SEE ALSO

AddScrollBars()

1.87 GetPalette32()

NAME

GetPalette32 - get a 32 bit palette from a ViewPort

SYNOPSIS

```
GetPalette32(vp, palette, count, first)
```

```
    A0    A1        D0    D1
```

```
void GetPalette32(struct ViewPort *, ULONG, USHORT, short);
```

FUNCTION

This routine copies the palette from the supplied ViewPort into the supplied buffer. Under OS39 it is identical in operation to the graphics.library GetRGB32() function. The advantage of using this function is that it also works under OS37. Under OS37, the 4 bit colour values are extended to full 32 bit.

INPUTS

vp - ViewPort to load colours from

palette - array of ULONGs to store palette (3 per pen)

count - number of pen values to copy

first - first pen to copy

RESULT

The palette values are stored in the supplied array.

SEE ALSO

```
LoadPalette32()  
, graphics.library/GetRGB32()
```

1.88 LoadPalette32()

NAME

LoadPalette32 - load a 32 bit palette in a ViewPort

SYNOPSIS

```
LoadPalette32(vp, palette)
           A0      A1
```

```
void LoadPalette32(struct ViewPort *, ULONG *);
```

FUNCTION

This routine loads a 32 bit palette into a ViewPort. It is identical in operation to the graphics.library LoadRGB32() function, except that it also works under OS37.

INPUTS

vp - ViewPort to load palette
palette - Palette to load (in LoadRGB32() format)

RESULT

The palette is loaded.

SEE ALSO

```
GetPalette32
, graphics.library/LoadRGB32()
```

1.89 NewBitMap()

NAME

NewBitMap - allocate a bitmap and bitplanes

SYNOPSIS

```
NewBitMap(sizeX, sizeY, depth, flags, friend)
           D0      D1      D2      D3      A0
```

```
struct BitMap *NewBitMap(ULONG, ULONG, ULONG, ULONG, struct BitMap *);
```

FUNCTION

This routine allocates a BitMap and the bitplanes for it. It is identical in operation to the graphics.library AllocBitMap() call, except that it works under OS37. Under OS37 the bitmap and planes are allocated manually, and the friend parameter is ignored.

Under OS39, if a friend bitmap is supplied and that friend is not a standard BitMap (ie BMF_STANDARD is not set), the new bitmap will be allocated with the BMF_MINPLANES flag. This makes CyberGfx allocate a fast chunky bitmap.

INPUTS

sizeX - width of bitmap
sizeY - height of bitmap
depth - number of bitplanes
flags - bitmap flags

friend - friend bitmap

RESULT

Returns the BitMap if successful, otherwise NULL.

SEE ALSO

```
DisposeBitMap()  
, graphics.library/AllocBitMap()
```

1.90 ScreenInfo()

NAME

ScreenInfo - return information about a screen

SYNOPSIS

```
ScreenInfo(screen)  
A0
```

```
ULONG ScreenInfo(struct Screen *);
```

FUNCTION

This routine is designed to return simple information about a screen.

INPUTS

screen - Screen to obtain information about.

RESULT

Currently, the only flag defined for the result is SCRI_LORES, which indicates that the screen is low resolution, or does not have a 1:1 pixel ratio.

1.91 FindPubScreen()

NAME

FindPubScreen - find (and lock) a public screen

SYNOPSIS

```
FindPubScreen(screen, lock)  
A0 D0
```

```
struct PubScreenNode *FindPubScreen(struct Screen *, BOOL);
```

FUNCTION

This function takes the address of a Screen, and searches for it in the system Public Screen list. If it is found, the address of the PubScreenNode is returned.

INPUTS

screen - Screen to search for
lock - Set to TRUE if you want the screen to be locked on return

RESULT

Returns the PubScreenNode of the screen if found, otherwise NULL. If 'lock' is set to TRUE, the public screen will be locked for you and you should call UnlockPubScreen() on it when you are finished.

SEE ALSO

intuition.library/LockPubScreen()

1.92 SetBusyPointer()

NAME

SetBusyPointer - set busy pointer in a window

SYNOPSIS

```
SetBusyPointer(window)
    A0
```

```
void SetBusyPointer(struct Window *);
```

FUNCTION

This function sets the mouse pointer in the supplied window to the busy pointer. Under OS39 it uses the system-defined busy pointer. Under OS37 it uses a standard watch image.

INPUTS

window - Window to set busy pointer for

RESULT

The busy pointer is set in the supplied window. You should call ClearPointer() when you have finished.

SEE ALSO

intuition.library/SetWindowPointer, intuition.library/ClearPointer

1.93 FreeCachedDiskObject()

NAME

FreeCachedDiskObject - free a cached icon

SYNOPSIS

```
FreeCachedDiskObject(icon)
    A0
```

```
void FreeCachedDiskObject(struct DiskObject *);
```

FUNCTION

This function frees a cached icon obtained via a call to

GetCachedDiskObject()
or a similar function.

INPUTS

icon - icon to free

RESULT

The usage count of the cached icon is decremented. When the usage count reaches zero, the icon is flushed from the cache.

NOTES

You can pass a normal, uncached icon to this routine, in which case it just passes the call through to FreeDiskObject().

SEE ALSO

GetCachedDiskObject()
, icon.library/FreeDiskObject()

1.94 GetCachedDefDiskObject()

NAME

GetCachedDefDiskObject - GetDefDiskObject() with image caching

SYNOPSIS

GetCachedDefDiskObject(type)
DO

```
struct DiskObject *GetCachedDefDiskObject(long);
```

FUNCTION

This routine returns one of a number of default icons. The main advantage this has over the icon.library GetDefDiskObject() call is that the image data of the icons is cached. This can save a huge amount of chip memory if multiple copies of the same icon are required (compare the chip memory used when you open a large drawer in Workbench with the same drawer in Opus).

INPUTS

type - type of icon to create

RESULT

Returns a pointer to the icon or NULL for failure.

SEE ALSO

FreeCachedDiskObject()
, icon.library/GetDefDiskObject()

1.95 GetCachedDiskObject()

NAME

GetCachedDiskObject - get an icon from disk with image caching

SYNOPSIS

GetCachedDiskObject (name)

A0

```
struct DiskObject *GetCachedDiskObject(char *);
```

FUNCTION

This routine loads an icon from disk, and caches the image data. If the same icon is loaded again, the cached image data is used instead of loading a new copy. This can save valuable chip memory, especially as the cache is system wide.

INPUTS

name - name of icon to load (without the .info suffix)

RESULT

Returns a pointer to the icon if successful, otherwise NULL.

NOTES

You should not use this routine if you want to modify the image data. Only the image data is cached, however, so you can modify any of the other fields of the icon.

Also, this routine is slightly slower than a normal call to GetDiskObject(), and while the icon is loading requires slightly more memory (the whole icon is loaded, then if the image is found in the cache the new copy is discarded).

SEE ALSO

```
FreeCachedDiskObject()
, icon.library/GetDiskObject
```

1.96 GetCachedDiskObjectNew()

NAME

GetCachedDiskObjectNew - get icon with default fallback

SYNOPSIS

GetCachedDiskObjectNew (name)

A0

```
struct DiskObject *GetCachedDiskObjectNew(char *);
```

FUNCTION

This routine attempts to load the icon in the same way as the

```
GetCachedDiskObject()
routine. If no icon is found for the supplied
filename, the object in question is examined, and a default icon is
```

returned. This routine will return WBPROJECT, WBDRAWER, WBTOOL or WBDISK icons, depending on the name passed in.

INPUTS

name - name of object to load icon for (no .info suffix)

RESULT

Returns a pointer to the icon if successful, otherwise NULL.

SEE ALSO

```

        GetCachedDiskObject()
        ,
        FreeCachedDiskObject()
        ,
icon.library/GetDiskObjectNew()

```

1.97 GetIconFlags()

NAME

GetIconFlags - get special Opus icon flags

SYNOPSIS

```

GetIconFlags(icon)
        A0

```

```

ULONG GetIconFlags(struct DiskObject *);

```

FUNCTION

Opus stores additional information in icons to control some of the enhanced features. This routine returns the special flags set for the icon you supply.

INPUTS

icon - icon to retrieve flags for

RESULT

Returns ULONG containing the flags set. Current flags in use are :

```

ICONF_POSITION_OK    - an Opus-specific position is available
ICONF_NO_BORDER      - icon has no border
ICONF_NO_LABEL        - icon has no label

```

SEE ALSO

```

        SetIconFlags()
        ,
        GetIconPosition()

```

1.98 GetIconPosition()

NAME

GetIconPosition - get Opus-specific icon position

SYNOPSIS

```
GetIconPosition(icon, xptr, yptr)
    A0      A1      A2
```

```
void GetIconPosition(struct DiskObject *, short *, short *);
```

FUNCTION

Opus keeps a separate record from Workbench of icon positions. This function allows you to retrieve the Opus-specific position of the icon (the normal Workbench position is in do_CurrentX/do_CurrentY).

INPUTS

icon - icon to retrieve position for
 xptr - pointer to short to receive x position
 yptr - pointer to short to receive y position

RESULT

Stores the position in the two variables provided.

NOTES

You should call

```
    GetIconFlags()
    first to check that an Opus-specific
position is available for this icon.
```

SEE ALSO

```
    SetIconPosition()
    ,
    GetIconFlags()
```

1.99 SetIconFlags()

NAME

SetIconFlags - set Opus flags in an icon

SYNOPSIS

```
SetIconFlags(icon, flags);
    A0      D0
```

```
void SetIconFlags(struct DiskObject *, ULONG);
```

FUNCTION

This routine allows you to set the special Opus flags in an icon.

INPUTS

icon - icon to set flags for
 flags - new flags for the icon

RESULT

The flags in the icon are set. See
 GetIconFlags()
 for a description
of the available flags.

SEE ALSO

 GetIconFlags()

1.100 SetIconPosition()

NAME

SetIconPosition - set Opus position for an icon

SYNOPSIS

```
SetIconPosition(icon, x, y)
    A0  D0 D1
```

```
void SetIconPosition(struct DiskObject *, short, short);
```

FUNCTION

This routine allows you to set the Opus-specific position for an icon.

INPUTS

icon - icon to set position for
x - new x position of icon
y - new y position of icon

RESULT

The position is set in the icon.

NOTES

You should also call
 SetIconFlags()
 on the icon to set the
ICONF_POSITION_OK flag.

SEE ALSO

 GetIconPosition()
 ,
 SetIconFlags()

1.101 CopyFileIcon()

NAME

CopyFileIcon - copy icon from one file to another

SYNOPSIS

```
CopyFileIcon(source, dest)
    A0      A1
```

```
void CopyFileIcon(char *, char *);
```

FUNCTION

This routine copies the icon from the specified source object to a new icon for the destination object.

INPUTS

source - source icon (without .info)
dest - destination icon (without .info)

RESULT

The icon is copied. If an icon already exists for the source, it is NOT overwritten. No error code is available for this function.

NOTES

If the specified source file has no icon, a default icon is created.

1.102 IFF_Routines

IFF Routines

IFFChunkID ()

IFFChunkRemain ()

IFFChunkSize ()

IFFClose ()

IFFFailure ()

IFFGetForm ()

IFFNextChunk ()

IFFOpen ()

IFFPopChunk ()

IFFPushChunk ()

IFFReadChunkBytes ()

IFFWriteChunkBytes ()

IFFWriteChunk ()

1.103 IFFChunkID()

NAME

IFFChunkID - get current chunk ID

SYNOPSIS

```
IFFChunkID(handle)
    A0
```

```
ULONG IFFChunkID(APTR);
```

FUNCTION

Returns the four-byte ID of the current chunk.

INPUTS

handle - IFF handle

RESULT

Returns chunk ID.

SEE ALSO

```
IFFOpen()
, IFFGetFORM(),
IFFNextChunk()
```

1.104 IFFChunkRemain()

NAME

IFFChunkRemain - return size of remaining data

SYNOPSIS

```
IFFChunkRemain(handle)
    A0
```

```
long IFFChunkRemain(APTR);
```

FUNCTION

Returns the amount of data left to be read in the current chunk.

INPUTS

handle - IFF handle

RESULT

Returns size of data or 0 if no data left.

SEE ALSO

```
IFFOpen()
,
IFFNextChunk()
,
IFFChunkSize()
```

```
,  
IFFReadChunkBytes()
```

1.105 IFFChunkSize()

NAME

IFFChunkSize - size of current chunk

SYNOPSIS

```
IFFChunkSize(handle)
```

A0

```
long IFFChunkSize(APTR);
```

FUNCTION

Obtain the amount of data in the current chunk.

INPUTS

handle - IFF handle

RESULT

Returns the total number of bytes of data in the current chunk (excluding the chunk header). If there is no valid current chunk, returns -1.

SEE ALSO

```
IFFOpen()  
,  
IFFNextChunk()  
,  
IFFChunkRemain()  
,  
IFFReadChunkBytes()
```

1.106 IFFClose()

NAME

IFFClose - close an IFF file

SYNOPSIS

```
IFFClose(handle)
```

A0

```
void IFFClose(APTR);
```

FUNCTION

Closes an IFF file.

INPUTS

handle - file to close

RESULT

The file is closed.

SEE ALSO

IFFOpen()

1.107 IFFFailure()

NAME

IFFFailure - indicate failure to "safe" file routine

SYNOPSIS

IFFFailure(handle)
A0

void IFFFailure(APTR)

FUNCTION

The

IFFOpen()

routine has a "safe" mode, where an existing IFF file is renamed rather than being over-written. If you create a file in this safe mode, and then subsequent IFF operations fail (with the result that the whole operation fails), you should call this routine. Then, when you call

IFFClose()

, the pre-existing file will be renamed back to what it was instead of being deleted.

INPUTS

handle - IFF handle

RESULT

The 'failure' flag is set, and the "safe" file will be restored.

SEE ALSO

IFFOpen()

,

IFFClose()

1.108 IFFGetForm()

NAME

IFFGetForm - get current FORM ID

SYNOPSIS

```
IFFGetForm(handle)
    A0
```

```
ULONG IFFGetForm(APTR);
```

FUNCTION

This routine returns the four-byte FORM ID of the file.

INPUTS

handle - IFF handle

RESULT

Returns the FORM ID, or 0 if no valid FORM.

SEE ALSO

```
    IFFOpen()
    ,
    IFFChunkID()
```

1.109 IFFNextChunk()

NAME

IFFNextChunk - skip to the next chunk

SYNOPSIS

```
IFFNextChunk(handle, chunkid)
    A0      D0
```

```
ULONG IFFNextChunk(APTR, ULONG);
```

FUNCTION

This is the main work-horse of the IFF routines. This function will scan through the IFF file from the current position looking for the next chunk. You can optionally specify a chunk to look for.

INPUTS

handle - IFF handle

chunkid - ID of chunk to look for, or 0 for next chunk

RESULT

Returns ID of the new chunk, or 0 for error or end-of-file.

NOTES

The IFF routines do not handle LISTS, CATs or other complicated IFF structures. This routine will handle multiple FORMs within the one file, which allows it to read ANIMs.

SEE ALSO

```
    IFFOpen()
    ,
    IFFReadChunkBytes()
```

1.110 IFFOpen()

NAME

IFFOpen - open an IFF file

SYNOPSIS

```
IFFOpen(name, mode, form)
    A0    D0    D1
```

```
APTR IFFOpen(char *, USHORT, ULONG);
```

FUNCTION

This routine opens either a disk-based file or the clipboard for IFF reading/writing. This and the other IFF routines in the dopus5.library are far simpler (and seem to be less buggy) than the iffparse.library functions. These routines were specifically designed to make the reading and writing of "normal" IFF files as easy as possible. They use the buffered IO routines for high performance. They do not support LISTS, CATs or other complicated IFF structures.

INPUTS

name - Filename or clipboard unit to open.

mode - Mode to open in :

```
IFF_READ      - open for reading
IFF_WRITE     - open for writing
IFF_CLIP_READ - open clipboard for reading
IFF_CLIP_WRITE - open clipboard for writing
```

This flag can be set in conjunction with IFF_WRITE :

```
IFF_SAFE      - backup existing file
```

form - If this is not 0, it specifies an IFF FORM. If you open the file for reading, it will be tested against this FORM and the open will fail if the FORM does not match. If you open for writing, this specifies the FORM of the file to be created.

RESULT

If successful, returns an IFF handle which you use in all subsequent calls to the IFF routines. Returns 0 on failure, and IoErr() is set to the reason for the failure.

NOTES

To use the clipboard, you must specify either IFF_CLIP_READ or IFF_CLIP_WRITE. The clipboard unit you want is passed as the name parameter. For example, to open clipboard unit 3 to write an ILBM,

```
handle = IFFOpen( (char *)3, IFF_CLIP_WRITE, ID_ILBM );
```

If you specify IFF_WRITE|IFF_SAFE, and the file you are creating already exists, it will be renamed to a temporary filename. When

IFFClose()
 is called, the temporary file will be deleted. If you
 call

IFFFailure()
 because of a failure at some stage, it sets a flag
 which causes the old file to be restored.

SEE ALSO

IFFClose()
 ,
 IFFFailure()

1.111 IFFPopChunk()

NAME

IFFPopChunk - flush the chunk cache

SYNOPSIS

IFFPopChunk(handle)
 A0

long IFFPopChunk(APTR);

FUNCTION

This function flushes the current chunk write cache and writes the
 chunk to disk.

INPUTS

handle - IFF handle

RESULT

The current chunk is written to disk. Returns TRUE if successful.

SEE ALSO

IFFOpen()
 ,
 IFFPushChunk()
 ,
 IFFWriteChunkBytes()

1.112 IFFPushChunk()

NAME

IFFPushChunk - start writing a chunk

SYNOPSIS

IFFPushChunk(handle, id)

A0 D0

```
long IFFPushChunk(APTR, ULONG);
```

FUNCTION

This routine initialises a new chunk to be written to the file. It is similar in concept to the PushChunk() routine in iffparse.library, but does not support nesting of chunks. Therefore, you should always call

```
    IFFPopChunk()
    before you call IFFPushChunk()
```

again.

