

WINDOWS API

VOLUME III

WINDOWS 3.1
REFERENCE GUIDE

B O R L A N D

Amazon Ex. 1016
IPR Petition - USP 7,784,058

Oracle Ex. 1016
Oracle v. VirtaMove, IPR2025-00966

Windows API Guide

Reference

Volume 3

Version 3.1
for the MS-DOS and PC-DOS
Operating Systems

Copyright © 1992 by Borland International. All rights reserved.
All Borland products are trademarks or registered trademarks of
Borland International, Inc. Other brand and product names are
trademarks or registered trademarks of their respective holders.

PRINTED IN THE USA.
10 9 8 7 6 5 4 3

C o n t e n t s

Chapter 1 Common dialog box library	1
Using Color dialog boxes	3
Color models used by the Color dialog box	4
RGB color model	4
HSL color model	6
Converting HSL values to RGB values	6
Using the Color dialog box to display basic colors	7
Initializing the CHOOSECOLOR structure	7
Calling the ChooseColor function	8
Using the Color dialog box to display custom colors	8
Initializing the CHOOSECOLOR structure	8
Calling the ChooseColor function	9
Using Font dialog boxes	11
Displaying the Font dialog box in your application	11
Using Open and Save As dialog boxes	13
Displaying the Open dialog box in your application	13
Displaying the Save As dialog box in your application	16
Monitoring list box controls in an Open or Save As dialog box	18
Monitoring filenames in an Open or Save As dialog box	19
Using Print and Print Setup dialog boxes	20
Device drivers and the Print dialog box	21
Displaying a Print dialog box for the default printer	21
Using Find and Replace dialog boxes	23
Displaying the Find dialog box	23
Displaying the Replace dialog box	25
Processing dialog box messages for a Find or Replace dialog box	26
Customizing common dialog boxes	27
Appropriate and inappropriate customizations	27
Hook functions and custom dialog box templates	28
Hook function	28
Customizing a dialog box template	31
Displaying the custom dialog box	32
Supporting help for the common dialog boxes	34
Error detection	35
 Chapter 2 Dynamic Data Exchange Management Library	 37
Basic concepts	38
Client and server interaction	39
Transactions and the DDE callback function	39
Service names, topic names, and item names	40
System topic	40
Initialization	42

Callback function	43	Server applications	80
String management	45	Object handlers	80
Name service	47	Communication between OLE libraries	81
Service-name registration	47	Clipboard conventions	81
Service-name filter	48	Registration	85
Conversation management	48	Registration database	85
Single conversations	48	Version control for servers	87
Multiple conversations	52	Client user interface	88
Data management	54	New and changed commands	88
Transaction management	57	Using packages	91
Request transaction	57	Server user interface	92
Poke transaction	58	Updating objects from multiple-instance servers	92
Advise transaction	59	Updating objects from single-instance servers	93
Execute transaction	60	Object storage formats	93
Synchronous and asynchronous transactions	61	Client applications	95
Transaction control	62	Starting a client application	96
Transaction classes	63	Opening a compound document	97
Transaction summary	64	Document management	98
Error detection	66	Saving a document	99
Monitoring applications	66	Closing a document	99
Chapter 3 Object linking and embedding libraries 71		Asynchronous operations	99
Basics of object linking and embedding ..	71	Displaying and printing objects	102
Compound documents	72	Opening and closing objects	102
Linked and embedded objects	73	Deleting objects	103
Packages	74	Client Cut and Copy commands	103
Verbs	74	Creating objects	105
Benefits of object linking and embedding	75	Object-creation functions	105
Choosing between OLE and the DDEML	76	Paste and Paste Link commands .	107
Using OLE for standard DDE operations	77	Undo command	108
Using both OLE and the DDEML	79	Class Name Object command	109
Data transfer in object linking and embedding	79	Links command	109
Client applications	80	Closing a client application	110
		Server applications	111
		Starting a server application	112
		Opening a document or object	114

Server Cut and Copy commands	115	ChooseFont	147
Update, Save As, and New commands	116	ClassFirst	148
Closing a server application	117	ClassNext	149
Object handlers	119	CloseDriver	150
Implementing object handlers	119	CommDlgExtendedError	151
Creating objects in an object handler	122	CopyCursor	154
DefCreateFromClip and DllCreateFromClip	122	CopyIcon	155
DefLoadFromStream and DllLoadFromStream	123	CopyLZFile	155
Direct use of Dynamic Data Exchange	124	CPIApplet	157
Client applications and direct use of Dynamic Data Exchange	124	CreateScalableFontResource	157
Server applications and direct use of Dynamic Data Exchange	127	DdeAbandonTransaction	160
Conversations	128	DdeAccessData	162
Items for the system topic	128	DdeAddData	163
Standard item names and notification control	129	DdeCallback	165
Standard commands in DDE execute strings	131	DdeClientTransaction	167
International execute commands	131	DdeCmpStringHandles	170
Required commands	132	DdeConnect	172
Variants on required commands	134	DdeConnectList	174
Chapter 4 Functions	135	DdeCreateDataHandle	176
AbortDoc	135	DdeCreateStringHandle	179
AbortProc	136	DdeDisconnect	181
AllocDiskSpace	136	DdeDisconnectList	181
AllocFileHandles	137	DdeEnableCallback	182
AllocGDIMem	138	DdeFreeDataHandle	183
AllocMem	139	DdeFreeStringHandle	185
AllocUserMem	139	DdeGetData	186
CallNextHookEx	140	DdeGetLastError	187
CallWndProc	140	DdeInitialize	190
CBTProc	141	DdeKeepStringHandle	194
ChooseColor	145	DdeNameService	195
		DdePostAdvise	197
		DdeQueryConvInfo	199
		DdeQueryNextServer	200
		DdeQueryString	202
		DdeReconnect	203
		DdeSetUserHandle	204
		DdeUnaccessData	205
		DdeUninitialize	206

DebugOutput	207
DebugProc	208
DefDriverProc	209
DirectedYield	210
DlgDirSelectComboBoxEx	211
DlgDirSelectEx	212
DragAcceptFiles	213
DragFinish	213
DragQueryFile	214
DragQueryPoint	214
DriverProc	215
EnableCommNotification	217
EnableScrollBar	218
EndDoc	220
EndPage	220
EnumFontFamilies	221
EnumFontFamProc	222
EnumFontsProc	225
EnumMetaFileProc	227
EnumObjectsProc	228
EnumPropFixedProc	230
EnumPropMovableProc	231
EnumTaskWndProc	232
EnumWindowsProc	232
ExitWindowsExec	233
ExtractIcon	234
FindExecutable	234
FindText	236
FMExtensionProc	238
FreeAllGDIMem	239
FreeAllMem	240
FreeAllUserMem	240
GetAspectRatioFilterEx	240
GetBitmapDimensionEx	241
GetBoundsRect	241
GetBrushOrgEx	243
GetCharABCWidths	243
GetClipCursor	244