INPUTS

handle - IFF handle
id - ID of chunk to write

RESULT

The chunk is initialised for writing. The usual procedure is :

1. IFFPushChunk()
 2. One or more calls to


```
    IFFWriteChunkBytes()
```
 3.


```
    IFFPopChunk()
```
- SEE ALSO
- ```
IFFOpen()
',
IFFWriteChunkBytes()
',
IFFPopChunk()
```

## 1.113 IFFReadChunkBytes()

#### NAME

IFFReadChunkBytes - read data from a chunk

#### SYNOPSIS

```
IFFReadChunkBytes(handle, buffer, size)
```

A0 A1 D0

```
long IFFReadChunkBytes(APTR, APTR, long);
```

#### FUNCTION

This routine reads data from the current position in the current chunk.

#### INPUTS

handle - IFF handle  
buffer - buffer to store data  
size - amount of data to read

#### RESULT

Returns the amount of data read or -1 for failure. Will not read past the end of a chunk.

SEE ALSO

```
IFFOpen()
,
IFFNextChunk()
,
IFFChunkSize()
```

## 1.114 IFFWriteChunkBytes()

NAME

IFFWriteChunkBytes - write data in a chunk

SYNOPSIS

```
IFFWriteChunkBytes(handle, data, size)
 A0 A1 D0
```

```
long IFFWriteChunkBytes(APTR, APTR, long);
```

FUNCTION

This routine writes data to a chunk that was initialised with PushChunk(). The data is generally cached and not written to the disk until PopChunk() is called, resulting in higher performance.

INPUTS

handle - IFF handle  
data - data to write  
size - amount of data

RESULT

Returns TRUE if successful.

SEE ALSO

```
IFFOpen()
,
IFFPushChunk()
,
IFFPopChunk()
```

## 1.115 IFFWriteChunk()

NAME

IFFWriteChunk - write a chunk straight out

SYNOPSIS

```
IFFWriteChunk(handle, data, id, size)
```



A0     A1     D0     D1

```
long IFFWriteChunk (APTR, APTR, ULONG, ULONG);
```

#### FUNCTION

If you have a single structure or piece of data you wish to write as a chunk, this routine is simpler than the PushChunk()/PopChunk() method.

#### INPUTS

handle - IFF handle  
data - data to write  
id - ID of chunk  
size - size of data

#### RESULT

If successful, the chunk is written straight to the file. Returns FALSE on failure.

#### SEE ALSO

```
IFFOpen ()
,
IFFPushChunk ()
,
IFFPopChunk ()
,
IFFWriteChunkBytes ()
```

## 1.116 Image\_Routines

Image Routines

CloseImage ()

CopyImage ()

FreeImageRemap ()

FreeRemapImage ()

GetImageAttrs ()

GetImagePalette ()

OpenImage ()

RemapImage ()

RenderImage ()

---

## 1.117 CloseImage()

NAME

CloseImage - close an image

SYNOPSIS

```
CloseImage(image)
 A0
```

```
void CloseImage(APTR);
```

FUNCTION

Closes an image that was opened with `OpenImage()`. The usage count of the image is decremented. When the count reaches 0 the image is flushed from memory.

INPUTS

image - image to close

SEE ALSO

`OpenImage()`

## 1.118 CopyImage()

NAME

CopyImage - copy an opened image

SYNOPSIS

```
CopyImage(image)
 A0
```

```
APTR CopyImage(APTR);
```

FUNCTION

Returns another pointer to the supplied image.

INPUTS

image - image to copy

RESULT

Returns a new pointer to the image.

SEE ALSO

`OpenImage()`

---

## 1.119 FreeImageRemap()

### NAME

FreeImageRemap - free pens used to remap images

### SYNOPSIS

```
FreeImageRemap(remap)
 A0
```

```
void FreeImageRemap(ImageRemap *);
```

### FUNCTION

Frees all the pens allocated with the supplied ImageRemap structure. You should call this function after you have called

```
FreeRemapImage()
 or
CloseImage()
 on the individual images.
```

### INPUTS

remap - ImageRemap structure to free

### SEE ALSO

```
RemapImage()
```

## 1.120 FreeRemapImage()

### NAME

FreeRemapImage - free a remapped image

### SYNOPSIS

```
FreeRemapImage(image, remap)
 A0 A1
```

```
void FreeRemapImage(APTR, ImageRemap *);
```

### FUNCTION

This function frees the remapped bitplanes allocated for an image via the

```
RemapImage()
 call.
```

### INPUTS

image - image to free remap bitplanes for  
remap - ImageRemap structure

### RESULT

The remapped bitplanes are freed. This routine does not free any pens that were allocated - these are released when you call

```
FreeImageRemap()
```

Note that the image itself is not freed, only the remapped version of

---

it.

SEE ALSO

```
RemapImage()
,
FreeImageRemap()
```

## 1.121 GetImageAttrs()

NAME

GetImageAttrs - get information about an image

SYNOPSIS

```
GetImageAttrs(image, tags)
 A0 A1
```

```
void GetImageAttrs(APTR, struct TagItem *);
```

FUNCTION

This routine allows you to retrieve information about an image opened with

```
OpenImage()
```

.

INPUTS

image - image to investigate

tags - control tags. The following tags are valid :

```
IM_Width - width of image
IM_Height - height of image
IM_Depth - number of bitplanes
IM_State - 1 if the image has two frames, 0 if not
```

RESULT

The requested information is stored in the ti\_Data field of each of the Tags passed in.

SEE ALSO

```
OpenImage()
```

## 1.122 GetImagePalette()

NAME

GetImagePalette - get pointer to image palette

SYNOPSIS

```
GetImagePalette(image)
 A0
```

```
ULONG *GetImagePalette(APTR);
```

#### FUNCTION

This allows you to retrieve a pointer to the palette of the image.

#### INPUTS

image - image you want the palette for

#### RESULT

If the image has associated palette information (eg a brush), a pointer to a longword palette table is returned. This palette table is in LoadRGB32() format. If the image has no associated palette, this routine returns NULL.

#### SEE ALSO

```
OpenImage()
, graphics.library/LoadRGB32()
```

## 1.123 OpenImage()

#### NAME

OpenImage - read an image off disk

#### SYNOPSIS

```
OpenImage(name, info)
 A0 A1
```

```
APTR OpenImage(char *, OpenImageInfo *);
```

#### FUNCTION

The primary purpose of this function is to read an image from a file on disk. This routine supports ILBM brushes and pictures, animbrushes and Amiga icons.

This routine is also used to create an image handle to bitmap data that you supply. This image handle can then be used with the image remapping functions.

This function caches images based on their full pathname. If two copies of the same file are loaded, the first copy will be used to save memory.

#### INPUTS

name - name of image to load, or NULL if you are supplying a bitmap  
 info - if 'name' is NULL, this must point to an initialised  
 OpenImageInfo structure:

oi\_ImageData - must point to the image data. This data does not need to be in chip memory.

oi\_Palette - must point to a palette for the image, in LoadRGB32() format.

oi\_Width - width of the image

oi\_Height - height of the image

oi\_Depth - number of image bitplanes

#### RESULT

Returns an image handle if it succeeds. This handle is used in subsequent calls to the image routines. This routine returns NULL if it fails.

#### SEE ALSO

CloseImage()  
 , graphics.library/LoadRGB32()

## 1.124 RemapImage()

#### NAME

RemapImage - remap an image

#### SYNOPSIS

RemapImage(image, screen, remap)  
           A0          A1          A2

BOOL RemapImage(APTR, struct Screen \*, ImageRemap \*);

#### FUNCTION

This function remaps an image to the colours of the specified screen. It will allocate pens from the screen if necessary (and possible).

#### INPUTS

image - image to remap (from  
           OpenImage()  
           )

remap - ImageRemap structure. This structure must be initialised for the first call to RemapImage(). All fields must be set to NULL.

For the first and subsequent calls to this function, the ir\_Flags field can be set with the following values :

IRF\_REMAP\_COLO - remap colour 0 in the image  
 IRF\_PRECISION\_EXACT - use best precision when pen matching  
 IRF\_PRECISION\_ICON - lower precision  
 IRF\_PRECISION\_GUI - lowest precision

You can use the one ImageRemap structure to remap multiple images, but only for the one screen. The ir\_Flags field can be changed for every call to this function, but none of the other fields may be changed.

#### RESULT

This image returns TRUE if it was able to remap the image. Once the image has been remapped, any call to

```
RenderImage()
 to display it will
```

show the remapped version. Call

```
FreeRemapImage()
 to free the remap
```

and return to the original image.

SEE ALSO

```
OpenImage()
,
RenderImage()
,
FreeRemapImage()
,
FreeImageRemap()
```

## 1.125 RenderImage()

NAME

RenderImage - display an image

SYNOPSIS

```
RenderImage(rp, image, left, top, tags)
 A0 A1 D0 D1 A2
```

```
short RenderImage(struct RastPort *, APTR, USHORT, USHORT,
 struct TagItem *);
```

FUNCTION

This routine is used to render an image to a RastPort.

INPUTS

rp - RastPort to render to  
image - image to render  
left - x position to render to  
top - y position to render to  
tags - control tags. The following tags are available :

IM\_State - 0 or 1 (default 0)

This tag controls which frame of the image is shown. Defaults to frame 0, but for two-frame images (eg icons or animbrushes) you can set this to 1.

IM\_Rectangle - struct Rectangle \* (default not supplied)

This specifies a rectangle to display the image within. If you supply this tag, the image will be centred within this area. Use of this tag overrides the 'left' and 'top' parameters.

IM\_ClipBoundary - integer (default 2)

This is used with the IM\_Rectangle tag. If IM\_Rectangle is specified, the image is clipped to the boundaries of the rectangle. The default operation is to leave a two pixel margin around the image (to allow room for a border). Using the IM\_ClipBoundary tag you can adjust this margin (set to 0 if you want no margin).

IM\_Mask - TRUE or FALSE (default FALSE)

If you set this tag to TRUE, the image will be masked when it is rendered. This has the effect of making colour 0 transparent, and the existing background will show through the image.

IM\_Erase - integer (default not supplied)

This tag allows you to specify a pen value that is used to erase the background before the image is rendered. By default the background is not cleared.

IM\_NoDrawInvalid (default not supplied)

If you specify this tag, and also specify 1 for IM\_State, then the call to RenderImage() will fail if the image has no secondary image. If this tag is not specified and you try to draw the second frame of an image that doesn't have one, it falls back to drawing the first frame.

IM\_NoIconRemap - TRUE or FALSE (default FALSE)

By default, an icon that is drawn with RenderImage() is "remapped". This is not a true colour remapping, but the third bitplane of an eight colour icon is shifted to the top bitplane of the display. This makes most normal eight colour icons work properly on screens of more than eight colours. However, it can cause problems with NewIcons icons. Specify TRUE with this tag to disable this remapping.

#### RESULT

The image is rendered. If you specified IM\_NoDrawInvalid and you tried to draw an image that didn't exist, this routine returns FALSE. Otherwise it returns TRUE.

#### NOTES

If the image has been remapped with  
    RemapImage()  
    , the remapped image  
will be automatically drawn by this routine.

#### SEE ALSO

OpenImage()  
,  
RemapImage()



## 1.126 IPC\_Routines

IPC Routines

IPC\_Command()

IPC\_FindProc()

IPC\_Flush()

IPC\_Free()

IPC\_Launch()

IPC\_ListCommand()

IPC\_ProcStartup()

IPC\_Reply()

## 1.127 IPC\_Command()

NAME

IPC\_Command - send a command to an IPC process

SYNOPSIS

```
IPC_Command(ipc, command, flags, data, data_free, reply)
 A0 D0 D1 A1 A2 A3
```

```
ULONG IPC_Command(IPCData *, ULONG, ULONG, APTR, APTR,
 struct MsgPort *);
```

FUNCTION

Sends a command to an IPC process. Can be used from a non-IPC process but this is not recommended.

INPUTS

ipc - IPC process to send command to

command - command code (application-specific)

flags - command flags (application-specific)

data - command data (application-specific)

data\_free - additional data. The data specified here will be automatically freed with FreeVec() when the message is replied to, so you MUST allocate it with

AllocVec().

reply - reply port. You can either specify a message port for the reply, or use one of these special values :

REPLY\_NO\_PORT - use default port for reply  
 REPLY\_NO\_PORT\_IPC - specify this if the message is sent from a non-IPC process

If you don't want a reply to this message (ie you want it to be sent asynchronously), specify NULL for this value.

#### RESULT

The command will be sent to the specified process. If the command was not sent asynchronously, the destination process' result code is returned.

#### NOTES

There are several reserved command codes. You are free to use these for your own applications, or use your own codes. These are listed in <dopus/ipc.h>.

#### SEE ALSO

IPC\_Launch()  
 ,  
 IPC\_Reply()

## 1.128 IPC\_FindProc()

#### NAME

IPC\_FindProc - find an IPC process by name

#### SYNOPSIS

IPC\_FindProc(list, name, activate, data)  
           A0      A1      D0      D1

IPCData \*IPC\_FindProc(struct ListLock \*, char \*, BOOL, ULONG);

#### FUNCTION

This routine searches the supplied list for a named IPC process. Optionally, it can send this process an IPC\_ACTIVATE message with user-specified data.

#### INPUTS

list - ListLock to search. This routine locks this list in shared mode, and will block until the list is available.

name - name to search for (case sensitive)

activate - specify TRUE if you want an IPC\_ACTIVATE message to be sent automatically.

data - if 'activate' is TRUE, this value will be passed in the data

field of the IPC\_ACTIVATE command.

#### RESULT

If the process is found, its IPCData pointer is returned. To ensure that this pointer remains valid you should Forbid(), or lock the list yourself (in shared mode!).

#### SEE ALSO

```
IPC_Launch()
,
IPC_Command()
```

## 1.129 IPC\_Flush()

#### NAME

IPC\_Flush - flush an IPC command port

#### SYNOPSIS

```
IPC_Flush(ipc)
A0
```

```
void IPC_Flush(IPCData *);
```

#### FUNCTION

This routine searches the command port for any messages and replies to them with an IPC\_ABORT.

#### INPUTS

ipc - IPCData of the process

#### RESULT

The port is emptied.

#### NOTES

In practice you rarely need this function.

#### SEE ALSO

```
IPC_Free()
```

## 1.130 IPC\_Free()

#### NAME

IPC\_Free - free an IPC process

#### SYNOPSIS

```
IPC_Free(ipc)
A0
```

```
void IPC_Free(IPCData *);
```

#### FUNCTION

This routine frees all the memory associated with an IPC process entry. It does NOT remove the process itself from the system. It is designed to be called by a process on itself, as the last step before exiting.

#### INPUTS

ipc - IPCData to free

#### RESULT

The IPCData handle is freed. If the process was a member of a list, it is removed from that list. Any commands still in its message port are replied to with IPC\_ABORT.

#### SEE ALSO

IPC\_Launch()

## 1.131 IPC\_Launch()

#### NAME

IPC\_Launch - launch a new process

#### SYNOPSIS

```
IPC_Launch(list, ipcptr, name, entry, stack, data, doslib)
 A0 A1 A2 D0 D1 D2 A3
```

```
long IPC_Launch(struct ListLock *, IPCData **, char *,
 ULONG, ULONG, ULONG, struct Library *);
```

#### FUNCTION

The IPC routines in the dopus5.library provide an easy and efficient way to implement multi-threading in your application. Using the IPC\_Launch functions creates an IPCData, which is a handle to a process. Using this handle you can easily send command between multiple processes.

The important fields in the IPCData structure are as follows :

```
proc - pointer to the Process structure
command_port - port to listen to for commands
userdata - user-specified data
memory - a memory pool you can use
```

All Opus 5 modules are launched as IPC processes, and are passed a pointer to their IPCData structures. They are also passed the address of the main Directory Opus IPCData structure, which allows them to send direct commands to Opus.

If you are writing a stand-alone application and wish to use the IPC routines, your main thread will need to create an IPCData structure manually. It must be initialised as follows :

```
proc - pointer to your main Process
command_port - pointer to a message port
list - NULL
reply_port - pointer to a DIFFERENT message port
```

IPC processes can automatically be added to a list. There is no need for you to keep track of a process once it has been launched, except that your main process can't exit while a child is still running (as the code would be freed).

#### INPUTS

list - pointer to an initialise ListLock structure if you want this process to be automatically added to a list (can be NULL).

ipcptr - pointer to a pointer to IPCData. If the process is launched successfully, the new IPCData handle will be stored in this address (can be NULL).

name - name for the new process.

entry - pointer to code for the new process.

stack - stack size for the new process. You can also set the IPCF\_GETPATH flag in the stack variable, to have the new process automatically inherit the system path list.

data - data that is automatically passed to the new process

doslib - you must supply a pointer to the DOS library

#### RESULT

This routine returns 0 if the child process failed to launch. However, if the child process was actually launched, but failed to initialise because of lack of memory, or a failure in your user-defined initialisation code (see

```
 IPC_ProcStartup
), the
```

return value will still indicate success.

A better way to test failure is to specify a variable for the 'ipcptr' parameter. If this is NULL after this call, the process failed to start.

#### SEE ALSO

```
 IPC_ProcStartup()
 ,
 IPC_Command()
 ,
 IPC_Free
```

## 1.132 IPC\_ListCommand()

## NAME

IPC\_ListCommand - send a command to a list of processes

## SYNOPSIS

```
IPC_ListCommand(list, command, flags, data, wait)
 A0 D0 D1 D2 D3
```

```
void IPC_ListCommand(struct ListLock *, ULONG, ULONG, ULONG, BOOL);
```

## FUNCTION

Sends the same command to every process on the supplied list. Optionally waits for a reply from every process.

## INPUTS

list - list of processes  
 command - command ID to send  
 flags - command flags  
 data - command data  
 wait - specify TRUE if you want to wait for replies

## RESULT

The command is sent to every process on the list. If 'wait' is TRUE, does not return until every process has replied.

## SEE ALSO

```
IPC_Launch()
,
IPC_Command()
```

### 1.133 IPC\_ProcStartup()

## NAME

IPC\_ProcStartup - startup code for an IPC process

## SYNOPSIS

```
IPC_ProcStartup(data, code)
 A0 A1
```

```
IPCData *IPC_ProcStartup(ULONG *, ULONG *);
```

## FUNCTION

Your IPC process should call this routine as the very first instruction. It receives the startup message from the parent process, and lets you retrieve your own IPCData handle.

## INPUTS

data - pointer to a variable to receive a pointer to the data that was passed to  
       IPC\_Launch  
       .

code - address of a user-supplied initialisation routine to call. If

you provide this, your routine is called from this function.  
The prototype of this routine is as follows :

```
ULONG __asm code(register __a0 IPCData *ipc,
 register __a1 APTR data)
```

Your intitialisation routine receives a pointer to the IPCData handle of the new process, and a pointer to the data passed to

IPC\_Launch

. This routine can do pretty much anything, but you should keep it as simple as possible (there should certainly be no IPC functions called from within it). Your routine should return FALSE for failure and TRUE for success. If it returns FALSE, the IPC\_ProcStartup() call will return NULL, and the new process should then quit.

RESULT

Returns a pointer to your new IPCData handle. The 'data' value that was passed to

IPC\_Launch()

is stored in the supplied variable.

If this routine returns NULL, it means an error occurred (either in the initialisation of the new process, or in your own initialisation code). In this case, you should exit immediately.

SEE ALSO

IPC\_Launch()

## 1.134 IPC\_Reply()

NAME

IPC\_Reply - reply to an IPC message

SYNOPSIS

IPC\_Reply(msg)

A0

```
void IPC_Reply(IPCMessage *);
```

FUNCTION

Call this routine to reply to IPCMessages you receive at your command port (do not call ReplyMsg())

INPUTS

msg - message to reply

RESULT

The message is replied. It is possible to pass a return code back to the sending task (providing the message was sent synchronously). To do this, set the 'command' field of the message to the value. This will then be the return from the

IPC\_Command()

function for the  
other process.

SEE ALSO

IPC\_Launch()  
,  
IPC\_Command()

## 1.135 Layout\_Routines

Layout Routines

AddObjectList()

AddWindowMenus()

BoundsCheckGadget()

BuildMenuStrip()

CheckObjectArea()

ClearWindowBusy()

CloseConfigWindow()

DisableObject()

DisplayObject()

EndRefreshConfigWindow()

FindMenuItem()

FreeObjectList()

FreeWindowMenus()

GetGadgetValue()

GetObject()

GetObjectRect()

GetWindowAppPort()

GetWindowID()

GetWindowMsg()

LayoutResize()



```

OpenConfigWindow()

ReplyWindowMsg()

SetConfigWindowLimits()

SetGadgetChoices()

SetGadgetValue()

SetWindowBusy()

SetWindowID()

StartRefreshConfigWindow()
PURPOSE

```

1. The `dopus5.library` provides font-sensitive layout routines to make it easy to create and use a user interface for your application or module. The layout code is not as straightforward, or indeed as powerful, as MUI or some of the other GUI engines available, and so you might want to consider using one of those instead of the DOpus routines.

The normal procedure in creating an interface is :

1. Define a list of "objects" (gadgets, text, etc)
2. Define a window
3. Call

```

OpenConfigWindow()
to open the window

```

4. Call

```

AddObjectList()
to add the objects

```

5. Call

```

SetGadgetValue()
to initialise gadgets

```

6. Message loop with

```

GetWindowMsg()

```

7. Call

```

GetGadgetValue()
to get gadget final values

```

8. Call

```

CloseConfigWindow()
to close the window

```

2. The list of objects is defined as an array of `ObjectDef` structures.

You call

```

AddObjectList()

```

to add a list of objects to a window; indeed, you can make multiple calls to this function to add multiple lists. Each `ObjectDef` structure is defined as follows:

`od_Type`

This field indicates the type of object. Current values are :

OD\_GADGET - a gadget  
 OD\_TEXT - a text string  
 OD\_AREA - a rectangular area  
 OD\_IMAGE - an image

Two control values are also used:

OD\_SKIP - skip this entry  
 OD\_END - ends an ObjectDef array

#### od\_ObjectKind

If `od_Type` is set to `OD_GADGET`, this field describes the type of gadget. Valid types are:

BUTTON\_KIND - standard push button  
 STRING\_KIND - a string gadget  
 INTEGER\_KIND - an integer gadget  
 HOTKEY\_KIND - a hotkey field  
 CHECKBOX\_KIND - a checkbox gadget  
 OPUS\_LISTVIEW\_KIND - an Opus listview gadget  
 PALETTE\_KIND - a palette gadget (pen selection)  
 FILE\_BUTTON\_KIND - a button that opens a file requester  
 DIR\_BUTTON\_KIND - a button that opens a directory requester  
 FONT\_BUTTON\_KIND - a button that opens a font requester  
 (only works under v38 ASL)  
 FRAME\_KIND - a frame (rectangular area)  
 FIELD\_KIND - a readonly string gadget  
 NUMBER\_KIND - a number display  
 TEXT\_KIND - a text display

The above gadgets are all implemented by the `dopus5.library`. Many are similar to their GadTools equivalents. Any GadTools gadget that is not replaced by the above types is also available. Currently, these are:

LISTVIEW\_KIND - standard GadTools listview  
 MX\_KIND - radio buttons gadget  
 CYCLE\_KIND - cycle gadget  
 SCROLLER\_KIND - scroller gadget  
 SLIDER\_KIND - slider gadget

If `od_Type` is `OD_AREA` or `OD_TEXT`, the `od_ObjectKind` field is used to specify the pen used for text rendering. This is not a literal pen number but is a DrawInfo pen index (eg `TEXTPEN`).

#### od\_CharDims

This field structure is used to define the character position of the object. All objects (and windows, for that matter) have two sets of positioning information - character and fine. The character position is used to specify the size and position in "font units" - this value is scaled for the current font and so allows the display to be font-sensitive. A "font unit" is the average width of the font for a horizontal coordinate, and the height of the font for a vertical coordinate. The "Left" and "Top" fields of the IBox structure control

the position of the object, and the "Width" and "Height" fields control the size.

There are some magic values for object positioning and sizing. The positioning values are:

POS\_CENTER - use this in the "Left" or "Top" field to have the object positioned in the center of the window.

POS\_RIGHT\_JUSTIFY - position relative to the right/bottom border

POS\_CENTER is an absolute value. POS\_RIGHT\_JUSTIFY is more of a flag, used in conjunction with another value. For example, POS\_RIGHT\_JUSTIFY by itself would position an object hard up against the right border. POS\_RIGHT\_JUSTIFY-2 would position an object at a two 'font unit' margin from the border.

The magic values for sizes are:

SIZE\_MAXIMUM - the object will be the maximum possible size in this direction

SIZE\_MAX\_LESS - maximum possible size minus an amount

For example, `od_CharDims` might be defined as:

```
{POS_CENTER, POS_RIGHT_JUSTIFY-1, SIZE_MAX_LESS-4, 2}
```

This would create an object that was centred in the display, and as wide as possible with a two space gap on either side. The object would be one space from the bottom border, and be two spaces high.

#### `od_FineDims`

This field structure is used to define the fine position and size of the object. The fine position is given as an absolute pixel value, and allows you to make adjustments for GUI components that do not scale with the font (eg, borders are always 1 or 2 pixels high, irrespective of the font size). You can also make fine adjustments through this field when the `od_CharDims` field uses the `POS_` and `SIZE_` values.

#### `od_GadgetText`

This is the locale string ID for the gadget label, or text or area string. It must be a valid ID in the locale specified in the `NewConfigWindow` structure. If the `TEXTFLAG_TEXT_STRING` flag is set for this object, the `od_GadgetText` field is a pointer to an actual text string.

#### `od_Flags`

The object flags are heavily object dependent. For standard GadTools gadgets, the standard GadTools flags apply. Some GadTools flags are

also applicable to Opus gadgets:

```
PLACETEXT_LEFT
PLACETEXT_RIGHT
PLACETEXT_ABOVE (only works for some gadgets)
PLACETEXT_IN (only works for some gadgets)
```

General purpose Opus flags are:

TEXTFLAG\_TEXT\_STRING - this flag is used with all the object types, and indicates that the `od_GadgetText` field of the `ObjectDef` structure points to a literal text string and not a locale ID.

TEXTFLAG\_NO\_USCORE - if you specify this flag, an underscore character in the string will be treated literally. Otherwise, the underscore is used to specify a character to be underscored, indicating a keyboard equivalent.

Flags for `BUTTON_KIND` gadgets:

BUTTONFLAG\_OKAY\_BUTTON - indicates that this button is an "ok" button. The 'enter' key will automatically be used as a key equivalent for this button. The label for this button is automatically rendered in bold.

BUTTONFLAG\_CANCEL\_BUTTON - indicates that this button is a "cancel" button. The 'escape' key will automatically be used as a key equivalent for this button.

BUTTONFLAG\_TOGGLE\_SELECT - specifies that you want a toggle-select button (one that can be turned on or off).

BUTTONFLAG\_THIN\_BORDERS - specifies that you want 'thin' borders for the button. Thin borders are one pixel wide on all sides, whereas normal borders are two pixels wide on the left and right, and one pixel wide at the top and bottom. This flag can also be used with `CHECKBOX_KIND`, `FRAME_KIND`, `NUMBER_KIND`, `TEXT_KIND`, `PALETTE_KIND`, `FILE_BUTTON_KIND`, `DIR_BUTTON_KIND` and `FONT_BUTTON_KIND` gadgets.

Flags for `OPUS_LISTVIEW_KIND` gadgets:

LISTVIEWFLAG\_CURSOR\_KEYS - specifies that the cursor keys can be used to scroll up and down in this listview.

Flags for `FILE_BUTTON_KIND` gadgets:

FILEBUTFLAG\_SAVE - specifies that the file requester is to be opened in save mode.

Flags for `OD_TEXT` objects:

```
TEXTFLAG_RIGHT_JUSTIFY - right justify the text
TEXTFLAG_CENTER - center the text
```

Flags for `OD_AREA` objects:

---

AREAFLAG\_RAISED - a raised rectangle  
 AREAFLAG\_RECESSED - a recessed rectangle  
 AREAFLAG\_THIN - draw rectangle with thin borders  
 AREAFLAG\_ICON - draw an icon drop box  
 AREAFLAG\_ERASE - erase the interior of the rectangle  
 AREAFLAG\_LINE - draw a separator line  
 AREAFLAG\_OPTIM - optimised refreshing when updating  
 AREAFLAG\_TITLE - draw a group box with a title  
 AREAFLAG\_NOFILL - don't fill interior

#### od\_ID

This is the ID of the object. If the object is a gadget, it will also set the gadget ID.

#### od\_TagList

Object-specific taglist. For GadTools gadgets, all the standard GadTools flags apply.

GTCustom\_LocaleLabels - (USHORT \*)

Used with : MX\_KIND, CYCLE\_KIND

This tag points to an array of locale IDs. It allows you to specify the text contents of the gadgets using locale. The array must be terminated with a 0 value. You can use this instead of the GTCY\_Labels or GTMX\_Labels tags.

GTCustom\_Image - (struct Image \*)

Used with : BUTTON\_KIND

Points to an Image structure that defines an image to be displayed within the button.

GTCustom\_Callback - (void \_\_asm (\*) (register \_\_a1 struct TagItem \*, register \_\_a2 struct Window \*))

Used with : All types

This tag allows you to specify the address of a callback function that is called when the object is added to the window via

AddObjectList()

. The callback function is passed both the window pointer and a pointer to the tag. The function can modify both the ti\_Tag and ti\_Data values of the tag, and when it returns the Tag will be re-evaluated with the new contents.

GTCustom\_LayoutRel - USHORT

Used with : All types

Lets you position objects relative to another. The `ti_Data` field contains the ID of an object that \*has previously been added\* (eg is before this one in the `ObjectDef` array). The new object will be positioned relative to the specified object; from the new object's point of view, coordinate 0,0 is the top-left corner of the relative object. You can use `POS_CENTER` and the other magic position values with this tag. For example, to position an object in the center of another one, you would use the `GTCustom_LayoutRel` tag, and set `char_dims.Left` and `char_dims.Top` for the new object to `POS_CENTER`.

If the window is resizable, you must supply the `GTCustom_CopyTags` tag as well.

`GTCustom_LayoutPos` - USHORT

Used with : All types

Lets you position objects based on the position of another object. The `ti_Data` field contains the ID of an object that must have been previously added. This tag is used in conjunction with the following object flags:

`POSFLAG_ADJUST_POS_X` - X-position is relative to X-position of other object

`POSFLAG_ADJUST_POS_Y` - Y-position is relative to Y-position of other object

`POSFLAG_ALIGN_X` - Align X-position with other object

`POSFLAG_ALIGN_Y` - Align Y-position with other object

If the window is resizable, you must supply the `GTCustom_CopyTags` tag when you supply this tag.

`GTCustom_Control` - USHORT

Used with : `CHECKBOX_KIND`, `FILE_BUTTON_KIND`, `DIR_BUTTON_KIND`, `FONT_BUTTON_KIND`

Specifies another gadget that this gadget controls. The `ti_Data` field contains the ID of the other gadget. For `CHECKBOX_KIND` gadgets, the other gadget will be disabled when the checkbox is deselected, and enabled when it is selected. For the other types of gadget, the other gadget MUST be a `STRING_KIND`, into which will go the pathname that was selected by the file requester.

For `FONT_BUTTON_KIND` gadgets, the font name is copied into the `STRING_KIND` gadget specified with this tag, and the font size is copied into an `INTEGER_KIND` gadget with the control ID + 1.

---

GTCustom\_TextAttr - struct TextAttr \*

Used with : All gadgets

Lets you specify the font that will be used for a specific gadget.

GTCustom\_MinMax - ULONG

Used with : SLIDER\_KIND, INTEGER\_KIND, SCROLLER\_KIND

Allows you to specify the minimum and maximum values of a gadget. The ULONG contains the maximum value in the upper 16 bits and the minimum value in the lower 16 bits. For SLIDER\_KIND and SCROLLER\_KIND, you can also use the GadTools equivalent tags.

GTCustom\_ThinBorders - BOOL

Used with : BUTTON\_KIND, CHECKBOX\_KIND, FRAME\_KIND, NUMBER\_KIND, TEXT\_KIND, PALETTE\_KIND, FILE\_BUTTON\_KIND, DIR\_BUTTON\_KIND and FONT\_BUTTON\_KIND

This tag can be used instead of the BUTTONFLAG\_THIN\_BORDERS flag. This tag also allows you to control thin borders when accessing the Opus BOOPSI gadgets directly.

GTCustom\_Borderless - BOOL

Used with : BUTTON\_KIND, CHECKBOX\_KIND, FRAME\_KIND, NUMBER\_KIND, TEXT\_KIND, PALETTE\_KIND, FILE\_BUTTON\_KIND, DIR\_BUTTON\_KIND and FONT\_BUTTON\_KIND

If set to TRUE, this causes the gadget to be rendered without a border.

GTCustom\_LocaleKey - ULONG

Used with : All gadgets

This tag takes a locale ID, and uses the corresponding string to set the key equivalent for this gadget. The string is searched for the first underscore character, and the character immediately after the underscore is used as the key equivalent.

GTCustom\_NoSelectNext - BOOL

Used with : STRING\_KIND, INTEGER\_KIND

Ordinarily, pressing return in a string field causes the cursor to move to the next field automatically. If you specify TRUE for this tag, pressing return will simply deactivate the current field.

---

GTCustom\_PathFilter - BOOL

Used with : STRING\_KIND

If you specify TRUE for this tag, the string field will automatically filter the / and : path characters out.

GTCustom\_Secure - BOOL

Used with : STRING\_KIND

If you specify TRUE for this tag, the string field will operate in secure "password" mode.

GTCustom\_History - Att\_List \*

Used with : STRING\_KIND, INTEGER\_KIND

Lets you specify a history list for the gadget. See the docs for the EH\_History tag under

    GetEditHook()  
    for more information.

GTCustom\_CopyTags - BOOL

Used with : All types

If you specify TRUE for this tag, the supplied tag list will be copied when the object is created. You need to specify this tag in conjunction with other tags, depending on the situation.

GTCustom\_FontPens - ULONG \*

Used with : FONT\_BUTTON\_KIND

The ti\_Data field must point to a ULONG that will be used to store the front pen, back pen and draw mode result from the font requester. The data is stored with FgPen in the lowest byte, BgPen in the second byte and DrawMode in the third byte. The most significant byte is not used. You must supply the GTCustom\_CopyTags tag if you use this tag.

GTCustom\_FontPenCount - short

Used with : FONT\_BUTTON\_KIND (only under ASL v39.9)

This allows you to specify the number of pens displayed in the font requester. It is used in conjunction with the GTCustom\_FontPenTable tag, and the GTCustom\_CopyTags tag must also be supplied.

---



GTCustom\_FontPenTable - UBYTE \*

Used with : FONT\_BUTTON\_KIND (only under ASL v39.9)

This is used with GTCustom\_FontPenCount. The ti\_Data field points to a UBYTE array of pen numbers to display in the font requester. You must also specify the GTCustom\_CopyTags tag.