GetCurrentPositionEx	245
GetCursor	245
GetDCEx	246
GetDriverInfo	247
GetDriverModuleHandle	248
GetExpandedName	249
GetFileResource	250
GetFileResourceSize	251
GetFileTitle	252
GetFileVersionInfo	253
GetFileVersionInfoSize	254
GetFontData	254
GetFreeFileHandles	257
GetFreeSystemResources	257
GetGlyphOutline	258
GetKerningPairs	260
GetMessageExtraInfo	261
GetMsgProc	261
GetNextDriver	262
GetOpenClipboardWindow	263
GetOpenFileName	264
GetOutlineTextMetrics	266
GetQueueStatus	268
GetRasterizerCaps	269
GetSaveFileName	270
GetSelectorBase	272
GetSelectorLimit	273
GetSystemDebugState	273
GetSystemDir	274
GetTextExtentPoint	275
GetTimerResolution	276
GetViewportExtEx	276
GetViewportOrgEx	276
GetWinDebugInfo	277
GetWindowExtEx	278
GetWindowOrgEx	278
GetWindowPlacement	278
GetWindowsDir	279

GetWinMem32Version	280	LocalFirst	314
Global16PointerAlloc	281	LocalInfo	315
Global16PointerFree	282	LocalNext	315
Global32Alloc	283	LockInput	316
Global32CodeAlias	285	LockWindowUpdate	317
Global32CodeAliasFree	286	LogError	318
Global32Free	287	LogParamError	319
Global32Realloc	287	LZClose	322
GlobalEntryHandle	289	LZCopy	323
GlobalEntryModule	290	LZDone	324
GlobalFirst	291	LZInit	325
GlobalHandleToSel	292	LZOpenFile	327
GlobalInfo	293	LZRead	329
GlobalNext	293	LZSeek	331
GrayStringProc	295	LZStart	332
HardwareProc	295	MapWindowPoints	333
hardware_event	296	MemManInfo	334
hmemcpy	297	MemoryRead	335
_hread	298	MemoryWrite	336
_hwrite	298	MessageProc	337
InterruptRegister	299	ModuleFindHandle	338
Low-stack Faults	301	ModuleFindName	339
InterruptUnRegister	302	ModuleFirst	340
IsBadCodePtr	303	ModuleNext	341
IsBadHugeReadPtr	304	MouseProc	342
IsBadHugeWritePtr	304	MoveToEx	343
IsBadReadPtr	305	NotifyProc	343
IsBadStringPtr	305	NotifyRegister	344
IsBadWritePtr	306	NotifyUnRegister	347
IsGDIObject	306	OffsetViewportOrgEx	347
IsMenu	307	OffsetWindowOrgEx	348
IsTask	307	OleActivate	349
JournalPlaybackProc	307	OleBlockServer	350
JournalRecordProc	309	OleClone	351
KeyboardProc	310	OleClose	352
LibMain	311	OleCopyFromLink	352
LineDDAProc	312	OleCopyToClipboard	353
LoadProc	313	OleCreate	354

OleCreateFromClip	355	OleRenameClientDoc	395
OleCreateFromFile	357	OleRenameServerDoc	396
OleCreateFromTemplate	360	OleRequestData	396
OleCreateInvisible	362	OleRevertClientDoc	397
OleCreateLinkFromClip	364	OleRevertServerDoc	398
OleCreateLinkFromFile	366	OleRevokeClientDoc	399
OleDelete	368	OleRevokeObject	399
OleDraw	369	OleRevokeServer	400
OleEnumFormats	370	OleRevokeServerDoc	400
OleEnumObjects	371	OleSavedClientDoc	401
OleEqual	372	OleSavedServerDoc	402
OleExecute	372	OleSaveToStream	402
OleGetData	373	OleSetBounds	403
OleGetLinkUpdateOptions	374	OleSetColorScheme	404
OleIsDcMeta	375	OleSetData	405
OleLoadFromStream	376	OleSetHostNames	406
OleLockServer	377	OleSetLinkUpdateOptions	407
OleObjectConvert	378	OleSetTargetDevice	408
OleQueryBounds	379	OleUnblockServer	409
OleQueryClientVersion	380	OleUnlockServer	410
OleQueryCreateFromClip	380	OleUpdate	411
OleQueryLinkFromClip	382	OpenDriver	411
OleQueryName	383	PrintDlg	412
OleQueryOpen	384	QueryAbort	415
OleQueryOutOfDate	384	QuerySendMessage	415
OleQueryProtocol	385	RedrawWindow	416
OleQueryReleaseError	386	RegCloseKey	419
OleQueryReleaseMethod	386	RegCreateKey	420
OleQueryReleaseStatus	388	RegDeleteKey	422
OleQueryServerVersion	388	RegEnumKey	423
OleQuerySize	389	RegOpenKey	424
OleQueryType	389	RegQueryValue	425
OleReconnect	390	RegSetValue	426
OleRegisterClientDoc	390	ReplaceText	427
OleRegisterServer	391	ResetDC	430
OleRegisterServerDoc	393	ScaleViewPortExtEx	431
OleRelease	394	ScaleWindowExtEx	432
OleRename	394	ScrollWindowEx	432

SendDriverMessage	435	UnhookWindowsHookEx	474
SetAbortProc	435	VerFindFile	475
SetBitmapDimensionEx	436	VerInstallFile	478
SetBoundsRect	437	VerLanguageName	482
SetMetaFileBitsBetter	438	VerQueryValue	484
SetSelectorBase	439	WindowProc	487
SetSelectorLimit	439	WNetAddConnection	488
SetViewportExtEx	439	WNetCancelConnection	489
SetViewportOrgEx	440	WNetGetConnection	489
SetWinDebugInfo	442	WordBreakProc	490
SetWindowExtEx	443		
SetWindowOrgEx	444	Chapter 5 Data types	493
SetWindowPlacement	444		
SetWindowsHookEx	445	Chapter 6 Messages	503
ShellExecute	448	CB_ADDSTRING	503
ShellProc	451	CB_DELETESTRING	504
SpoolFile	452	CB_FINDSTRINGEXACT	505
StackTraceCSIPFirst	453	CB_GETDROPPEDCONTROLRECT	505
StackTraceFirst	454	CB_GETDROPPEDSTATE	506
StackTraceNext	455	CB_GETEXTENDEDUI	507
StartDoc	456	CB_GETITEMHEIGHT	507
StartPage	457	CB_SETEXTENDEDUI	508
SubtractRect	457	CB_SETITEMHEIGHT	509
SysMsgProc	458	EM_GETFIRSTVISIBLELINE	510
SystemHeapInfo	459	EM_GETPASSWORDCHAR	510
SystemParametersInfo	460	EM_GETWORDBREAKPROC	511
TaskFindHandle	466	EM_SETREADONLY	511
TaskFirst	467	EM_SETWORDBREAKPROC	512
TaskGetCSIP	468	LB_FINDSTRINGEXACT	513
TaskNext	468	LB_GETCARETINDEX	514
TaskSetCSIP	469	LB_SETCARETINDEX	514
TaskSwitch	470	STM_GETICON	515
TerminateApp	470	STM_SETICON	515
TimerCount	471	WM_CHOOSEFONT_GETLOG- FONT	516
TimerProc	472	WM_COMMNOTIFY	516
UnAllocDiskSpace	473	WM_DDE_ACK	517
UnAllocFileHandles	473	Posting	519
UndeleteFile	474	Receiving	519