GTCustom\_Bold - BOOL

Used with : BUTTON\_KIND, CHECKBOX\_KIND, FRAME\_KIND, NUMBER\_KIND, TEXT\_KIND, PALETTE\_KIND, FILE\_BUTTON\_KIND, DIR\_BUTTON\_KIND and FONT\_BUTTON\_KIND

If set to TRUE, this tag causes the label for the button to be rendered in bold. Use GTCustom\_Style for greater control.

GTCustom\_NoGhost - BOOL

Used with : BUTTON\_KIND, CHECKBOX\_KIND, FRAME\_KIND, NUMBER\_KIND, TEXT\_KIND, PALETTE\_KIND, FILE\_BUTTON\_KIND, DIR\_BUTTON\_KIND and FONT\_BUTTON\_KIND

If set to TRUE, when the button is disabled its image will not be ghosted.

GTCustom\_Style - ULONG

Used with : BUTTON\_KIND, CHECKBOX\_KIND, FRAME\_KIND, NUMBER\_KIND, TEXT\_KIND, PALETTE\_KIND, FILE\_BUTTON\_KIND, DIR\_BUTTON\_KIND and FONT\_BUTTON\_KIND

This tag allows you to control the text style used to render the button label. Valid flags for the style are FSF\_BOLD and FSF\_ITALIC.

GTCustom\_FrameFlags - ULONG

Used with : FRAME\_KIND

This tag lets you specify the type of frame rendered for the frame gadget. Currently the only value valid is AREAFLAG\_RECESSED, to specify a recessed frame.

GTCustom\_ChangeSigTask - struct Task \*

Used with : STRING\_KIND, INTEGER\_KIND

This tag lets you specify a Task that is to be signalled whenever the contents of the string gadget change.

---

GTCustom\_ChangeSigBit - short

Used with : STRING\_KIND, INTEGER\_KIND

This tag lets you specify the bit that a task is signalled with whenever the contents of the string gadget change.

GTCustom\_Justify - short

Used with : TEXT\_KIND, NUMBER\_KIND

This tag lets you specify the justification of the text displayed in the gadget. Valid values are JUSTIFY\_CENTER (the default), JUSTIFY\_LEFT and JUSTIFY\_RIGHT.

In addition to these tags, OPUS\_LISTVIEW\_KIND gadgets (via the Opus BOOPSI listview class) supports several additional tags. See the section on the listview class for information on these.

The last member in an array of ObjectDef structures must have a od\_Type of OD\_END set.

3. The layout routines can only add gadgets to a special sort of window; one that has been created with the

```
OpenConfigWindow()
 call. Two structures
```

are needed to open one of these windows. The first, a ConfigWindow structure, specifies the position and size of the new window. It has cw\_CharDims and cw\_FineDims fields, analogous to the fields in ObjectDef structures.

The second is a NewConfigWindow structure. This is initialised as follows:

nw\_Parent

This field points to the parent of the Window. The parent can be either another window (the default) or a screen (if the WINDOWF\_SCREEN\_PARENT flag is set).

nw\_Dims

This points to the ConfigWindow structure used to define the size of the new window.

nw\_Title

This points to a title string for the new window.

nw\_Locale

This points to a valid DOpusLocale structure for the window. All ObjectDefs that use a locale ID will obtain their strings via this

structure.

nw\_Port

Allows you to specify a message port to use; you should set this to NULL.

nw\_Flags

Control flags. Valid values are:

```

WINDOW_SCREEN_PARENT - nw_Parent points to a Screen
WINDOW_NO_CLOSE - no close gadget on the window
WINDOW_NO_BORDER - borderless window
WINDOW_SIMPLE - simplerefresh window
WINDOW_AUTO_REFRESH - use with WINDOW_SIMPLE
WINDOW_AUTO_KEYS - handle keypresses automatically
WINDOW_REQ_FILL - backfill the window with stipple pattern
WINDOW_NO_ACTIVATE - don't activate the window on open
WINDOW_VISITOR - open as a visitor window
WINDOW_SIZE_RIGHT - size gadget in right border
WINDOW_SIZE_BOTTOM - size gadget in bottom border
WINDOW_ICONIFY - iconify gadget in title bar

```

nw\_Font

Allows you to specify a font to use for the window. If NULL, the current screen font will be used.

## 1.136 AddObjectList()

NAME

AddObjectList - add a list of objects to a window

SYNOPSIS

```
AddObjectList(window, objects)
```

```
 A0 A1
```

```
ObjectList *AddObjectList(struct Window *, ObjectDef *);
```

FUNCTION

This function adds a list of objects to a window. The window must have previously been opened with the

```
 OpenConfigWindow()
 call.
```

INPUTS

window - window to add the objects to  
objects - array of objects to add

RESULT

Returns a pointer to the list of created objects, or NULL for failure.

#### NOTES

You can add multiple object lists to a window with multiple calls to `AddObjectList()`. These lists are chained together, and as long as you pass the address of the FIRST object list to any functions that search an object list, all chained lists will be searched automatically.

#### SEE ALSO

```
OpenConfigWindow()
,
SetGadgetValue()
```

## 1.137 AddWindowMenus()

#### NAME

`AddWindowMenus` - add menus to a window

#### SYNOPSIS

```
AddWindowMenus(window, menus)
 A0 A1
```

```
void AddWindowMenus(struct Window *, MenuData *);
```

#### FUNCTION

This function makes it easy to add menus to a window opened via

```
OpenConfigWindow()
. Even if you don't use the
AddObjectList()
routine
```

to add gadgets to the window, you can still use this call to add menus.

`AddWindowMenus()` takes an array of `MenuData` structures, and constructs and initialises the Intuition Menu structures automatically. The `MenuData` structures are initialised as follows:

```
md_Type - entry type; NM_TITLE, NM_ITEM or NM_SUB. The array
 must be terminated by an NM_END item.

md_ID - the ID value for the menu item.

md_Name - the name of the menu item. This can either be a
 locale ID, or, if the MENUFLAG_TEXT_STRING flag is
 set, a pointer to a real string.

md_Flags - control flags.
```

The `md_Flags` field supports the standard Intuition and GadTools menu flags as well as several custom flags:

MENUFLAG\_TEXT\_STRING - md\_Name is a real string  
 MENUFLAG\_COMM\_SEQ - the menu will be given a command sequence  
 (Right-Amiga + a key)  
 MENUFLAG\_AUTO\_MUTEX - automatic mutual exclusion

If MENUFLAG\_COMM\_SEQ is specified, the key used for the command sequence is normally the first character of the menu name. However, you can specify a character to use instead of this, by setting the MENUFLAG\_USE\_SEQ flag, and using the MENUFLAG\_MAKE\_SEQ() macro. For example, to specify a command sequence of Right Amiga and A, you would use :

```
MENUFLAG_COMM_SEQ|MENUFLAG_USE_SEQ|MENUFLAG_MAKE_SEQ('A')
```

The automatic mutual exclusion works on all items at the current level that have the CHECKIT flag set.

#### INPUTS

window - window to add menus to  
 menus - array of MenuData structures, terminated with NM\_END

#### RESULT

The menus are added to the window. You can check if this operation failed (through lack of memory) by examining the window->MenuStrip pointer; if NULL, the window has no menus.

#### SEE ALSO

```
FindMenuItem()
,
FreeWindowMenus()
```

## 1.138 BoundsCheckGadget()

#### NAME

BoundsCheckGadget - bounds check an integer gadget

#### SYNOPSIS

```
BoundsCheckGadget(list, id, min, max)
 A0 D0 D1 D2
```

```
long BoundsCheckGadget(ObjectList *, ULONG, long, long);
```

#### FUNCTION

This routine tests the value of an integer gadget against the supplied minimum and maximum, adjusts and refreshes it if it is invalid, and returns the new value.

#### INPUTS

list - ObjectList containing the gadget  
 id - gadget ID  
 min - minimum value  
 max - maximum value

## RESULT

Returns the new value of the gadget.

## 1.139 BuildMenuStrip()

## NAME

BuildMenuStrip - build a MenuStrip easily

## SYNOPSIS

BuildMenuStrip(menus, locale)

A0 A1

```
struct Menu *BuildMenuStrip(MenuData *, struct DOpusLocale *);
```

## FUNCTION

This routine takes the supplied MenuData array, and returns an initialised menu strip. This menu strip can then be layed out using the LayoutMenus() function in gadtools.library, and added to any window.

## INPUTS

menus - array of MenuData structures

locale - locale to use for text strings

## RESULT

Returns a pointer to the head of the menu strip.

## NOTES

The menus returned by this function can be used on any window. If your window was opened with

```
OpenConfigWindow()
, you should use the
```

```
AddWindowMenus()
function instead of this one.
```

The returned menu strip can be freed with a call to FreeMenus() in gadtools.library.

See the instructions for

```
AddWindowMenus()
for information about
```

initialising the MenuData structures.

## SEE ALSO

```
AddWindowMenus()
, gadtools.library/LayoutMenusA(),
gadtools.library/FreeMenus()
```

## 1.140 CheckObjectArea()

### NAME

CheckObjectArea - check if a point is within an object's area

### SYNOPSIS

```
CheckObjectArea(object, x, y)
 A0 D0 D1
```

```
BOOL CheckObjectArea(GL_Object *, long, long);
```

### FUNCTION

This routine tests if the coordinate is within the "select" area of the object.

### INPUTS

object - object to test  
x - x coordinate  
y - y coordinate

### RESULT

Returns TRUE if the point falls within the object.

## 1.141 ClearWindowBusy()

### NAME

ClearWindowBusy - make a window unbusy

### SYNOPSIS

```
ClearWindowBusy(window)
 A0
```

```
void ClearWindowBusy(struct Window *);
```

### FUNCTION

This routine undoes the effect of a `SetWindowBusy()` call. The mouse pointer of the window is returned to normal, and input is unblocked.

### INPUTS

window - the window in question

### SEE ALSO

`SetWindowBusy()`

## 1.142 CloseConfigWindow()

---

## NAME

CloseConfigWindow - close a window

## SYNOPSIS

CloseConfigWindow(window)

A0

```
void CloseConfigWindow(struct Window *);
```

## FUNCTION

This function closes a window that was opened with a call to

OpenConfigWindow()

. All memory associated with the window, including object lists, menus and memory allocated from the window's memory pool is freed by this call.

## INPUTS

window - window to close. MUST have been opened with

OpenConfigWindow()

## RESULT

The window is closed.

## SEE ALSO

OpenConfigWindow()

## 1.143 DisableObject()

## NAME

DisableObject - disable/enable an object

## SYNOPSIS

DisableObject(list, id, state)

A0 D0 D1

```
void DisableObject(ObjectList *, ULONG, BOOL);
```

## FUNCTION

This routine disables or enables an object. Currently, only gadgets support being disabled.

## INPUTS

list - ObjectList containing object

id - object ID

state - TRUE for disable, FALSE for enable

## RESULT

The object is disabled or enabled.

---



## 1.144 DisplayObject()

### NAME

DisplayObject - redisplay an object

### SYNOPSIS

```
DisplayObject(window, object, fpen, bpen, text)
 A0 A1 D0 D1 A2
```

```
void DisplayObject(struct Window *, GL_Object *, long, long, char *);
```

### FUNCTION

This routine lets you change and refresh an object. Currently, it is only used for OD\_TEXT and OD\_AREA objects.

### INPUTS

window - window containing the object  
 object - object to display  
 fpen - new foreground pen, -1 for no change  
 bpen - new background pen, -1 for no change  
 text - new text

### RESULT

The display is refreshed immediately.

## 1.145 EndRefreshConfigWindow()

### NAME

EndRefreshConfigWindow - finish refreshing a config window

### SYNOPSIS

```
EndRefreshConfigWindow(window)
 A0
```

```
void EndRefreshConfigWindow(struct Window *);
```

### FUNCTION

This is analogous to calling GT\_EndRefresh(window,TRUE) on the window. It finishes and cleans up the refresh process begun with

```
StartRefreshConfigWindow()
```

### INPUTS

window - window to end refresh of

### NOTES

If you are using a smart refresh window, or have set the WINDOW\_AUTO\_REFRESH flag, you will not need to call this function.

### SEE ALSO

```
StartRefreshConfigWindow()
```

```
, gadtools.library/GT_EndRefresh()
```

## 1.146 FindMenuItem()

NAME

FindMenuItem - find a menu item by ID

SYNOPSIS

```
FindMenuItem(menu, id)
 A0 D0
```

```
struct MenuItem *FindMenuItem(struct Menu *, USHORT);
```

FUNCTION

Traverses the menu list from the supplied pointer, searching for a menu with the given ID. This ID is extracted using the GTMENUITEM\_USERDATA() macro. If you constructed the menu item with

```
 AddWindowMenus()
```

or

```
 BuildMenuStrip()
```

, the ID was supplied

in the md\_ID field of the MenuData structure.

INPUTS

menu - Menu to start search

id - ID to search for

RESULT

Returns a pointer to the MenuItem or NULL if not found.

SEE ALSO

```
 AddWindowMenus()
```

,

```
 BuildMenuStrip()
```

## 1.147 FreeObjectList()

NAME

FreeObjectList - free a list of objects

SYNOPSIS

```
FreeObjectList(list)
 A0
```

```
void FreeObjectList(ObjectList *);
```

FUNCTION

This routine frees the supplied list of objects. If there are any

gadgets in the list, they are automatically removed from the window before being freed.

Ordinarily you do not need to call this function; any existing objects are freed automatically when you close the window with

```
CloseConfigWindow()
. However, this function together with
```

```
AddObjectList()
allows you to add and remove objects (gadgets) from
the window dynamically. This is something that is not possible under
GadTools.
```

#### INPUTS

list - object list to free

#### RESULT

The list is delinked and all its objects are freed. If you are freeing just one list from a window, you will need to refresh the display after you have freed the objects.

#### SEE ALSO

```
AddObjectList()
,
CloseConfigWindow()
```

## 1.148 FreeWindowMenus()

#### NAME

FreeWindowMenus - free menus from a window

#### SYNOPSIS

```
FreeWindowMenus(window)
A0
```

```
void FreeWindowMenus(struct Window *);
```

#### FUNCTION

Frees the menus that were attached with a call to

```
AddWindowMenus()
```

Normally you do not need to call this function, as menus are automatically freed when you close the window with

```
CloseConfigWindow()
```

#### INPUTS

window - window to free menus for

#### RESULT

The menus are removed from the window and freed.

---

SEE ALSO

AddWindowMenus()

## 1.149 GetGadgetValue()

NAME

GetGadgetValue - get the value of a gadget

SYNOPSIS

```
GetGadgetValue(list, id)
 A0 A1
```

```
long GetGadgetValue(ObjectList *, USHORT);
```

FUNCTION

This returns the current value of the gadget, specified by gadget ID. The supplied list is searched for the gadget.

INPUTS

list - ObjectList containing the gadget  
id - gadget ID

RESULT

Returns the current value of the gadget. The contents of the return value are dependant on the type of gadget:

  BUTTON\_KIND       - if the BUTTONFLAG\_TOGGLE\_SELECT flag was set,  
                    returns TRUE or FALSE to indicate the state of  
                    the gadget.

  MX\_KIND

  CYCLE\_KIND

  OPUS\_LISTVIEW\_KIND

  LISTVIEW\_KIND

  SLIDER\_KIND

  SCROLLER\_KIND

  PALETTE\_KIND     - returns the current selection or value

  CHECKBOX\_KIND   - returns TRUE or FALSE to indicate the state  
                    of the gadget.

  INTEGER\_KIND     - returns the integer value of the gadget

  STRING\_KIND      - returns a pointer to the string contents. This  
                    pointer is READ ONLY!

SEE ALSO

SetGadgetValue()

---

## 1.150 GetObject()

### NAME

GetObject - get an object by ID from a list

### SYNOPSIS

```
GetObject(list, id)
 A0 D0
```

```
GL_Object *GetObject(ObjectList *, ULONG);
```

### FUNCTION

Searches the supplied object list (and any chained lists) for the object with the given ID value.

### INPUTS

list - ObjectList to search  
id - ID to search for

### RESULT

Returns a pointer to the object or NULL if not found.

SEE ALSO

AddObjectList()

## 1.151 GetObjectRect()

### NAME

GetObjectRect - get an object's rectangle

### SYNOPSIS

```
GetObjectRect(list, id, rect)
 A0 D0 A1
```

```
BOOL GetObjectRect(ObjectList *, ULONG, struct Rectangle *);
```

### FUNCTION

Searches for the object, and if found, copies the coordinates of the object's display rectangle into the supplied structure.

### INPUTS

list - ObjectList containing the object  
id - ID of the object  
rect - Rectangle structure for result

### RESULT

Returns FALSE if the object could not be found.

---

## 1.152 GetWindowAppPort()

NAME

GetWindowAppPort - get a window's application port

SYNOPSIS

```
GetWindowAppPort(window)
 A0
```

```
struct MsgPort *GetWindowAppPort(struct Window *);
```

FUNCTION

If a window has registered itself with a call to `SetWindowID()`, this function will return the address of its application message port.

INPUTS

window - window in question

RESULT

Returns a pointer to the message port, or NULL if the window wasn't registered.

SEE ALSO

```
SetWindowID()
,
GetWindowID()
```

## 1.153 GetWindowID()

NAME

GetWindowID - get a window's ID code

SYNOPSIS

```
GetWindowID(window)
 A0
```

```
ULONG GetWindowID(struct Window *);
```

FUNCTION

If a window has been registered with `SetWindowID()`, this function returns the ID code.

INPUTS

window - window in question

RESULT

Returns the ID code of the window if it is registered. Returns `WINDOW_UNKNOWN` if not registered. Returns `WINDOW_UNDEFINED` if the

Window is registered, but does not have an ID or application port.

SEE ALSO

```
SetWindowID()
,
GetWindowAppPort()
```

## 1.154 GetWindowMsg()

NAME

GetWindowMsg - get IntuiMessage for a window

SYNOPSIS

```
GetWindowMsg(port)
 A0
```

```
struct IntuiMessage *GetWindowMsg(struct MsgPort *);
```

FUNCTION

This routine is analogous to the GT\_GetIMsg call under GadTools. It searches the supplied port for an IntuiMessage, performs pre-processing on the message and returns the result to you. It is highly recommended that you use this instead of a normal call to GetMsg() when using the layout routines. In particular, auto refreshing of simplerefresh windows, resizing, key equivalents and proper gadget processing will all be affected if you do not call this function.

INPUTS

port - port to search for messages (window->UserPort)

RESULT

Returns a pointer to the message, or NULL if there was none. You must call

```
ReplyWindowMsg()
 to reply to messages from this function.
```

SEE ALSO

```
ReplyWindowMsg()
```

## 1.155 LayoutResize()

NAME

LayoutResize - resize a window

SYNOPSIS

```
LayoutResize(window)
 A0
```

```
void LayoutResize(struct Window *);
```

#### FUNCTION

This routine is called to handle refreshing of a window when it is resized by the user. If you call

```
 GetWindowMsg()
 in your event loop,
```

this is called automatically for you.

#### INPUTS

window - window that has been resized

#### RESULT

All objects in the window are resized (if they were defined with proportional or relative sizes) and refreshed.

## 1.156 OpenConfigWindow()

#### NAME

OpenConfigWindow - open a window

#### SYNOPSIS

```
OpenConfigWindow(newwin)
```

```
 A0
```

```
struct Window *OpenConfigWindow(NewConfigWindow *);
```

#### FUNCTION

This routine opens the window defined by the supplied NewConfigWindow structure.

#### INPUTS

newwin - initialised NewConfigWindow structure

#### RESULT

Returns a pointer to the Window. This is a normal Intuition window in most respects, but its UserData field points to a structure of additional information. You MUST NOT modify the UserData field of such a window.

#### SEE ALSO

```
 AddObjectList()
```

```
 ,
 CloseConfigWindow()
```

## 1.157 ReplyWindowMsg()



## NAME

ReplyWindowMsg - reply to a message

## SYNOPSIS

ReplyWindowMsg(msg)

A0

```
void ReplyWindowMsg(struct IntuiMessage *);
```

## FUNCTION

Call this function to reply to a message you received via

GetWindowMsg()

.

## INPUTS

msg - message to reply to

## SEE ALSO

GetWindowMsg()

## 1.158 SetConfigWindowLimits()

## NAME

SetConfigWindowLimits - set size limits for a window

## SYNOPSIS

SetConfigWindowLimits(window, min, max)

A0 A1 A2

```
void SetConfigWindowLimits(struct Window *, ConfigWindow *,
 ConfigWindow *);
```

## FUNCTION

Sets the sizing limits of the supplied window. The minimum and maximum dimensions are specified with two ConfigWindow structures.

## INPUTS

window - window to set limits for

min - minimum dimensions

max - maximum dimensions

## SEE ALSO

OpenConfigWindow()

## 1.159 SetGadgetChoices()

## NAME

SetGadgetChoices - set the "choices" for a gadget

## SYNOPSIS

```
SetGadgetChoices(list, id, choices)
 A0 D0 A1
```

```
void SetGadgetChoices(ObjectList *, ULONG, APTR);
```

## FUNCTION

This routine sets the choices for a gadget. It operates differently for different types of gadgets:

## SCROLLER\_KIND

Sets the maximum value of the gadget, and adjusts the current value if it exceeds this limit.

## SLIDER\_KIND

Sets the minimum (lower 16 bits) and maximum (upper 16 bits) values of the gadget, and adjusts the current value if it exceeds either of these limits.

## LISTVIEW\_KIND/OPUS\_LISTVIEW\_KIND

Sets the list contents pointer. This is either a struct List \* or an Att\_List \*. If you set the value to NULL, the current list will be detached from the gadget and the list will be cleared. If you set the value to -1, the list will be detached but the gadget display will not be cleared.

## CYCLE\_KIND

Sets the cycle gadget contents. This points to a char \* array, or NULL if you want the gadget to be empty. The array must be null-terminated.

## INPUTS

list - ObjectList containing the gadget  
 id - gadget ID  
 choices - new choices for the gadget

## RESULT

The display is updated immediately, if appropriate.

## 1.160 SetGadgetValue()

## NAME

SetGadgetValue - set the value of a gadget

## SYNOPSIS

```
SetGadgetValue(list, id, value)
```

A0 D0 D1

```
void SetGadgetValue(ObjectList *, USHORT, ULONG);
```

#### FUNCTION

Sets the value of a gadget. See the instructions for

GetGadgetValue()

for a list of the gadgets a value can be set for. Note that in ←  
the

case of a STRING\_KIND gadget, the string that you supply is copied,  
and does not need to remain valid once you have set it.

#### INPUTS

list - ObjectList containing the gadget

id - gadget ID

value - new value for the gadget

#### RESULT

The display is updated immediately.

#### SEE ALSO

GetGadgetValue()

## 1.161 SetWindowBusy()

#### NAME

SetWindowBusy - make a window busy

#### SYNOPSIS

```
SetWindowBusy(window)
```

A0

```
void SetWindowBusy(struct Window *);
```

#### FUNCTION

Makes the supplied window busy. The mouse pointer is changed to the  
system busy pointer, and all gadget input to the window is blocked.  
You must call

ClearWindowBusy()

to reverse this state.

#### INPUTS

window - window to make busy

#### RESULT

The window goes busy.

#### NOTES

You can only call this routine on a window opened with

OpenConfigWindow()

.

SEE ALSO

```
OpenConfigWindow()
,
ClearWindowBusy()
```

## 1.162 SetWindowID()

NAME

SetWindowID - register a window's ID

SYNOPSIS

```
SetWindowID(window, idptr, id, port)
 A0 A1 D0 A2
```

```
void SetWindowID(struct Window *, WindowID *, ULONG, struct MsgPort *);
```

FUNCTION

This routine "registers" a window, giving it an ID value and associating a message port with it.

This is invaluable in a drag and drop situation, when you want to determine whether a particular window supports a particular type of drop operation.

The function takes a pointer to a WindowID structure, which is stored in the UserData field of the Window. You therefore lose the use of the UserData field, but you can easily recover it by embedding a WindowID structure in a larger structure.

It also takes an ID value, and a pointer to a message port. These values are retrievable with the

```
GetWindowID()
and
GetWindowAppPort()
calls.
```

All windows opened with

```
OpenConfigWindow()
have an ID associated
```

with them automatically; by default, the ID is set to WINDOW\_UNDEFINED. If you wish to change it, you can use the SET\_WINDOW\_ID() macro.

INPUTS

```
window - window to register
idptr - pointer to WindowID structure. This MUST remain valid for the
 life of the window
id - ID value for the window. You should set the WINDOW_USER bit
 of any IDs you define.
port - pointer to message port. This does not actually have to be
 a message port; it can be any 32 bit value.
```

RESULT

The window association is made.

SEE ALSO

```
GetWindowID()
,
GetWindowAppPort()
```

## 1.163 StartRefreshConfigWindow()

NAME

StartRefreshConfigWindow - begin refreshing a window

SYNOPSIS

```
StartRefreshConfigWindow(window, finish)
 A0 D0
```

```
void StartRefreshConfigWindow(struct Window *, long);
```

FUNCTION

This routine begins refresh of a simplerefresh window that was opened with

```
OpenConfigWindow()
```

. If you specify the WINDOW\_AUTO\_REFRESH flag, you will never need to call this function.

INPUTS

window - window to begin refreshing

finish - if set to TRUE, the refresh is ended by this function too. Use this if you don't want to do any rendering of your own for the refresh (but if that's the case, why not just use WINDOW\_AUTO\_REFRESH?)

RESULT

Refreshing is started (and optionally finished). All the objects in the window will be refreshed.

SEE ALSO

```
EndRefreshConfigWindow()
, intuition.library/BeginRefresh()
```

## 1.164 List\_Routines

```
List_Routines
```

```
AddSorted()
```

```
Att_ChangeNodeName()
```

```
Att_FindNode ()
Att_FindNodeData ()
Att_FindNodeNumber ()
Att_NewList ()
Att_NewNode ()
Att_NodeCount ()
Att_NodeDataNumber ()
Att_NodeName ()
Att_NodeNumber ()
Att_PosNode ()
Att_RemList ()
Att_RemNode ()
FindNameI ()
GetSemaphore ()
InitListLock ()
IsListLockEmpty ()
LockAttList ()
SwapListNodes ()
UnlockAttList ()
```

## 1.165 AddSorted()

### NAME

AddSorted - add a node to a list in alphabetical order

### SYNOPSIS

```
AddSorted(list, node)
```

```
 A0 A1
```

```
void AddSorted(struct List *, struct Node *);
```

### FUNCTION

This function adds a Node to a List, in alphabetical order based on ln\_Name.

---

## INPUTS

list - List to add Node to  
node - Node to add

## RESULT

The node is inserted in the list in its alphabetical position.  
ALL the nodes in the list must have a valid ln\_Name, or this routine will cause enforcer hits.

## NOTES

This routine uses a simple insertion sort based on strcmpi(). As such, it is neither terrible efficient, or locale-sensitive.

## SEE ALSO

exec.library/Insert()

## 1.166 Att\_ChangeNodeName()

## NAME

Att\_ChangeNodeName - change the name of a node

## SYNOPSIS

Att\_ChangeNodeName(node, name)  
A0 A1

```
void Att_ChangeNodeName(Att_Node *, char *);
```

## FUNCTION

Frees the old name of the node and copies the new name.

## INPUTS

node - Att\_Node to change name for  
name - new name of the node

## RESULT

The new name is copied and installed.

## SEE ALSO

Att\_NewNode()

## 1.167 Att\_FindNode()

## NAME

Att\_FindNode - find a node by number

## SYNOPSIS

Att\_FindNode(list, number)  
A0 D0

```
Att_Node *Att_FindNode(Att_List *, long);
```

---

## FUNCTION

This routine finds the specified node in the list and returns a pointer to it.

## INPUTS

list - list to search  
 number - cardinal number of the node to find

## RESULT

Returns the specified Att\_Node or NULL if not found.

## NOTES

This routine can also work on normal Lists with proper type-casting.

## SEE ALSO

```
Att_NewNode()
,
Att_NodeName()
```

## 1.168 Att\_FindNodeData()

## NAME

Att\_FindNodeData - find a node by its data

## SYNOPSIS

```
Att_FindNodeData(list, data)
 A0 D0
```

```
Att_Node *Att_FindNodeData(Att_List *, ULONG);
```

## FUNCTION

This function searches the list for a node with data that matches the supplied ULONG value (the data is specified with the

```
Att_NewNode()
function).
```

## INPUTS

list - list to search  
 data - data to look for

## RESULT

Returns the Att\_Node if found, otherwise NULL.

## SEE ALSO

```
Att_NewNode()
```

## 1.169 Att\_FindNodeNumber()



## NAME

Att\_FindNodeNumber - find cardinal number of a node

## SYNOPSIS

Att\_FindNodeNumber(list, node)

A0 A1

```
long Att_FindNodeNumber(Att_List *, Att_Node *);
```

## FUNCTION

This routine searches the list for the specified node, and returns the offset from the beginning of the list.

## INPUTS

list - list to search

node - node to look for

## RESULT

Returns the cardinal number of the node or -1 if not found.

## NOTES

This routine can also work on normal Lists with proper type-casting.

## SEE ALSO

Att\_NewNode()

## 1.170 Att\_NewList()

## NAME

Att\_NewList - create a new list

## SYNOPSIS

Att\_NewList(flags)

D0

```
Att_List *Att_NewList(ULONG);
```

## FUNCTION

Creates a new Att\_List structure, which you use in conjunction with Att\_Nodes. These functions provide a convenient way of dynamically allocating members of lists and performing searches and sorts on them.

## INPUTS

flags - control flags. Currently valid values are :

LISTF\_LOCK - list is to be shared and requires locking

LISTF\_POOL - use memory pooling for nodes and names

If you specify LISTF\_LOCK, a semaphore will be initialised for this list. Any of the list management functions in the dopus5.library will lock the semaphore before accessing the list.

If you specify `LISTF_POOL`, a small memory pool will be used to allocate list nodes and node names, which can result in greater speed and less memory fragmentation.

#### RESULT

Returns pointer to an `Att_List` structure or `NULL` for failure.

#### SEE ALSO

```
Att_RemList()
, exec.library/NewList()
```

## 1.171 Att\_NewNode()

### NAME

`Att_NewNode` - add a new node to a list

### SYNOPSIS

```
Att_NewNode(list, name, data, flags)
 A0 A1 D0 D1
```

```
Att_Node *Att_NewNode(Att_List *, char *, ULONG, ULONG);
```

### FUNCTION

This routine allocates a new node and adds it to the specified list. It can also allocate and copy a name for the node, and store user-defined data with it. The new node can be added to the list sorted in several ways.

### INPUTS

`list` - list to add node to  
`name` - name for the node (will be copied)  
`data` - user-defined data for the node  
`flags` - control flags. Currently valid flags are:

`ADDNODEF_SORT` - sort names alphabetically

`ADDNODEF_EXCLUSIVE` - name must be exclusive; if a node already exists with this name, the call will fail. Only works in conjunction with `ADDNODEF_SORT`.

`ADDNODEF_NUMSORT` - the node name is taken to be an ascii string containing a number, and the sort is based on numerical order rather than ascii order (so that 10 would come after 1 rather than before).

`ADDNODEF_PRI` - sort is based on priority. If you specify this flag, the 'data' parameter is taken to be the node's priority.

If no sorting flags are specified, the node is added to the end of the list.

#### RESULT

If successful, the new Att\_Node is returned.

#### SEE ALSO

```
Att_NewList()
,
Att_RemNode()
```

## 1.172 Att\_NodeCount()

#### NAME

Att\_NodeCount - count the nodes in a list

#### SYNOPSIS

```
Att_NodeCount(list)
A0
```

```
long Att_NodeCount(Att_List *);
```

#### FUNCTION

Returns the number of nodes in the list.

#### INPUTS

list - list to count

#### NOTES

This routine can also work on normal Lists with proper type-casting.

#### SEE ALSO

```
Att_NewNode()
```

## 1.173 Att\_NodeDataNumber()

#### NAME

Att\_NodeDataNumber - find cardinal number of a node

#### SYNOPSIS

```
Att_NodeDataNumber(list, data)
A0 D0
```

```
long Att_NodeDataNumber(Att_List *, ULONG);
```

#### FUNCTION

This routine is similar to

---

```

 Att_FindNodeNumber()
 , except that it takes
a ULONG value and searches the list for a node with that value as
it's 'data' (the data specified in the call to
 Att_NewNode
).

```

#### INPUTS

```

list - list to search
data - node data to look for

```

#### RESULT

Returns the Att\_Node if found, or NULL.

#### SEE ALSO

```

 Att_NewNode()
 ,
 Att_FindNodeNumber()

```

## 1.174 Att\_NodeName()

#### NAME

Att\_NodeName - find a node name by number

#### SYNOPSIS

```

Att_NodeName(list, number)
 A0 D0

```

```

char *Att_NodeName(Att_List *, long);

```

#### FUNCTION

This routine is similar to  
 Att\_FindNode()  
 except that it returns  
a pointer to the node's name rather than the node itself.

#### INPUTS

```

list - list to search
number - cardinal number of the node to find

```

#### RESULT

Returns a pointer to the node's name, or NULL if not found.

#### NOTES

This routine can also work on normal Lists with proper type-casting.

#### SEE ALSO

```

 Att_NewNode()
 ,
 Att_FindNode()

```

## 1.175 Att\_NodeNumber()

NAME

Att\_NodeNumber - find cardinal number of node by name

SYNOPSIS

```
Att_NodeNumber(list, name)
 A0 A1
```

```
long Att_NodeNumber(Att_List *, char *);
```

FUNCTION

This routine is similar to  
     Att\_FindNodeNumber()  
     , except that you  
 specify a name to search for rather than a node pointer.

INPUTS

list - list to search  
 name - name of node to search for

RESULT

Returns the cardinal number of the node or -1 if not found.

NOTES

This routine can also work on normal Lists with proper type-casting.  
 The search is not case-sensitive.

SEE ALSO

```
Att_NewNode()
,
Att_FindNodeNumber()
```

## 1.176 Att\_PosNode()

NAME

Att\_PosNode - reposition an Att\_Node in an Att\_List

SYNOPSIS

```
Att_PosNode(list, node, before)
 A0 A1 A2
```

```
void Att_PosNode(Att_List *, Att_Node *, Att_Node *);
```

FUNCTION

This routine removes an Att\_Node from its current position and  
 re-inserts it in the list in a new position.

INPUTS

list - Att\_List containing node  
 node - Att\_Node to reposition  
 before - Att\_Node to re-insert the node before

## RESULT

The node is inserted in the list before the supplied node.

## SEE ALSO

Att\_NewNode()

## 1.177 Att\_RemList()

## NAME

Att\_RemList - free an entire Att\_List

## SYNOPSIS

```
Att_RemList(list, flags)
 A0 D0
```

```
void Att_RemList(Att_List *, long);
```

## FUNCTION

This function releases all the memory used by an Att\_List, including freeing all of the Att\_Nodes attached to it.

## INPUTS

list - Att\_List to free

flags - control flags. Current flag values are :

REMLISTF\_FREEDATA - If you specify this flag, the data pointers of each of the nodes will be automatically freed with FreeVec(). Therefore, if you use this feature, the data you supply to Att\_NewNode() MUST have been allocated with AllocVec().

REMLISTF\_SAVELIST - If you specify this flag, only the nodes of the list will be freed. The Att\_List itself will be reinitialised, ready for use.

## RESULT

The list nodes and optionally the list itself is freed.