WM_DDE_ADVISE	520
Posting	520
Receiving	521
WM_DDE_DATA	521
Posting	522
Receiving	522
WM_DDE_EXECUTE	523
Posting	524
Receiving	524
WM_DDE_INITIATE	524
Sending	525
Receiving	526
WM_DDE_POKE	526
Posting	527
Receiving	527
WM_DDE_REQUEST	527
Posting	528
Receiving	528
WM_DDE_TERMINATE	528
Posting	528
Receiving	529
WM_DDE_UNADVISE	529
Posting	529
Receiving	530
WM_DROPFILES	530
WM_PALETTEISCHANGING	530
WM_POWER	531
WM_QUEUESYNC	532
WM_SYSTEMERROR	532
WM_USER	533
WM_WINDOWPOSCHANGED	534
WM_WINDOWPOSCHANGING ...	534
Notification messages	536
BN_HILITE	536
BN_PAINT	536
BN_UNHILITE	536
CBN_CLOSEUP	537
CBN_SELENDNCANCEL	537

CBN_SELENDOK	538
LBN_SELCANCEL	538

Chapter 7 Structures 539

ABC	539
CBT_CREATEWND	540
CBTACTIVATESTRUCT	540
CHOOSECOLOR	541
CHOOSEFONT	544
CLASSENTRY	551
COMSTAT	552
CONVCONTEXT	553
CONVINFO	554
CPLINFO	557
CTLINFO	558
CTLSTYLE	559
CTLTYPE	561
DDEACK	562
DDEADVISE	563
DDEDATA	564
DDEPOKE	565
DEBUGHOOKINFO	566
DEVNAMES	567
DOCINFO	568
DRIVERINFOSTRUCT	569
DRVCONFIGINFO	569
EVENTMSG	570
FINDREPLACE	571
FIXED	575
FMS_GETDRIVEINFO	576
FMS_GETFILESEL	577
FMS_LOAD	578
GLOBALENTRY	579
GLOBALINFO	582
GLYPHMETRICS	583
HARDWAREHOOKSTRUCT	584
HELPWININFO	584
HSZPAIR	585

KERNINGPAIR	586	Return Value	625
LOCALENTRY	587	Comments	625
LOCALINFO	590	Parameters	626
MAT2	591	Return Value	626
MEMMANINFO	592	Parameters	626
METAHEADER	593	Return Value	626
METARECORD	594	Comments	627
MINMAXINFO	595	Parameters	627
MODULEENTRY	596	Return Value	627
MONCBSTRUCT	597	Comments	627
MONCONVSTRUCT	598	See Also	627
MONERRSTRUCT	599	Parameters	627
MONHSZSTRUCT	600	Return Value	628
MONLINKSTRUCT	602	Parameters	628
MONMSGSTRUCT	603	Return Value	628
MOUSEHOOKSTRUCT	604	Comments	629
NCCALCSIZE_PARAMS	605	OLESERVER	629
NEWCPINFO	606	OLESERVERDOC	630
NEWTEXTMETRIC	607	OLESERVERDOCVTBL	630
NFYLOADSEG	612	Parameters	631
NFYLOGERROR	613	Return Value	631
NFYLOGPARAMERROR	614	Parameters	631
NFYRIP	615	Return Value	632
NFYSTARTDLL	616	Comments	632
OLECLIENT	617	Parameters	632
OLECLIENTVTBL	617	Return Value	633
Parameters	618	Parameters	633
Return Value	619	Return Value	633
Comments	620	Parameters	633
OLEOBJECT	620	Return Value	634
OLEOBJECTVTBL	621	Comments	634
Parameters	624	Parameters	634
Return Value	624	Return Value	634
Comments	624	Parameters	635
Parameters	624	Return Value	635
Return Value	625	Comments	635
Comments	625	Parameters	636
Parameters	625	Return Value	636

Comments	636
OLESERVERVTBL	636
Parameters	637
Return Value	637
Comments	637
Parameters	638
Return Value	638
Comments	638
Parameters	639
Return Value	639
Comments	639
Parameters	640
Return Value	640
Comments	640
Parameters	641
Return Value	641
Comments	641
Parameters	641
Return Value	641
Comments	641
Parameters	642
Return Value	642
Comments	642
OLESTREAM	643
OLESTREAMVTBL	643
Parameters	644
Return Value	644
Comments	644
Parameters	644
Return Value	645
Comments	645
OLETARGETDEVICE	645
OPENFILENAME	646
OUTLINETEXTMETRIC	655
PANOSE	659
POINTFX	664
PRINTDLG	664
RASTERIZER_STATUS	673

SEGINFO	673
SIZE	675
STACKTRACEENTRY	676
SYSHEAPINFO	677
TASKENTRY	678
TIMERINFO	679
TTPOLYCURVE	680
TTPOLYGONHEADER	681
VS_FIXEDFILEINFO	682
WINDEBUGINFO	686
WINDOWPLACEMENT	690
WINDOWPOS	692

Chapter 8 Macros 695

DECLARE_HANDLE	695
DECLARE_HANDLE32	695
FIELDOFFSET	696
GetBValue	696
GetGValue	697
GetRValue	697
MAKELP	697
MAKELPARAM	698
MAKELRESULT	698
OFFSETOF	699
SELECTOROF	699

Chapter 9 Printer escapes 701

MOUSETRAILS	701
POSTSCRIPT_DATA	702
POSTSCRIPT_IGNORE	702
SETALLJUSTVALUES	703

Chapter 10 Dynamic Data Exchange transactions 705

XTYP_ADVDATA	705
XTYP_ADVREQ	706
XTYP_ADVSTART	707
XTYP_ADVSTOP	708
XTYP_CONNECT	708

XTYP_CONNECT_CONFIRM	709
XTYP_DISCONNECT	710
XTYP_ERROR	711
XTYP_EXECUTE	711
XTYP_MONITOR	712
XTYP_POKE	713
XTYP_REGISTER	714
XTYP_REQUEST	715
XTYP_UNREGISTER	715
XTYP_WILDCONNECT	716
XTYP_XACT_COMPLETE	717
Chapter 11 Common dialog box messages	719
COLOROKSTRING	719
FILEOKSTRING	720
FINDMSGSTRING	721
HELPMMSGSTRING	721
LBSELCHSTRING	722
SETRGBSTRING	723
SHAREVISTRING	723
Index	725

Common dialog box library

Common dialog boxes make it easier for you to develop applications for the Microsoft Windows operating system. A common dialog box is a dialog box that an application displays by calling a single function rather than by creating a dialog box procedure and a resource file containing a dialog box template. The dynamic-link library `COMMDLG.DLL` provides a default procedure and template for each type of common dialog box. Each default dialog box procedure processes messages and notifications for a common dialog box and its controls. A default dialog box template defines the appearance of a common dialog box and its controls.

In addition to simplifying the development of Windows applications, a common dialog box assists users by providing a standard set of controls for performing certain operations. As Windows developers begin using the common dialog boxes in their applications, users will find that after they master using a common dialog box in one application, they can easily perform the same operations in other applications.

This chapter describes the various common dialog boxes and includes sample code to help you use common dialog boxes in your Windows applications.

Following are the types of common dialog boxes in the order in which they are presented in this chapter:

Name	Description
Color	Displays available colors, from which the user can select one; displays controls that let the user define a custom color.
Font	Displays lists of fonts, point sizes, and colors that correspond to available fonts; after the user selects a font, the dialog box displays sample text rendered with that font.
Open	Displays a list of filenames matching any specified extensions, directories, and drives. By selecting one of the listed filenames, the user indicates which file an application should open.
Save As	Displays a list of filenames matching any specified extensions, directories, and drives. By selecting one of the listed filenames, the user indicates which file an application should save.
Print	Displays information about the installed printer and its configuration. By altering and selecting controls in this dialog box, the user specifies how output should be printed and starts the printing process.
Print Setup	Displays the current list of available printers. The user can select a printer from this list. This common dialog box also provides options for setting the paper orientation, size, and source (when the printer driver supports these options). In addition to being called directly, the Print Setup dialog can be opened from within the Print dialog.
Find	Displays an edit control in which the user can type a string for which the application should search. The user can specify the direction of the search, whether the application should match the case of the specified string, and whether the string to match is an entire word.
Replace	Displays two edit controls in which the user can type strings: the first string identifies a word or value that the application should replace, and the second string identifies the replacement word or value.

Applications that use the common dialog boxes should specify at least 8K for the stack size, as shown in the following example:

```
NAME cd

EXETYPEWINDOWS

STUB    'WINSTUB.EXE'

CODE    PRELOAD MOVEABLE DISCARDABLE

DATA    PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 1024

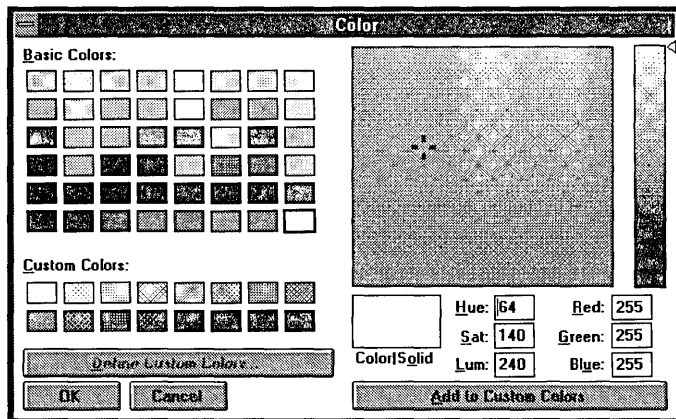
STACKSIZE8192

EXPORTS
    FILEOPENHOOKPROC @1
```

Using Color dialog boxes

The Color dialog box contains controls that make it possible for a user to select and create colors.

Following is a Color dialog box.



The Basic Colors control displays up to 48 colors. The actual number of colors displayed is determined by the display driver. For example, a VGA driver displays 48 colors, and a monochrome

display driver displays only 16. With the Basic Colors control, the user can select a displayed color.

To display the Custom Colors control, the user clicks the Define Custom Colors button. The Custom Colors control displays custom colors. The user can select one of the 16 rectangles in this control and then create a new color by using one of the following methods:

- Specifying red, green, and blue (RGB) values by using the Red, Green, and Blue edit controls, and then choosing the Add to Custom Colors button to display the new color in the selected rectangle.
- Moving the cursor in the color spectrum control (at the upper-right of the dialog box) to select hue and saturation values; moving the cursor in the luminosity control (the rectangle to the right of the spectrum control); and then choosing the Add to Custom Colors button to display the new color in the selected rectangle.
- Specifying hue, saturation, and luminosity (HSL) values by using the Hue, Sat, and Lum edit controls and then choosing the Add to Custom Colors button to display the new color in the selected rectangle.

The Color | Solid control displays the dithered and solid colors that correspond to the user's selection. (A dithered color is a color created by combining one or more pure or solid colors.) The **Flags** member of the **CHOOSECOLOR** structure contains a flag bit that, when set, displays a Help button.

An application can display the Color dialog box in one of two ways: fully open or partially open. When the Color dialog box is displayed partially open, the user cannot change the custom colors.

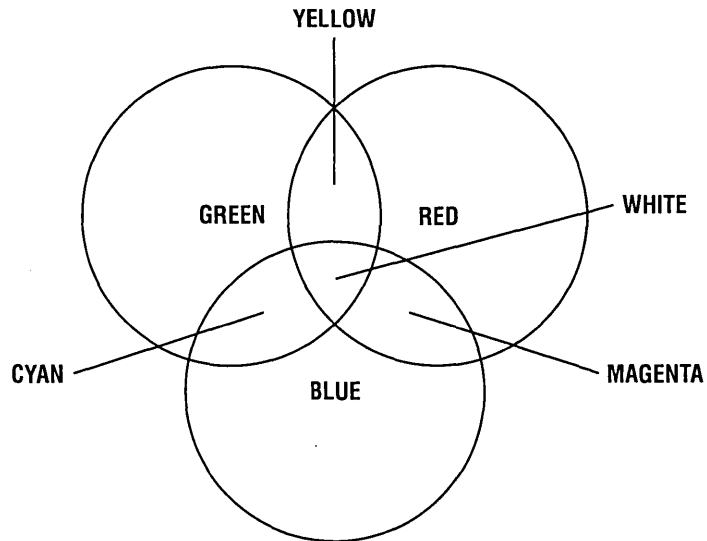
Color models used by the Color dialog box

The Color dialog box uses two models for specifying colors: the RGB model and the HSL model. Regardless of the model used, internal storage is accomplished by use of the RGB model.

RGB color model

The RGB model is used to designate colors for displays and other devices that emit light. Valid red, green, and blue values are in the range 0 through 255, with 0 indicating the minimum intensity

and 255 indicating the maximum intensity. The following illustration shows how the primary colors red, green, and blue can be combined to produce four additional colors. (With display devices, the color black results when the red, green, and blue values are set to 0—that is, with display technology, black is the absence of all colors.)