## SEE ALSO

```
Att_NewList()
,
Att_NewNode()
,
Att_RemNode()
```

## 1.178 Att\_RemNode()

NAME

Att\_RemNode - remove a node from a list

SYNOPSIS

```
Att_RemNode(node)
 A0
```

```
void Att_RemNode(Att_Node *);
```

FUNCTION

This function removes the specified node from its list, and frees the name copy and node structure.

INPUTS

node - node to free

RESULT

The node is removed and freed. The node data is NOT freed by this routine.

SEE ALSO

```
Att_NewNode()
,
Att_RemList()
```

## 1.179 FindNameI()

NAME

FindNameI - find a node by name

SYNOPSIS

```
FindNameI(list, name)
 A0 A1
```

```
struct Node *FindNameI(struct List *, char *);
```

FUNCTION

This routine is similar to the exec.library/FindName routine, except that the comparison used in FindNameI() is not case-sensitive.

INPUTS

list - list to search  
name - name to search for

RESULT

Returns pointer to the node if found, otherwise NULL.

SEE ALSO

```
exec.library/FindName()
```

---

## 1.180 GetSemaphore()

### NAME

GetSemaphore - lock a semaphore

### SYNOPSIS

```
GetSemaphore(semaphore, flags, unused)
 A0 D0 A1
```

```
long GetSemaphore(struct SignalSemaphore *, long, APTR);
```

### FUNCTION

This routine locks or attempts to lock the given Semaphore. This routine fixes some bugs that the exec.library Semaphore routines have under some versions of the OS.

### INPUTS

semaphore - Semaphore to lock  
flags - control flags. Valid flags are :

```
SEM_F_SHARED - lock in shared mode
SEM_F_EXCLUSIVE - lock in exclusive mode
SEM_F_ATTEMPT - only attempt to lock
```

unused - must be NULL

### RESULT

Returns TRUE if the Semaphore was successfully locked. If SEM\_F\_ATTEMPT is not specified, this routine will block until the Semaphore is available, and will always return TRUE.

### NOTES

To unlock a Semaphore locked with this function, use the standard exec.library ReleaseSemaphore() call.

### SEE ALSO

```
exec.library/ObtainSemaphore(), exec.library/ObtainSemaphoreShared(),
exec.library/AttemptSemaphore(), exec.library/AttemptSemaphoreShared(),
exec.library/ReleaseSemaphore()
```

## 1.181 InitListLock()

### NAME

InitListLock - initialise a list/lock pair

### SYNOPSIS

```
InitListLock(ll, unused)
 A0 A1
```

```
void InitListLock(struct ListLock *, APTR);
```

### FUNCTION



A ListLock is a convenient structure that ties a List to a Semaphore. This routine initialises both with the one call.

#### INPUTS

ll - ListLock to initialised  
unused - must be NULL

#### RESULT

The List and the Semaphore in the ListLock are initialised.

#### SEE ALSO

exec.library/NewList(), exec.library/InitSemaphore()

## 1.182 IsListLockEmpty()

#### NAME

IsListLockEmpty - see if a list is empty

#### SYNOPSIS

IsListLockEmpty(ll)  
A0

```
BOOL IsListLockEmpty(struct ListLock *);
```

#### FUNCTION

This routine is equivalent to the IsListEmpty() macro, except that it locks the list in shared mode before accessing it.

#### INPUTS

ll - ListLock to test

#### RESULT

Returns TRUE if the list is empty.

#### SEE ALSO

InitListLock()  
, exec.library/IsListEmpty()

## 1.183 LockAttList()

#### NAME

LockAttList - lock an Att\_List

#### SYNOPSIS

LockAttList(list, exclusive)  
A0 D0

```
void LockAttList(Att_List *, BOOL);
```

#### FUNCTION

If an Att\_List was created with the LISTF\_LOCK flag, this routine can be used to lock the list.

#### INPUTS

list - Att\_List to lock  
exclusive - set to TRUE if you want exclusive access

#### RESULT

This routine will block until the list can be locked, and then return. You must call

```
UnlockAttList()
 to unlock the list when you have
finished with it.
```

#### SEE ALSO

```
Att_NewList()
,
UnlockAttList()
```

## 1.184 SwapListNodes()

#### NAME

SwapListNodes - swap two nodes in a list around

#### SYNOPSIS

```
SwapListNodes(list, node1, node2);
 A0 A1 A2
```

```
void SwapListNodes(struct List *, struct Node *, struct Node *);
```

#### FUNCTION

This routine exchanges the positions of two nodes in a list.

#### INPUTS

list - List containing the nodes  
node1 - first node to swap  
node2 - second node to swap

#### RESULT

The nodes' positions will be exchanged.

## 1.185 UnlockAttList()

#### NAME

UnlockAttList - unlock an Att\_List

#### SYNOPSIS

```
UnlockAttList(list)
 A0
```

```
void UnlockAttList (Att_List *);
```

#### FUNCTION

This routine unlocks an Att\_List that was locked with  
 LockAttList()

.

#### INPUTS

list - list to unlock

#### RESULT

The list is unlocked. Calls to  
 LockAttList()  
 are nested, so you must  
 unlock the list as many times as you locked it.

#### SEE ALSO

LockAttList()

## 1.186 Locale\_Routines

Locale Routines

DOpusGetString()

## 1.187 DOpusGetString()

#### NAME

DOpusGetString - get a text string from the locale table

#### SYNOPSIS

```
DOpusGetString(locale, id)
 A0 D0
```

```
STRPTR DOpusGetString(struct DOpusLocale *, long);
```

#### FUNCTION

This routine searches the string table referenced by the supplied  
 DOpusLocale structure for the string matching the supplied ID,  
 and returns a pointer to it.

The DOpusLocale structure must be initialised in the following way :

```
li_LocaleBase - locale.library base address, or NULL
li_Catalog - OpenCatalog() result, or NULL
li_BuiltIn - default string table
```

li\_Locale - current system locale or NULL

If there is no external catalog file, or locale.library is not available, all fields except li\_BuiltIn must be initialised to NULL. li\_BuiltIn MUST point to a table of default strings. This table is in the CatComp block format. The easiest way to initialise this pointer is to have a separate source module to a) include the string table, and b) initialise the pointer. For example,

```
#define CATCOMP_BLOCK
#include "strings.h"

void init_locale_ptr(struct DOpusLocale *locale)
{
locale->li_BuiltIn=(char *)CatCompBlock;
}
```

#### INPUTS

locale - pointer to initialised DOpusLocale structure.  
id - string ID to return.

#### RESULT

Returns a pointer to the requested string. If there is no catalog, or the given string is not in the supplied catalog, the default string is returned. This pointer is READ-ONLY!

You MUST NOT pass invalid string IDs to this routine.

#### SEE ALSO

locale.library/OpenLocale(), locale.library/OpenCatalog(),  
locale.library/GetLocaleStr()

## 1.188 Memory\_Routines

Memory Routines

AllocMemH()

ClearMemHandle()

FreeMemH()

FreeMemHandle()

NewMemHandle()

## 1.189 AllocMemH()

#### NAME

AllocMemH - allocate memory using pooling routines

---

## SYNOPSIS

```
AllocMemH(handle, size)
 A0 D0
```

```
void *AllocMemH(APTR, ULONG);
```

## FUNCTION

This function allows you to allocate a chunk of memory. The type of memory allocated was specified when the memory handle was created. The size of the allocation is tracked automatically (similar to AllocVec).

You can actually use this function with a NULL memory handle - in this case, the function performs much like AllocVec(). This disadvantage to this is that you are unable to specify the type of memory you need (the default is MEMF\_ANY|MEMF\_CLEAR). Memory allocated in this way can obviously not be tracked, and you must

```
FreeMemH()
 each allocation
```

individually.

## INPUTS

```
handle - memory handle (from
 NewMemHandle()
)
```

```
size - the amount of memory to allocate
```

## RESULT

Returns a pointer to the memory block for you to use, or NULL if the request could not be satisfied.

## SEE ALSO

```
NewMemHandle()
,
FreeMemH()
```

## 1.190 ClearMemHandle()

## NAME

ClearMemHandle - free all memory allocated via a handle

## SYNOPSIS

```
ClearMemHandle(handle)
 A0
```

```
void ClearMemHandle(APTR);
```

## FUNCTION

This function frees all memory that has been allocated with

```
AllocMemH()
 via the specified handle. The memory handle itself
```

remains intact.

#### INPUTS

handle - memory handle (from  
NewMemHandle()  
)

#### SEE ALSO

NewMemHandle()  
,  
AllocMemH()  
,  
FreeMemHandle()

## 1.191 FreeMemH()

#### NAME

FreeMemH - free memory allocated with  
AllocMemH()

#### SYNOPSIS

FreeMemH(memory)  
A0

void FreeMemH(APTR);

#### FUNCTION

This function frees an individual memory chunk that was allocated  
using

AllocMemH()  
.

#### INPUTS

memory - memory address returned from  
AllocMemH()  
SEE ALSO

NewMemHandle()  
,  
AllocMemH()

## 1.192 FreeMemHandle()

#### NAME

FreeMemHandle - free a memory handle completely

#### SYNOPSIS

FreeMemHandle(handle)  
A0

```
void FreeMemHandle(APTR);
```

#### FUNCTION

This function frees all memory that was allocated using the specified handle, and then frees the handle itself.

#### INPUTS

```
handle - memory handle from
 NewMemHandle()
 SEE ALSO

 NewMemHandle()
 ,
 ClearMemHandle()
```

## 1.193 NewMemHandle()

#### NAME

NewMemHandle - allocate a new memory handle

#### SYNOPSIS

```
NewMemHandle(puddle_size, thresh_size, type)
 D0 D1 D2
```

```
APTR NewMemHandle(ULONG, ULONG, ULONG);
```

#### FUNCTION

This function allocates a new memory handle, to enable easy access to memory pooling and tracking functions.

If you wish to use the OS memory pooling routines, specify a puddle and a threshold size for the memory pool. If you do not specify these, the memory handle will use ordinary memory allocations and keep track of these via a linked list. A linked list will also be used if the creation of a memory pool fails for any reason.

You must specify the type of memory you want when you create the handle. All memory allocated with this handle will be of the requested type (ie you can not mix fast and chip memory within the same handle). The normal MEMF\_ flags are used for this, with the following notes:

- If MEMF\_CLEAR is specified, the AllocMemH() routine clears the memory itself (as the OS pooling routines do not support this).
- If MEMF\_PUBLIC is specified, it indicates that you want the memory handle to be shareable between tasks, and the allocation routines will use semaphore locking when accessing the handle.

The dopus5.library is linked with the standalone memory pool routines, and therefore these routines work under OS37 as well as OS39.

#### INPUTS

puddle\_size - size of puddles to use for pooling, or 0 for no pools  
thresh\_size - allocation threshold size for pooling  
type - type of memory to allocate

#### RESULT

Returns a memory handle for use with the other memory functions, or NULL for failure.

#### SEE ALSO

```
AllocMemH()
,
ClearMemHandle()
,
FreeMemH()
,
FreeMemHandle()
,
exec.library/AllocPooled(), exec.library/FreePooled(),
exec.library/CreatePool(), exec.library/DeletePool()
```

## 1.194 Misc\_Routines

### Miscellaneous Routines

```
Atoh()

BtoCStr()

BuildKeyString()

BytesToString()

ConvertRawKey()

DivideToString()

DivideU()

Itoa()

ItoaU()

QualValid()

Random()

StrCombine()

StrConcat()

Seed()
```



## 1.195 Atoh()

### NAME

Atoh - convert a hex ascii string to a long

### SYNOPSIS

```
Atoh(string, len)
 A0 D0
```

```
long Atoh(char *, long);
```

### FUNCTION

Converts an ascii representation of a hex value to a long value.

### INPUTS

string - string to convert  
len - length of string to convert, or -1 for the whole string

### RESULT

Returns the long value equivalent to the ascii string.

## 1.196 BtoCStr()

### NAME

BtoCStr - convert a BCPL string to a C string

### SYNOPSIS

```
BtoCStr(bstr, cstr, length)
 A0 A1 D0
```

```
void BtoCStr(BSTR, char *, long);
```

### FUNCTION

Converts the supplied BSTR to a null-terminated C string.

### INPUTS

bstr - BCPL pointer to BSTR to convert  
cstr - buffer to store converted string in  
length - size of buffer

### RESULT

The string is converted. BSTRs are limited to 255 characters.

## 1.197 BuildKeyString()

---

## NAME

BuildKeyString - build a commodities key code string

## SYNOPSIS

```
BuildKeyString(code, qual, qual_mask, qual_same, buffer)
 D0 D1 D2 D3 A0
```

```
void BuildKeyString(USHORT, USHORT, USHORT, USHORT, char *);
```

## FUNCTION

Takes the supplied key code and qualifier and converts them to an ASCII string that is compatible with Commodities.

## INPUTS

code - key code  
 qual - key qualifier  
 qual\_mask - mask of the qualifiers to care about  
 qual\_same - which qualifiers are equivalent

## RESULT

The string is stored in the supplied buffer. String lengths can vary but for safety this buffer should be at least 80 bytes.

## SEE ALSO

commodities.library/ParseIX()

## 1.198 BytesToString()

## NAME

BytesToString - build a string representation of a byte size

## SYNOPSIS

```
BytesToString(bytes, buffer, places, separator)
 D0 A0 D1 D2
```

```
void BytesToString(ULONG, char *, short, char);
```

## FUNCTION

This routine takes a long value and creates a string to represent that value as an expression of size. Some examples are :

```
102 -> 102b
5804 -> 5K
1829382 -> 1.8M
```

## INPUTS

bytes - byte value  
 buffer - buffer to store result  
 places - number of decimal places. This must be set to 1 currently.  
 separator - column separator (eg a comma could produce "1,193")

## RESULT

The string is stored in the buffer. The buffer should be at least 16 bytes long.

## 1.199 ConvertRawKey()

### NAME

ConvertRawKey - convert a key from the raw key code

### SYNOPSIS

```
ConvertRawKey(code, qual, keybuf)
 D0 D1 A0
```

```
BOOL ConvertRawKey(USHORT, USHORT, char *);
```

### FUNCTION

Takes the supplied code and qualifier and returns the equivalent key in the current key map. This function provides a convenient path to the console.device's RawKeyConvert() routine.

### INPUTS

code - key code  
qual - key qualifier  
keybuf - buffer to store key

### RESULT

The key is stored in the supplied buffer. Most keys only require a single byte but in case one is larger the buffer should be at least 8 bytes in size.

### SEE ALSO

console.device/RawKeyConvert()

## 1.200 DivideToString()

### NAME

DivideToString - divide two numbers, store the result as ASCII

### SYNOPSIS

```
DivideToString(buffer, numerator, denominator, places, separator)
 A0 D0 D1 D2 D3
```

```
void DivideToString(char *, ULONG, ULONG, short, char);
```

### FUNCTION

This routine divides the numerator by the denominator, and stores the result with one decimal place precision as an ASCII string.

### INPUTS

string - buffer to store result  
numerator - number to divide

---

denominator - number to divide by  
places - decimal places, must be set to 1 for now  
separator - columns separator (eg a comma might produce "1,103")

#### RESULT

The division is performed and the result stored in the buffer.

## 1.201 DivideU()

#### NAME

DivideU - 32bit unsigned division with remainder

#### SYNOPSIS

DivideU( numerator, denominator, remainptr, utllib)  
D0 D1 A0 A1

```
ULONG DivideU(ULONG, ULONG, ULONG *, struct Library *);
```

#### FUNCTION

This routine calls the utility.library UDivMod32() routine, and returns the result. Any remainder is stored in the supplied variable.

#### INPUTS

numerator - number to divide  
denominator - number to divide by  
remainptr - pointer to ULONG to store the remainder  
utllib - pointer to UtilityBase

#### RESULT

Returns the integer result. The remainder is stored in the supplied variable.

#### SEE ALSO

utility.library/UDivMod32()

## 1.202 Itoa()

#### NAME

Itoa - converts signed integer to a string

#### SYNOPSIS

Itoa(num, string, separator)  
D0 A0 D1

```
void Itoa(long, char *, char);
```

#### FUNCTION

This routine takes the supplied signed number and converts it to an ASCII string.

---

## INPUTS

num - number to convert  
string - string to store result  
separator - column separator character, or 0 for no separator.

## RESULT

The string is stored in the supplied buffer.

## SEE ALSO

ItoaU()

## 1.203 ItoaU()

## NAME

ItoaU - converts unsigned integer to a string

## SYNOPSIS

```
ItoaU(num, string, separator)
 D0 A0 D1
```

```
void ItoaU(long, char *, char);
```

## FUNCTION

This routine takes the supplied unsigned number and converts it to an ASCII string.

## INPUTS

num - number to convert  
string - string to store result  
separator - column separator character, or 0 for no separator.

## RESULT

The string is stored in the supplied buffer.

## SEE ALSO

Itoa()

## 1.204 QualValid()

## NAME

QualValid - mask out invalid qualifiers

## SYNOPSIS

```
QualValid(qual)
 D0
```

```
USHORT QualValid(USHORT);
```

## FUNCTION

Masks out invalid qualifiers from the supplied value and returns the result.

## INPUTS

qual - qualifier mask

## RESULT

The return value is the new qualifier mask. Only the following qualifiers are considered "valid" for operations within Opus :

```
IEQUALIFIER_LCOMMAND, IEQUALIFIER_RCOMMAND,
IEQUALIFIER_LSHIFT, IEQUALIFIER_RSHIFT,
IEQUALIFIER_LALT, IEQUALIFIER_RALT,
IEQUALIFIER_CONTROL, IEQUALIFIER_NUMERICPAD
```

## 1.205 Random()

## NAME

Random - generate a pseudo-random number

## SYNOPSIS

```
Random(limit)
D0
```

```
long Random(long);
```

## FUNCTION

Returns a pseudo-random number between 0 and 'limit' inclusive.