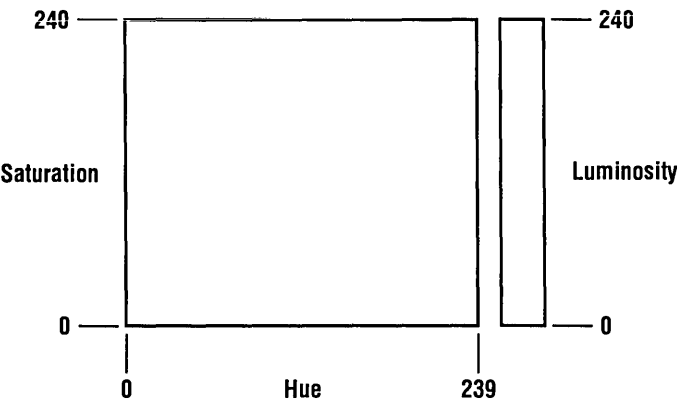


Following are eight colors and their associated RGB values:

Color	RGB values
Red	255, 0, 0
Green	0, 255, 0
Blue	0, 0, 255
Cyan	0, 255, 255
Magenta	255, 0, 255
Yellow	255, 255, 0
White	255, 255, 255
Black	0, 0, 0

Windows stores internal colors as 32-bit RGB values. The high-order byte of the high-order word is reserved; the low-order byte of the high-order word specifies the intensity of the blue component; the high-order byte of the low-order word specifies the intensity of the green component; and the low-order byte of the low-order word specifies the intensity of the red component.

HSL color model The Color dialog box provides controls for specifying HSL values. The following illustration shows the color spectrum control and the vertical luminosity control that appear in the Color dialog box and shows the ranges of values the user can specify with these controls.



In the Color dialog box, the saturation and luminosity values must be in the range 0 through 240 and the hue value must be in the range 0 through 239.

Converting HSL values to RGB values The dialog box procedure provided in COMMDLG.DLL for the Color dialog box contains code that converts HSL values to the corresponding RGB values. Following are several colors with their associated HSL and RGB values:

Color	HSL values	RGB values
Red	(0, 240, 120)	(255, 0, 0)
Yellow	(40, 240, 120)	(255, 255, 0)
Green	(80, 240, 120)	(0, 255, 0)
Cyan	(120, 240, 120)	(0, 255, 255)
Blue	(160, 240, 120)	(0, 0, 255)
Magenta	(200, 240, 120)	(255, 0, 255)
White	(0, 0, 240)	(255, 255, 255)
Black	(0, 0, 0)	(0, 0, 0)

Using the Color dialog box to display basic colors

An application can display the Color dialog box so that a user can select one color from a list of basic screen colors. This section describes how you can provide code and structures in your application that make this possible.

Initializing the CHOOSECOLOR structure

Before you display the Color dialog box you need to initialize a **CHOOSECOLOR** structure. This structure should be global or declared as a **static** variable. The members of this structure contain information about such items as the following:

- Structure size
- Which window owns the dialog box
- Whether the application is customizing the common dialog box
- The hook function and custom dialog box template to use for a customized version of the Color dialog box
- RGB values for the selected basic color

If your application does not customize the dialog box and you want the user to be able to select a single color from the basic colors, you should initialize the **CHOOSECOLOR** structure in the following manner:

```
/* Color variables */

CHOOSECOLOR cc;
COLORREF clr;
COLORREF acclrCust[16];
int i;

/* Set the custom color controls to white. */

for (i = 0; i < 16; i++)
    acclrCust[i] = RGB(255, 255, 255);

/* Initialize clr to black. */

clr = RGB(0, 0, 0);

/* Set all structure fields to zero. */

memset(&cc, 0, sizeof(CHOOSECOLOR));

/* Initialize the necessary CHOOSECOLOR members. */

cc.lStructSize = sizeof(CHOOSECOLOR);
cc.hwndOwner = hwnd;
cc.rgbResult = clr;
cc.lpCustColors = acclrCust;
```

```
cc.Flags = CC_PREVENTFULOPEN;

if (ChooseColor(&cc))
{
    /* Use cc.rgbResult to select the user-requested color. */
}
```

In the previous example, the array to which the **lpCustColors** member points contains 16 doubleword RGB values that specify the color white, and the **CC_PREVENTFULOPEN** flag is set in the **Flags** member to disable the Define Custom Colors button and prevent the user from selecting a custom color.

Calling the
ChooseColor function

After you initialize the structure, you should call the **ChooseColor** function. If the function is successful and the user chooses the OK button to close the dialog box, the **rgbResult** member contains the RGB values for the basic color that the user selected.

Using the Color dialog box to display custom colors

An application can display the Color dialog box so that the user can create and select a custom color. This section describes how you can provide code and structures in your application that make this possible.

Initializing the
CHOOSECOLOR
structure

Before you display the Color dialog box, you need to initialize a **CHOOSECOLOR** structure. This structure should be global or declared as a **static** variable. The members of this structure contain information about such items as the following:

- Structure size
- Which window owns the dialog box
- Whether the application is customizing the common dialog box
- The hook function and custom dialog box template to use for a customized version of the Color dialog box
- RGB values for the custom color control

If your application does not customize the dialog box and you want the user to be able to create and select custom colors, you should initialize the **CHOOSECOLOR** structure in the following manner:

```
/* Color Variables */

CHOOSECOLOR chsclr;
DWORD dwCustClrs[16] = { RGB(255, 255, 255), RGB(239, 239, 239),
                        RGB(223, 223, 223), RGB(207, 207, 207),
                        RGB(191, 191, 191), RGB(175, 175, 175),
                        RGB(159, 159, 159), RGB(143, 143, 143),
                        RGB(127, 127, 127), RGB(111, 111, 111),
                        RGB(95, 95, 95),   RGB(79, 79, 79),
                        RGB(63, 63, 63),   RGB(47, 47, 47),
                        RGB(31, 31, 31),   RGB(15, 15, 15)
                      };
BOOL fSetColor = FALSE;
int i;

chsclr.lStructSize = sizeof (CHOOSECOLOR);
chsclr.hwndOwner = hwnd;
chsclr.hInstance = NULL;
chsclr.rgbResult = 0L;
chsclr.lpCustColors = (LPDWORD) dwCustClrs;
chsclr.Flags = CC_FULLOPEN;
chsclr.lCustData = 0L;
chsclr.lpfnHook = (FARPROC) NULL;
chsclr.lpTemplateName = (LPSTR) NULL;
```

In the previous example, the array to which **lpCustColors** points contains sixteen 32-bit RGB values that specify 16 scales of gray, and the **CC_FULLOPEN** flag is set in the **Flags** member to display the complete Color dialog box.

Calling the
ChooseColor function

After you initialize the structure, you should call the **ChooseColor** function as shown in the following code fragment:

```
if (fSetColor = ChooseColor(&chsclr))
{
    /* Use chsclr.lpCustColors to select user specified colors */
}
```

If the function is successful and the user chooses the OK button to close the dialog box, the **lpCustColors** member points to an array that contains the RGB values for the custom colors requested by the application's user.

Applications can exercise more control over custom colors by creating a new message identifier for the string defined by the **COLOROKSTRING** constant. The application creates the new message identifier by calling the **RegisterWindowMessage**

function and passing this constant as the single parameter. After calling **RegisterWindowMessage**, the application receives a message immediately prior to the dismissal of the dialog box. The *lParam* parameter of this message contains a pointer to the **CHOOSECOLOR** structure. The application can use the **lpCustColors** member of this structure to check the current color. If the application returns a nonzero value when it processes this message, the dialog box is not dismissed.

Similarly, applications can create a new message identifier for the string defined by the **SETRGBSTRING** constant. The application's hook function can use the message identifier returned by calling **RegisterWindowMessage** with the **SETRGBSTRING** constant to set a color in the dialog box. For example, the following line of code sets the color selection to blue:

```
SendMessage (hwhndDlg, wSetRGBMsg, 0, (LPARAM) RGB (0, 0, 255) );
```

In this example, *wSetRGBMsg* is the message identifier returned by the **RegisterWindowMessage** function. The *lParam* parameter of the **SendMessage** function is set to the RGB values of the desired color. The *wParam* parameter is not used.

The application can specify any valid RGB values in this call to **SendMessage**. If the RGB values match one of the basic colors, the system selects the basic color and updates the spectrum and luminosity controls. If the RGB values do not match one of the basic colors, the system updates the spectrum and luminosity controls, but the basic color selection remains unchanged.

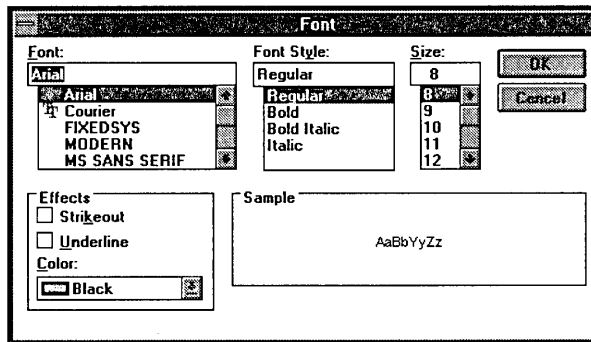
Note that if the Color dialog box is not fully open and the application sends RGB values that do not match one of the basic colors, the system does not update the dialog box. Updates are unnecessary because the spectrum and luminosity controls are not visible when the dialog box is only partially open.

For more information about processing registered window messages, see "Using Find and Replace dialog boxes."

Using Font dialog boxes

The Font dialog box contains controls that make it possible for a user to select a font, a font style (such as bold, italic, or regular), a point size, and an effect (such as underline, strikeout, or a text color).

Following is a Font dialog box.



Displaying the Font dialog box in your application

The Font dialog box appears after you initialize the members in a **CHOOSEFONT** structure and call the **ChooseFont** function. This structure should be global or declared as a **static** variable. The members of the **CHOOSEFONT** structure contain information about such items as the following:

- The attributes of the font that initially is to appear in the dialog box.
- The attributes of the font that the user selected.
- The point size of the font that the user selected.
- Whether the list of fonts corresponds to a printer, a screen, or both.
- Whether the available fonts listed are TrueType only.
- Whether the Effects box should appear in the dialog box.
- Whether dialog box messages should be processed by an application-supplied hook function.

- Whether the point sizes of the selectable fonts should be limited to a specified range.
- Whether the dialog box should display only what-you-see-is-what-you-get (WYSIWIG) fonts. (These fonts are resident on both the screen and the printer.)
- The color that the **ChooseFont** function should use to render text in the Sample box the first time the application displays the dialog box.
- The color that the user selected for text output.

To display the Font dialog box, an application should perform the following steps:

1. If the application requires printer fonts, retrieve a device-context handle for the printer and use this handle to set the **hDC** member of the **CHOOSEFONT** structure. (If the Font dialog box displays only screen fonts, this member should be set to **NULL**.)
2. Set the appropriate flags in the **Flags** member of the **CHOOSEFONT** structure. This setting must include **CF_SCREENFONTS**, **CF_PRINTERFONTS**, or **CF_BOTH**.
3. Set the **rgbColors** member of the **CHOOSEFONT** structure if the default color (black) is not appropriate.
4. Set the **nFontType** member of the **CHOOSEFONT** structure using the appropriate constant.
5. Set the **nSizeMin** and **nSizeMax** members of the **CHOOSEFONT** structure if the **CF_LIMITSIZE** value is specified in the **Flags** member.
6. Call the **ChooseFont** function.

The following example initializes the **CHOOSEFONT** structure and calls the **ChooseFont** function:

```
LOGFONT lf;
CHOOSEFONT cf;

/* Set all structure fields to zero. */

memset(&cf, 0, sizeof(CHOOSEFONT));

cf.lStructSize = sizeof(CHOOSEFONT);
cf.hwndOwner = hwnd;
cf.lpLogFont = &lf;
cf.Flags = CF_SCREENFONTS | CF_EFFECTS;
cf.rgbColors = RGB(0, 255, 255); /* light blue */
```

```
cf.nFontType = SCREEN_FONTTYPE;

ChooseFont (&cf);
```

When the user closes the Font dialog box by choosing the OK button, the **ChooseFont** function returns information about the selected font in the **LOGFONT** structure to which the **lpLogFont** member points. An application can use this **LOGFONT** structure to select the font that will be used to render text. The following example selects a font by using the **LOGFONT** structure and renders a string of text:

```
hdc = GetDC (hwnd);
hFont = CreateFontIndirect (cf.lpLogFont);
hFontOld = SelectObject (hdc, hFont);
TextOut (hdc, 50, 150,
        "AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz", 52);
SelectObject (hdc, hFontOld);
DeleteObject (hFont);
ReleaseDC (hwnd, hdc);
```

An application can also use the **WM_CHOOSEFONT_GETLOGFONT** message to retrieve the current **LOGFONT** structure for the Font dialog box before the user closes the dialog box.

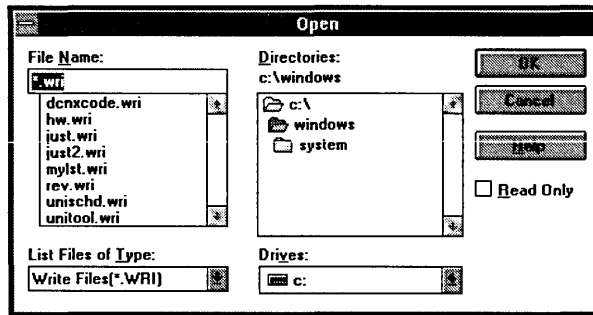
Using Open and Save As dialog boxes

The Open dialog box and the Save As dialog box are similar in appearance. Each contains controls that make it possible for the user to specify the location and name of a file or set of files. In the case of the Open dialog box, the user selects the file or files to be opened; in the case of the Save As dialog box, the user selects the file or files to be saved.