## INPUTS

limit - upper limit of number

## RESULT

Returns random number.

## SEE ALSO

Seed()

## 1.206 StrCombine()

## NAME

StrCombine - combine two strings into one buffer

## SYNOPSIS

```
StrCombine(buffer, first, second, size)
A0 A1 A2 D0
```

```
BOOL StrCombine(char *, char *, char *, long);
```

## FUNCTION

Combines the two supplied strings into the one buffer.

## INPUTS

buffer - buffer to store result  
first - first string  
second - second string  
size - size of buffer

## RESULT

Returns TRUE if both strings fitted in the buffer, or FALSE if they had to be truncated.

## SEE ALSO

StrConcat()

## 1.207 StrConcat()

## NAME

StrConcat - concatenate two strings

## SYNOPSIS

StrConcat(first, second, size)  
          A0      A1      D0

BOOL StrConcat(char \*, char \*, long);

## FUNCTION

Joins the second string to the end of the first string.

## INPUTS

first - first string  
second - string to join  
size - size of first buffer

## RESULT

The second string is joined to the end of the first string. This function returns TRUE if the second string fitted in the buffer, or FALSE if it had to be truncated.

## SEE ALSO

StrCombine()

## 1.208 Seed()

## NAME

Seed - seed the random number generator

---

## SYNOPSIS

```
Seed(seed)
 D0
```

```
void Seed(long);
```

## FUNCTION

Seeds the random number generator.

## INPUTS

seed - value to seed generator with

## SEE ALSO

```
Random()
```

## 1.209 Notify\_Routines

Notify Routines

```
AddNotifyRequest()
```

```
RemoveNotifyRequest()
```

```
ReplyFreeMsg()
```

```
SetNotifyRequest()
```

## 1.210 AddNotifyRequest()

## NAME

AddNotifyRequest - add a request to Opus's notify chain

## SYNOPSIS

```
AddNotifyRequest(type, userdata, port)
 D0 D1 A0
```

```
APTR AddNotifyRequest(ULONG, ULONG, struct MsgPort *);
```

## FUNCTION

Opus keeps track of several different system events, and this routine allows you to request notification on them.

The events currently available for notification:

```
DN_WRITE_ICON - an icon is written to disk
DN_APP_ICON_LIST - an AppIcon is added or removed
DN_APP_MENU_LIST - an AppMenuItem is added or removed
```



DN\_CLOSE\_WORKBENCH - CloseWorkbench() has been called  
 DN\_OPEN\_WORKBENCH - OpenWorkbench() has been called  
 DN\_RESET\_WORKBENCH - the workbench screen has been closed and  
     re-opened  
 DN\_DISKCHANGE - a disk has been inserted or removed  
 DN\_DOS\_ACTION - a DOS event has occurred. In Opus 5.5,  
     these messages are only available if the  
     dopus/DOSPatch environment variable is set.  
 DN\_REXX\_UP - the ARExx process has been started.

Several Opus events are also available for notification:

DN\_OPUS\_START - Opus has started  
 DN\_OPUS\_QUIT - Opus has quit  
 DN\_OPUS\_HIDE - Opus has been hidden  
 DN\_OPUS\_SHOW - Opus has been revealed

When an event occurs that you have requested notification for, a DOpusNotify message is sent to your message port. The message structure is defined as follows:

dn\_Msg - Exec message header  
 dn\_Type - Event type  
 dn\_UserData - the userdata you supplied to AddNotifyRequest()  
 dn\_Data - data specific to the type of event  
 dn\_Flags - flags specific to the type of event  
 dn\_Fib - a FileInfoBlock for some types of event  
 dn\_Name - pathname specific to the type of event

The event-specific fields are used in the following way:

#### DN\_WRITE\_ICON

dn\_Data - NULL  
 dn\_Flags - if DNF\_ICON\_REMOVED is set, icon was deleted  
 dn\_Fib - NULL  
 dn\_Name - full pathname of icon

#### DN\_APP\_ICON\_LIST

dn\_Data - pointer to the AppIcon added or removed  
 dn\_Flags - if DNF\_ICON\_REMOVED is set, icon was removed  
     if DNF\_ICON\_CHANGED is set, the icon image  
     was changed  
 dn\_Fib - NULL  
 dn\_Name - NULL

#### DN\_APP\_MENU\_LIST

dn\_Data - pointer to the AppMenuItem added or removed

dn\_Flags - if DNF\_ICON\_REMOVED is set, item was removed  
 dn\_Fib - NULL  
 dn\_Name - NULL

#### DN\_DISKCHANGE

-----

dn\_Data - disk units the change occurred in (bits 0-3  
 represent units 0-3)  
 dn\_Flags - which units have disks in them (bits 0-3  
 represent units 0-3)  
 dn\_Fib - NULL  
 dn\_Name - NULL

#### DN\_DOS\_ACTION

-----

dn\_Data - NULL  
 dn\_Flags - which DOS action occurred (see <dopus/notify.h>)  
 dn\_Fib - FileInfoBlock with file information. This is  
 supplied for all actions except Delete.  
 dn\_Name - full pathname of file involved

#### INPUTS

type - type of events you want to be notified of. One request can  
 ask for multiple events. See <dopus/notify.h> for the full  
 list.

userdata - a user-defined data field that is passed in any notify  
 messages.

port - message port to send notification messages to.

#### NOTES

Most notification messages are sent "reply free", meaning you must  
 use the

```

 ReplyFreeMsg()
 call to reply to them. Otherwise, the
message memory will be lost.
```

#### RESULT

Returns a notify handle which you use to remove the request.

#### SEE ALSO

```

 RemoveNotifyRequest()
 ,
 SetNotifyRequest()
```

## 1.211 RemoveNotifyRequest()

#### NAME

RemoveNotifyRequest - remove a notification request

#### SYNOPSIS

```
RemoveNotifyRequest(request)
```

A0

```
void RemoveNotifyRequest (APTR);
```

#### FUNCTION

Removes a notify request that was added with  
AddNotifyRequest()

AddNotifyRequest()

.

#### INPUTS

request - request to remove

#### RESULT

The request is removed. You will receive no more notifications for that request. Once you have removed the request you should check your message port for outstanding messages and reply to them.

#### SEE ALSO

AddNotifyRequest()

## 1.212 ReplyFreeMsg()

#### NAME

ReplyFreeMsg - reply or free a message

#### SYNOPSIS

```
ReplyFreeMsg(msg)
```

A0

```
void ReplyFreeMsg(struct Message *);
```

#### FUNCTION

If the message has a valid ReplyPort, this function simply passes it through to ReplyMsg(). If the message has no reply port set, this function calls FreeVec() on the message to free it.

#### INPUTS

msg - message to reply or free

#### NOTES

Most Opus notification messages are sent "reply free", meaning you MUST use this function to reply to them or the memory will be lost.

#### SEE ALSO

AddNotifyRequest()

## 1.213 SetNotifyRequest()

---

## NAME

SetNotifyRequest - change notification events

## SYNOPSIS

```
SetNotifyRequest(request, new_type, mask)
```

```
 A0 D0 D1
```

```
void SetNotifyRequest(APTR, ULONG, ULONG);
```

## FUNCTION

This routine changes the type of events that an existing notification request is interested in.

## INPUTS

request - notification request to change

new\_type - the new event flags to receive notification about

mask - mask of event flags to change (any events not specified in the mask field will not be modified)

## SEE ALSO

AddNotifyRequest()

## 1.214 Popup\_Routines

Popup Menu Routines

DoPopUpMenu()

GetPopUpItem()

## 1.215 DoPopUpMenu()

## NAME

DoPopUpMenu - display a popup menu

## SYNOPSIS

```
DoPopUpMenu(window, menu, itemptr, button)
```

```
 A0 A1 A2 D0
```

```
USHORT DoPopUpMenu(struct Window *, PopUpMenu *, PopUpItem **, USHORT);
```

## FUNCTION

This routine displays a popup menu. The PopUpMenu structure must be initialised as follows:

item\_list - a list of the menu items

---

locale - a pointer to an initialised DOpusLocale structure

flags - menu flags. Currently supported flags are:

- POPUPMF\_HELP - menu supports help via the user pressing the help key
- POPUPMF\_REFRESH - the callback function supplied should be used to refresh the parent window
- POPUPMF\_ABOVE - the popup menu should open above the parent window, instead of over the current mouse position

callback - pointer to your refresh callback function, or NULL

The callback function is a function that you define to handle the situation when the parent window needs to be refreshed. If the parent window is `simplerefresh`, you should provide this function. The function has the following prototype:

```
void __asm refresh_callback(register __d0 ULONG type,
 register __a0 struct Window *window,
 register __a1 ULONG userdata)
```

The routine will be called whenever the parent window needs to be refreshed. 'type' is the IDCMP message type; usually `IDCMP_REFRESHWINDOW`. 'window' is a pointer to the parent window, and 'userdata' is the userdata field of the `PopUpMenu` structure.

The 'item\_list' parameter is a `MinList` containing the items of the popup menu. Each node on this list is a `PopUpItem` structure, which is defined as follows :

item\_name - pointer to item name

id - item ID

flags - item flags. Currently supported flags are:

- POPUPF\_LOCALE - signifies that 'item\_name' is not a pointer to a string, but is a locale ID representing a string in the supplied locale.
- POPUPF\_CHECKIT - item can be checked, much like `CHECKIT` in Intuition menus
- POPUPF\_CHECKED - item starts out checked, much like `CHECKED` in Intuition menus
- POPUPF\_SUB - item has sub-items
- POPUPF\_DISABLED - item is disabled

data - unless POPUPF\_SUB is set, this is a userdata field that can be set to anything. If POPUPF\_SUB is set, this field must point to an initialised MinList containing the PopUpItem structures for the sub-menu. You can have up to four levels of sub-menus.

Set 'item\_name' to the special value POPUP\_BARLABEL to produce a separator bar in the menu.

#### INPUTS

window - Parent window to open menu over  
 menu - PopUpMenu to open  
 itemptr - Pointer to location to receive a pointer to the selected item  
 button - The code of the mouse button pressed to generate this menu. This is used to control which mouse button release will remove the menu (eg, if you pass SELECTDOWN for the 'button' value, the menu will be removed on a SELECTUP event)

#### RESULT

Returns -1 if no item was selected. If an item was selected, the item ID is returned, and the address of the PopUpItem structure is stored in 'itemptr'. If the user pressed the help key over an item and the POPUPMF\_HELP flag is set, the POPUP\_HELPFLAG flag will be set in the returned item ID.

#### SEE ALSO

GetPopUpItem()

## 1.216 GetPopUpItem()

#### NAME

GetPopUpItem - find a PopUpItem by ID

#### SYNOPSIS

```
GetPopUpItem(menu, id)
 A0 D0
```

```
PopUpItem *GetPopUpItem(PopUpMenu *, USHORT);
```

#### FUNCTION

This searches the supplied menu for a PopUpItem with the specified ID, and returns a pointer to it.

#### INPUTS

menu - PopUpMenu to search  
 id - ID to search for

#### RESULT

Returns a pointer to the PopUpItem if found, or else NULL. This routine supports one level of sub-menus, and will not find an item that is more than one sub-menu deep.

## NOTES

This routine is useful in allowing you to find an item to set the state of the POPUPF\_CHECKED and POPUPF\_DISABLED flags.

SEE ALSO

DoPopUpMenu()

## 1.217 Progress\_Routines

Progress Indicator Routines

CheckProgressAbort()

CloseProgressWindow()

GetProgressWindow()

HideProgressWindow()

OpenProgressWindow()

SetProgressWindow()

ShowProgressWindow()

## 1.218 CheckProgressAbort()

## NAME

CheckProgressAbort - check for abort in progress window

## SYNOPSIS

CheckProgressAbort(handle)  
A0

BOOL CheckProgressAbort(APTR);

## FUNCTION

Allows you to check the state of the abort flag in the specified progress window.

## INPUTS

handle - progress window handle

## RESULT

Returns TRUE if the Abort button has been clicked.

SEE ALSO

---

OpenProgressWindow()

## 1.219 CloseProgressWindow()

NAME

CloseProgressWindow - close a progress window

SYNOPSIS

CloseProgressWindow(handle)  
A0

void CloseProgressWindow(APTR);

FUNCTION

Closes the specified progress window.

INPUTS

handle - progress window to close

SEE ALSO

OpenProgressWindow()

## 1.220 GetProgressWindow()

NAME

GetProgressWindow - get progress window information

SYNOPSIS

GetProgressWindow(handle, tags)  
A0 A1

void GetProgressWindow(APTR, struct TagItem \*);

void GetProgressWindowTags(APTR, Tag, ...);

FUNCTION

Returns information about the progress window. Currently available information is :

PW\_Window - returns Window pointer

INPUTS

handle - progress window handle  
tags - inquiry tags

RESULT

The result of each tag query is stored in the ti\_Data field of the TagItem.

---



SEE ALSO

```
OpenProgressWindow()
,
SetProgressWindow()
```

## 1.221 HideProgressWindow()

NAME

HideProgressWindow - hide a progress window

SYNOPSIS

```
HideProgressWindow(handle)
A0
```

```
void HideProgressWindow(APTR);
```

FUNCTION

Removes the specified progress window from the display. The progress window is still operative; it can still be updated and even closed while it is hidden.

INPUTS

handle - progress window handle

SEE ALSO

```
OpenProgressWindow()
,
ShowProgressWindow()
```

## 1.222 OpenProgressWindow()

NAME

OpenProgressWindow - open a progress window display

SYNOPSIS

```
OpenProgressWindow(tags)
A0
```

```
APTR OpenProgressWindow(struct TagItem *);
```

```
APTR OpenProgressWindowTags(Tag, ...);
```

FUNCTION

Opens a progress window that your application can use to display the progression of some operation. The progress window can have a filename display, file counter and progress gauge.

INPUTS

tags - control tags. Control tags are:

PW\_Screen - screen to open on. The progress window will appear centred in the screen. Use of this tag overrides PW\_Window.

PW\_Window - window to open over. The progress window will appear centred over the supplied window.

PW\_Title - title for the progress window

PW\_SigTask - task to signal when the abort gadget is pressed.

PW\_SigBit - bit to signal task with (signal bit, not mask)

PW\_FileName - initial filename for display

PW\_FileSize - initial file size

PW\_FileCount - initial file count

PW\_Flags - control flags

The control flags for the PW\_Flags tag are :

PWF\_FILENAME - specify if you want a filename display

PWF\_FILESIZE - specify if you want a file size display (1)

PWF\_INFO - specify if you want an information line

PWF\_GRAPH - specify if you want a bar graph display (1)

PWF\_NOABORT - specify if you don't want an Abort button (2)

PWF\_INVISIBLE - if you want the progress window to open in 'hidden' mode (ie you need to call ShowProgressWindow() to make it visible)

PWF\_ABORT - specify if you do want an Abort button (2)

1. Ordinarily, the file size is displayed as a "xxxxxx bytes" string in the top-right of the progress window, and the bar graph is used to represent "x out of y files". If, however, you specify both PWF\_FILESIZE and PWF\_GRAPH, the meanings of these displays is automatically swapped around. The current file progress (eg bytes copied) is displayed in the bar graph, and the current operation progress (eg files copied) is displayed in text in the top-right corner.
2. If you specify a signal task with the pw\_SigTask flag, the progress window will automatically get an Abort button. You can use the PWF\_NOABORT flag to stop this happening.

If you do not specify the pw\_SigTask flag, you can use the PWF\_ABORT flag to add an Abort button without signalling (you will need to call

```
CheckProgressAbort()
to detect an abort).
```

RESULT

---

Returns a handle to the newly created progress window, or NULL for failure.

SEE ALSO

```
SetProgressWindow()
,
CloseProgressWindow()
```

## 1.223 SetProgressWindow()

NAME

SetProgressWindow - update progress window information

SYNOPSIS

```
SetProgressWindow(handle, tags)
 A0 A1
```

```
void SetProgressWindow(APTR, struct TagItem *);
```

```
void SetProgressWindowTags(APTR, Tag, ...);
```

FUNCTION

This is the routine you use to update the information displayed in a progress window.

INPUTS

handle - progress window handle  
tags - control tags. Valid tags are :

```
PW_Title - change the window title

PW_FileName - change the displayed filename

PW_FileSize - change the total size of the current file

PW_FileDone - change the "done" size of the current file
 (eg, if the file size was 12800 and you had
 copied half of it, the done size would be
 6400).

PW_Info - change the information field display

PW_FileCount - change the total number of files

PW_FileNum - change the number of files processed (eg,
 if the total file count was 84 and you had
 processed a quarter of them, the current
 file number would be 21).
```

RESULT

The changes are displayed immediately. If the progress window is currently hidden, the changes are still effective and will be visible when the progress window is revealed.

SEE ALSO

OpenProgressWindow()

## 1.224 ShowProgressWindow()

NAME

ShowProgressWindow - reveal a hidden progress window

SYNOPSIS

```
ShowProgressWindow(handle, screen, window)
 A0 A1 A2
```

```
void ShowProgressWindow(APTR, struct Screen *, struct Window *);
```

FUNCTION

Reveals a progress window that was hidden with  
HideProgressWindow()

.

INPUTS

handle - progress window handle  
screen - new parent screen (if no window supplied)  
window - new owner window (if no screen supplied)

RESULT

The progress window is revealed. If possible, it will be displayed at the same position on the screen as it was when it was hidden.