Displaying the Open dialog box in your application

The Open dialog box appears after you initialize the members of an **OPENFILENAME** structure and call the **GetOpenFileName** function.

Following is an Open dialog box.



Before the call to **GetOpenFileName**, structure members contain such data as the name of the directory and the filter that are to appear in the dialog box. (A filter is a filename extension. The common dialog box code uses the extension to filter appropriate filenames from a directory.) After the call, structure members contain such data as the name of the selected file and the number of characters in that filename.

To display an Open dialog box, an application should perform the following steps:

1. Store the valid filters in a character array.
2. Set the **lpstrFilter** member to point to this array.
3. Set the **nFilterIndex** member to the value of the index that identifies the default filter.
4. Set the **lpstrFile** member to point to an array that contains the initial filename and receives the selected filename.
5. Set the **nMaxFile** member to the value that specifies the length of the filename array.
6. Set the **lpstrFileName** member to point to a buffer that receives the title of the selected file.
7. Set the **nMaxFileName** member to specify the length of the buffer.
8. Set the **lpstrInitialDir** member to point to a string that specifies the initial directory. (If this member does not point to a valid string, it must be set to 0 or point to a string that is set to NULL.)

9. Set the **lpstrTitle** member to point to a string specifying the name that should appear in the title bar of the dialog box. (If this pointer is NULL, the title will be Open.)
10. Initialize the **lpstrDefExt** member to point to the default extension. (This extension can be 0, 1, 2, or 3 characters long.)
11. Call the **GetOpenFileName** function.

The following example initializes an **OPENFILENAME** structure, calls the **GetOpenFileName** function, and opens the file by using the **lpstrFile** member of the structure. The **OPENFILENAME** structure should be global or declared as a **static** variable.

```
OPENFILENAME ofn;
char szDirName[256];
char szFile[256], szFileTitle[256];
UINT i, cbString;
char chReplace; /* string separator for szFilter */
char szFilter[256];
HFILE hf;

/* Get the system directory name, and store in szDirName. */

GetSystemDirectory(szDirName, sizeof(szDirName));
szFile[0] = '\0';

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
    szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0L;
}
chReplace = szFilter[cbString - 1]; /* retrieve wildcard */

for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
        szFilter[i] = '\0';
}

/* Set all structure members to zero. */

memset(&ofn, 0, sizeof(OPENFILENAME));

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.nFilterIndex = 1;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_PATHMUSTEXIST |
    OFN_FILEMUSTEXIST;
```

```

if(GetOpenFileName(&ofn)){
    hf = _lopen(ofn.lpstrFile, OF_READ);
    .
    . /* Perform file operations. */
    .
}
else
    ErrorHandler();

```

The string referred to by the `IDS_FILTERSTRING` constant in the preceding example is defined as follows in the resource-definition file:

```

STRINGTABLE
BEGIN
    IDS_FILTERSTRING  "Write Files(*.WRI)|*.wri|Word Files(*.DOC)|*.doc|"
END

```

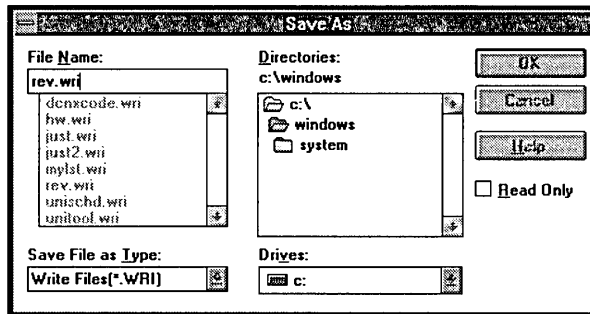
The vertical bars in this string are used as wildcards. After using the **LoadString** function to retrieve the string, the wildcards are replaced with NULL. The wildcard can be any unique character and must be included as the last character in the string. Initializing strings in this manner guarantees that the parts of the string are contiguous in memory and that the string is terminated with two null characters.

Applications that can open files over a network can create a new message identifier for the string defined by the `SHAREVISTRING` constant. The application creates the new message identifier by calling the **RegisterWindowMessage** function and passing this constant as the single parameter. After calling **RegisterWindowMessage**, the application is notified whenever a sharing violation occurs during a call to the **OpenFile** function. For more information about processing registered window messages, see "Using Find and Replace dialog boxes."

Displaying the Save As dialog box in your application

The Save As dialog box appears after you initialize the members of an **OPENFILENAME** structure and call the **GetSaveFileName** function.

Following is a Save As dialog box.



Before the call to **GetSaveFileName**, structure members contain such data as the name of the initial directory and a filter string. After the call, structure members contain such data as the name of the file to be saved and the number of characters in that filename.

The following example initializes an **OPENFILENAME** structure, calls **GetSaveFileName** function, and saves the file. The **OPENFILENAME** structure should be global or declared as a **static** variable.

```
OPENFILENAME ofn;
char szDirName[256];
char szFile[256], szFileTitle[256];
UINT i, cbString;
char chReplace; /* string separator for szFilter */
char szFilter[256];
HFILE hf;

/*
 * Retrieve the system directory name, and store it in
 * szDirName.
 */

GetSystemDirectory(szDirName, sizeof(szDirName));

if ((cbString = LoadString(hinst, IDS_FILTERSTRING,
    szFilter, sizeof(szFilter))) == 0) {
    ErrorHandler();
    return 0;
}

chReplace = szFilter[cbString - 1]; /* retrieve wildcard */

for (i = 0; szFilter[i] != '\0'; i++) {
    if (szFilter[i] == chReplace)
        szFilter[i] = '\0';
}
```

```

/* Set all structure members to zero. */

memset(&ofn, 0, sizeof(OPENFILENAME));

/* Initialize the OPENFILENAME members. */

szFile[0] = '\0';

ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.lpstrFilter = szFilter;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileTitle = szFileTitle;
ofn.nMaxFileTitle = sizeof(szFileTitle);
ofn.lpstrInitialDir = szDirName;
ofn.Flags = OFN_SHOWHELP | OFN_OVERWRITEPROMPT;

if(GetSaveFileName(&ofn)){
    .
    . /* Perform file operations. */
    .
}
else
    ErrorHandler();

```

The string referred to by the `IDS_FILTERSTRING` constant in the preceding example is defined in the resource-definition file. It is used in exactly the same way as the `IDS_FILTERSTRING` constant discussed in “Displaying the Open dialog box in your application.”