SEE ALSO

OpenProgressWindow()

,

HideProgressWindow()

## 1.225 Requester\_Routines

Requester Routines

AsyncRequest()

OpenStatusWindow()

SelectionList()

SetStatusText()

## 1.226 AsyncRequest()

### NAME

AsyncRequest - display a requester

### SYNOPSIS

```
AsyncRequest(ipc, type, window, callback, data, tags)
 A0 D0 A1 A2 A3 D1
```

```
long AsyncRequest(IPCData *, long, struct Window *,
 REF_CALLBACK, APTR, struct TagItem *);
```

```
long AsyncRequestTags(IPCData *, long, struct Window *,
 REF_CALLBACK, APTR, Tag, ...);
```

### FUNCTION

Displays requesters of different types. The name of this function is slightly misleading, as the routine itself is not asynchronous. However, the requester is launched by a separate process, which makes it possible for you to provide a callback function that can handle refreshing of a window while the requester is displayed. There are currently two types of requesters defined:

#### REQTYPE\_FILE

This opens an ASL file requester. The FileRequester itself is defined by you; this routine simply opens it with a separate process, providing asynchronicity. The only value tag for this requester type is AR\_Requester, with which you specify the address of a file requester structure obtained via AllocAslRequest().

#### REQTYPE\_SIMPLE

This displays a simple text requester to the user. There are several control tags for this requester type which give you great control over the appearance of the requester :

AR\_Window (struct Window \*)

Use this to specify a parent window for the requester. The requester will appear centred over this window (overrides AR\_Screen)

AR\_Screen (struct Screen \*)

Use this to specify a parent screen for the requester. The requester will appear centred in the screen.

AR\_Title (char \*)

The requester title. This is displayed in the title bar of the requester window. If not specified, this value defaults to "Directory Opus Request".

AR\_Message (char \*)

The requester message. This is the text displayed in the main body of the requester. Use a \n character to represent a linefeed.

AR\_Button (char \*)

This tag allows you to define a button for the requester. You can use this tag multiple times.

AR\_ButtonCode (long)

Specifies the ID code for the previous AR\_Button tag. By default, buttons are numbered 1, 2, 3, ... in the order they appear in the tag list. This tag allows you to change the ID codes, and therefore the result code from the AsyncRequest() function.

AR\_Buffer (char \*)

If you want a string gadget to be displayed in the requester, specify this tag with a pointer to a string buffer.

AR\_BufLen (long)

If a buffer was specified with AR\_Buffer, you must also supply this tag to set the size of the buffer.

AR\_History (Att\_List \*)

Points to an Att\_List which contains the history list for this gadget. If supplied, the user will be able to press the cursor up and down keys to access the history. See the docs on

GetEditHook()  
for more information.

AR\_CheckMark (char \*)

If you want a check mark gadget to appear in the requester, specify this as a pointer to the text for the gadget.

AR\_CheckPtr (short \*)

If you specify the AR\_CheckMark tag, you must also supply this tag. ti\_Data is a pointer to a short variable which will receive the state of the checkmark gadget when the requester is closed.

AR\_Flags (ULONG)

Control flags.

The control flags for the AR\_Flags tag are :

|                 |                                            |
|-----------------|--------------------------------------------|
| SRF_LONGINT     | - the string gadget is an integer field    |
| SRF_CENTJUST    | - center-justify the string gadget         |
| SRF_RIGHTJUST   | - right-justify the string gadget          |
| SRF_PATH_FILTER | - filter path characters from string field |

```

SRF_SECURE - set for secure password field
SRF_HISTORY - set if supplying the AR_History tag
SRF_CHECKMARK - set if supplying the AR_CheckMark tag
SRF_MOUSE_POS - center requester over mouse pointer

```

The callback function is a function that you define to handle the situation when the parent window needs to be refreshed. If the parent window is `simplerefresh`, you should provide this function. The function has the following prototype:

```

void __asm refresh_callback(register __d0 ULONG type,
 register __a0 struct Window *window,
 register __a1 ULONG data)

```

The routine will be called whenever the parent window needs to be refreshed. 'type' is the IDCMP message type; usually `IDCMP_REFRESHWINDOW`. 'window' is a pointer to the parent window, and 'data' is the data value passed to the `AsyncRequest()` function.

#### INPUTS

```

ipc - your process' IPCData pointer
type - type of requester to display
window - parent window for requester
callback - your callback function
data - data that is passed to the callback
tags - control tags

```

#### RESULT

Returns the result from the requester. Returns 0 if the requester could not be displayed.

#### NOTES

For a `REQTYPE_SIMPLE` requester, the default gadget IDs are (from left to right), 1, 2, 3 ... 0. The right-most gadget is defined as 0 to act as a "cancel" gadget. Therefore, in a simple "Ok", "Cancel" requester, "Ok" returns 1 (or TRUE) and "Cancel" returns 0 (or FALSE).

#### SEE ALSO

```

asl.library/AllocAslRequest(),
 GetEditHook()

```

## 1.227 OpenStatusWindow()

#### NAME

`OpenStatusWindow` - open a status window

#### SYNOPSIS

```

OpenStatusWindow(title, text, screen, flags, unused)
 A0 A1 A2 D0 D1

```

```

struct Window *OpenStatusWindow(char *, char *, struct Screen *,
 ULONG, long);

```

## FUNCTION

A status window is kind of like a "dumb" progress window; it has the ability to display a single line of text.

## INPUTS

title - status window title  
 text - initial text to display  
 screen - screen to open on  
 flags - set to 0  
 unused - set to 0

## RESULT

Returns a pointer to the new window. To close the status window, call

```
CloseConfigWindow()
 on it.
```

## SEE ALSO

```
SetStatusText()
,
CloseConfigWindow()
```

## 1.228 SelectionList()

## NAME

SelectionList - display a list in a requester

## SYNOPSIS

```
SelectionList(list, window, screen,
 title, initialsel, flags,
 buffer, okay_txt, cancel_txt)
```

```
short SelectionList(Att_List *, struct Window *, struct Screen *,
 char *, short, ULONG,
 char *, char *, char *);
```

## FUNCTION

This routine displays a requester containing a listview gadget, prompting the user to select an item from the list. The requester can optionally have a directory field, which allows the user to open an ASL file requester to locate a directory that is not in the list.

## INPUTS

list - Att\_List to display (the name of each node is displayed)  
 window - parent window  
 screen - screen to open on if no window specified  
 title - title of requester  
 initialsel - initially selected item, or -1 for no selection  
 flags - control flags. Specify SLF\_DIR\_FIELD to get a directory field  
 buffer - If SLF\_DIR\_FIELD is specified, this must point to a buffer (256 bytes or greater) to contain the path name chosen by the user



okay\_txt - text for the "Ok" gadget  
cancel\_txt - text for the "Cancel" gadget

#### RESULT

Returns the number of the selected item in the list, or -1 if the user made no selection. If a directory field was specified with SLF\_DIR\_FIELD, and -1 is returned, you should check the supplied buffer to see if it is empty. If not, the user selected a path manually.

#### SEE ALSO

Att\_NewList()

## 1.229 SetStatusText()

#### NAME

SetStatusText - change text in a status window

#### SYNOPSIS

```
SetStatusText(window, text)
 A0 A1
```

```
void SetStatusText(struct Window *, char *);
```

#### FUNCTION

Changes the text displayed in the supplied status window.

#### INPUTS

window - status window  
text - new text to display

#### RESULT

The text is displayed immediately. Do NOT call this function on a window other than one returned by the  
OpenStatusWindow()  
call.

#### SEE ALSO

OpenStatusWindow()

## 1.230 Timer\_Routines

Timer Routines

AllocTimer()

CheckTimer()

```
FreeTimer()

GetTimerBase()

StartTimer()

StopTimer()

TimerActive()
```

## 1.231 AllocTimer()

NAME

AllocTimer - allocate a timer handle

SYNOPSIS

```
AllocTimer(unit, port)
 D0 A0
```

```
TimerHandle *AllocTimer(ULONG, struct MsgPort *);
```

FUNCTION

This function allocates a timer handle to enable easy use of the timer.device. You can supply a message port for it to use, or have it create one for you. If you do not supply a message port, the "port" field of the returned TimerHandle structure contains the address of the port that was created for you.

INPUTS

unit - the timer.device unit you wish to use (eg UNIT\_VBLANK)  
port - message port to use (or NULL to have one created)

RESULT

Returns a TimerHandle to use with the other functions.

SEE ALSO

```
FreeTimer()
,
StartTimer()
```

## 1.232 CheckTimer()

NAME

CheckTimer - see if a timer request has completed

SYNOPSIS

```
CheckTimer(handle)
 A0
```

```
BOOL CheckTimer(TimerHandle *);
```

#### FUNCTION

This function allows you to discover if a timer request you have started has completed.

#### INPUTS

handle - timer handle

#### RESULT

Returns TRUE if the request is complete, or FALSE if it has not completed or is invalid.

#### SEE ALSO

```
StartTimer()
,
StopTimer()
```

### 1.233 FreeTimer()

#### NAME

FreeTimer - free a timer handle

#### SYNOPSIS

```
FreeTimer(handle)
A0
```

```
void FreeTimer(TimerHandle *);
```

#### FUNCTION

This function frees a timer handle created with  
AllocTimer()

. Any

outstanding request is aborted automatically. If you supplied your own message port to the

AllocTimer()

function, you are responsible for deleting the port yourself.

#### INPUTS

handle - timer handle

#### SEE ALSO

```
AllocTimer()
```

### 1.234 GetTimerBase()

---

## NAME

GetTimerBase - get a pointer to the timer.device library base

## SYNOPSIS

GetTimerBase()

```
struct Library *GetTimerBase(void);
```

## FUNCTION

This function returns a pointer to the library base of the timer.device. The library base pointer is needed if you want to call any of the library functions of the timer.device. This routine saves you having to open the timer.device to get this base pointer.

## INPUTS

none

## RESULT

Returns struct Library \* pointer. You must NOT call CloseLibrary() on this pointer.

## 1.235 StartTimer()

## NAME

StartTimer - send a timer request

## SYNOPSIS

```
StartTimer(handle, seconds, micros)
 A0 D0 D1
```

```
void StartTimer(TimerHandle *, ULONG, ULONG);
```

## FUNCTION

This function starts a timer request for a given period of time. Your code should wait on "handle->port" for a signal indicating a completed request. You can call

```
 CheckTimer()
```

at any time to see if the request has been completed.

## INPUTS

handle - timer handle  
seconds - number of seconds for the request  
micros - number of microseconds (0-999999)

## NOTES

You can call this routine with a request already pending; the first request will automatically be aborted.

## SEE ALSO

```
 AllocTimer()
```

,

---

```
StopTimer()
,
CheckTimer()
```

## 1.236 StopTimer()

NAME

StopTimer - stop a timer request in progress

SYNOPSIS

```
StopTimer(handle)
A0
```

```
void StopTimer(TimerHandle *);
```

FUNCTION

This function aborts a timer request that was previously started with

```
StartTimer()
. If the request has already completed, this
function simply does the cleanup.
```

INPUTS

handle - timer handle

SEE ALSO

```
AllocTimer()
,
StartTimer()
,
CheckTimer()
```

## 1.237 TimerActive()

NAME

TimerActive - check if a timer request is pending

SYNOPSIS

```
TimerActive(handle)
A0
```

```
BOOL TimerActive(TimerHandle *);
```

FUNCTION

If you lose track of (or can't be bothered keeping track of) whether or not you have a pending timer request, this function allows you to find out.

This function is actually not really necessary. All the timer

---

functions are robust enough to cope with multiple requests  
(a

    StartTimer()  
    with an already-pending request), or a  
    StopTimer()  
    when a request is already complete (or was never sent), or any ←  
    of

the other "error" conditions that the timer.device is usually  
sensitive to.

#### INPUTS

handle - timer handle

#### RESULT

Returns TRUE if there is a request pending.

#### SEE ALSO

AllocTimer()

## 1.238 Index

Index to DOpus55 Developer Guide & dopus5.library

A

ActivateStrGad()

AddNotifyRequest()

AddObjectList()

AddScrollBars()

AddSorted()

AddWindowMenus()

AllocAppMessage()

AllocMemH()

AllocTimer()

AppWindowData()

AppXXX\_routines

Arg\_Routines

AsyncRequest()

Atoh()

Att\_ChangeNodeName ()

Att\_FindNode ()

Att\_FindNodeData ()

Att\_FindNodeNumber ()

Att\_NewList ()

Att\_NewNode ()

Att\_NodeCount ()

Att\_NodeDataNumber ()

Att\_NodeName ()

Att\_NodeNumber ()

Att\_PosNode ()

Att\_RemList ()

Att\_RemNode ()

B

BOOPSIFree ()

BOOPSI\_gadgets

BoundsCheckGadget ()

BtoCStr ()

BufIO\_Routines

BuildKeyString ()

BuildMenuStrip ()

BytesToString ()

C

ChangeAppIcon ()

CheckAppMessage ()

CheckObjectArea ()

CheckProgressAbort ()

CheckTimer ()

---

ClearMemHandle()  
ClearWindowBusy()  
Clipboard\_Routines  
CloseBuf()  
CloseClipboard()  
CloseConfigWindow()  
CloseDisk()  
CloseImage()  
CloseProgressWindow()  
Contact and Support  
ConvertRawKey()  
CopyFileIcon()  
CopyImage()  
Copyrights  
D  
  
#defines (a-d)  
#defines (e-f)  
#defines (g-i)  
#defines (j-o)  
#defines (p-r)  
#defines (s-w)  
  
DateFromStrings()  
DeviceFromHandler()  
DeviceFromLock()  
DevNameFromLock()  
DisableObject()  
DiskIO\_Routines  
DisplayObject()  
DisposeArgs()

---



---

DisposeBitMap()  
DivideToString()  
DivideU()  
DoPopUpMenu()  
DOpus55 Developer Guide Index  
dopusbuttongclass  
dopuscheckgclass  
dopusframeclass  
DOpusGetString()  
dopusiclass  
dopuslistviewgclass  
dopuspalettegclass  
dopusstrgclass  
dopusviewgclass  
DOS\_Routines  
Drag\_Routines  
DrawBox()  
DrawFieldBox()  
E  
  
Edit\_Hook  
EndRefreshConfigWindow()  
ExamineBuf()  
Example files  
F  
  
FHFromBuf()  
FindAppWindow()  
FindBOOPSIGadget()  
FindMenuItem()

---

---

FindNameI ()  
FindPubScreen ()  
FlushBuf ()  
FreeAppMessage ()  
FreeCachedDiskObject ()  
FreeDosPathList ()  
FreeDragInfo ()  
FreeEditHook ()  
FreeImageRemap ()  
FreeMemH ()  
FreeMemHandle ()  
FreeObjectList ()  
FreeRemapImage ()  
FreeTimer ()  
FreeWindowMenus ()  
G  
  
GetCachedDefDiskObject ()  
GetCachedDiskObject ()  
GetCachedDiskObjectNew ()  
GetDosPathList ()  
GetDragImage ()  
GetDragInfo ()  
GetDragMask ()  
GetEditHook ()  
GetFileVersion ()  
GetGadgetValue ()  
GetIconFlags ()  
GetIconPosition ()  
GetImageAttrs ()

---

GetImagePalette ()

GetObject ()

GetObjectRect ()

GetPalette32 ()

GetPopUpItem ()

GetProgressWindow ()

GetSecureString ()

GetSemaphore ()

GetTimerBase ()

GetWBArgPath ()

GetWindowAppPort ()

GetWindowID ()

GetWindowMsg ()

Global table of contents

GUI\_Routines

H

Headers etc

HideDragImage ()

HideProgressWindow ()

I

IFFChunkID ()

IFFChunkRemain ()

IFFChunkSize ()

IFFClose ()

IFFFailure ()

IFFGetForm ()

IFFNextChunk ()

IFFOpen ()

---

IFFPopChunk ()  
IFFPushChunk ()  
IFFReadChunkBytes ()  
IFFWriteChunk ()  
IFFWriteChunkBytes ()  
IFF\_Routines  
Image\_Routines  
InitListLock ()  
IPC\_Command ()  
IPC\_FindProc ()  
IPC\_Flush ()  
IPC\_Free ()  
IPC\_Launch ()  
IPC\_ListCommand ()  
IPC\_ProcStartup ()  
IPC\_Reply ()  
IPC\_Routines  
IsListLockEmpty ()  
Itoa ()  
ItoaU ()  
L  
  
LaunchCLI ()  
LaunchWB ()  
LayoutResize ()  
Layout\_Routines  
List\_Routines  
LoadPalette32 ()  
Locale\_Routines  
LockAttList ()

---

---

M

Memory\_Routines

Misc\_Routines

Module\_Definition

N

NewBitMap ()

NewMemHandle ()

Notify\_Routines

O

OpenBuf ()

OpenClipboard ()

OpenConfigWindow ()

OpenDisk ()

OpenImage ()

OpenProgressWindow ()

OpenStatusWindow ()

P

ParseArgs ()

ParseDateStrings ()

Popup\_Routines

Progress\_Routines

Q

QualValid ()

R

Random ()

ReadBuf ()

ReadClipString ()

RemapImage ()

---

---

RemoveNotifyRequest ()  
RenderImage ()  
ReplyAppMessage ()  
ReplyFreeMsg ()  
ReplyWindowMsg ()  
Requester\_Routines  
S  
  
ScreenInfo ()  
SearchFile ()  
Seed ()  
SeekBuf ()  
SelectionList ()  
SetAppIconMenuState ()  
SetBusyPointer ()  
SetConfigWindowLimits ()  
SetEnv ()  
SetGadgetChoices ()  
SetGadgetValue ()  
SetIconFlags ()  
SetIconPosition ()  
SetNotifyRequest ()  
SetProgressWindow ()  
SetStatusText ()  
SetWBArg ()  
SetWindowBusy ()  
SetWindowID ()  
ShowDragImage ()  
ShowProgressWindow ()  
StampDragImage ()

---

StartRefreshConfigWindow()

StartTimer()

StopTimer()

StrCombine()

StrConcat()

SwapListNodes()

T

TimerActive()

Timer\_Routines

TypeDefs etc

U

UnlockAttList()

W

WriteBuf()

WriteClipString()