Monitoring list box controls in an Open or Save As dialog box

An application can monitor list box selections in order to process and display data in custom controls. For example, an application can use a custom control to display the total length, in bytes, of all of the files selected in the File Name box. One method the application can use to obtain this value is to recompute the total count of bytes each time the user selects a file or cancels the selection of a file. A faster method is for the application to use the `LBSELCHSTRING` message to identify a new selection and add the corresponding file length to the value that appears in the custom control. (Note that in this example, the custom control is a standard Windows control that you identify in a resource file template for one of the common dialog boxes.)

An application registers the selection-change message with the **RegisterWindowMessage** function. Once the application registers the message, it uses this function’s return value to identify

messages from the dialog box. The message is processed in the application-supplied hook function for the common dialog box. The *wParam* parameter of each message identifies the list box in which the selection occurred. The low-order word of the *lParam* parameter identifies the list box item. The high-order word of the *lParam* parameter is one of the following values:

Value	Meaning
CD_LBSELCHANGE	Specifies that the item identified by the low-order word of <i>lParam</i> was the item in a single-selection list box.
CD_LBSELSUB	Specifies that the item identified by the low-order word of <i>lParam</i> is no longer selected in a multiple-selection list box.
CD_LBSELADD	Specifies that the item identified by the low-order word of <i>lParam</i> was selected from a multiple-selection list box.
CD_LBSELNOITEMS	Specifies that no items exist in a multiple-selection list box.

For an example that registers a common dialog box message, see “Using Find and Replace dialog boxes.”

Monitoring filenames in an Open or Save As dialog box

Applications can alter the normal processing of an Open or Save As dialog box by monitoring which filename the user types and by performing other, unique operations. For example, one application could prevent the user from closing the dialog box if the selected filename is prohibited; another application could make it possible for the user to select multiple filenames.

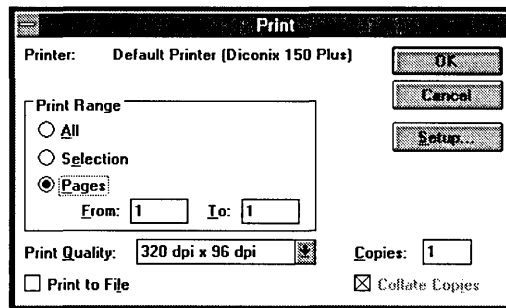
To monitor filenames, an application should register the **FILEOKSTRING** message. An application registers this message by calling the **RegisterWindowMessage** function and passing the message name as its single parameter. After the message is registered, the dialog box procedure in **COMMDLG.DLL** uses it to signal that the user has selected a filename and chosen the OK button and that the dialog box has checked the filename and is ready to return. The dialog box procedure signals these actions by sending the message to the application’s hook function. After receiving the message, the hook function should return a value to the dialog box procedure that called it. If the hook function did not process the message, it should return 0; if the hook function did process the message and the dialog box should close, the

hook function should return 0; if the hook function did process the message but the dialog box should not close, the hook function should return 1. (All other return values are reserved.)

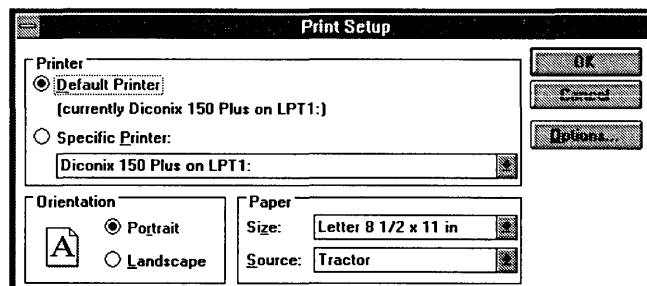
Using Print and Print Setup dialog boxes

A Print dialog box contains controls that let a user configure a printer for a particular print job. The user can make such selections as print quality, page range, and number of copies (if the printer supports multiple copies).

Following is a Print dialog box.



Choosing the Setup button in the Print dialog box displays the following Print Setup dialog box for a PostScript printer.



The Print Setup dialog box provides controls that make it possible for the user to reconfigure the selected printer.

Device drivers and the Print dialog box

The Print dialog box differs from other common dialog boxes in that part of its dialog box procedure resides in **COMMDDL.DLL** and part in a printer driver. A printer driver is a program that configures a printer, converts graphics device interface (GDI) commands to low-level printer commands, and stores commands for a particular print job in a printer's queue.

A printer driver exports a function called **ExtDeviceMode**, which displays a dialog box and its controls. In previous versions of Windows, an application called the **LoadLibrary** function to load a device driver and the **GetProcAddress** function to obtain the address of the **ExtDeviceMode** function. This is no longer necessary with the Windows common dialog box interface. Instead of calling **LoadLibrary** and **GetProcAddress**, a Windows application can call a single function, **PrintDlg**, to display the Print dialog box and begin a print job. The code for **PrintDlg** resides in **COMMDDL.DLL**. The dialog box that appears when an application calls **PrintDlg** differs slightly from the dialog box that appears when the application calls directly into the device driver. The functionality is very similar in spite of the different appearance.

Displaying a Print dialog box for the default printer

To display a Print dialog box for the default printer, an application must initialize a **PRINTDLG** structure and then call the **PrintDlg** function.

The members of the **PRINTDLG** structure can contain information about such items as the following:

- ▣ The printer device context
- ▣ Values that should appear in the dialog box controls
- ▣ The hook function and custom dialog box template to use for a customized version of the Print dialog box or Print Setup dialog box

An application can display a Print dialog box for the currently installed printer by performing the following steps:

1. Setting the **PD_RETURNDC** flag in the **Flags** member of the **PRINTDLG** structure. (This flag should only be set if the application requires a device-context handle.)

2. Initializing the **lStructSize**, **hDevMode**, and **hDevNames** members.
3. Calling the **PrintDlg** function and passing a pointer to the **PRINTDLG** structure just initialized.

Setting the **PD_RETURNDC** flag causes **PrintDlg** to display the Print dialog box and return a handle identifying a printer device context in the **hDC** member of the **PRINTDLG** structure. (The application passes the device-context handle as the first parameter to the GDI functions that render output on the printer.)

The following example initializes the members of the **PRINTDLG** structure and calls the **PrintDlg** function prior to printing output. This structure should be global or declared as a **static** variable.

```
PRINTDLGpd;

/* Set all structure members to zero. */

memset(&pd, 0, sizeof(PRINTDLG));

/* Initialize the necessary PRINTDLG structure members. */

pd.lStructSize = sizeof(PRINTDLG);
pd.hwndOwner = hwnd;
pd.Flags = PD_RETURNDC;

/* Print a test page if successful. */

if (PrintDlg(&pd) != 0) {
    Escape(pd.hDC, STARTDOC, 8, "Test-Doc", NULL);

    /* Print text and rectangle. */

    TextOut(pd.hDC, 50, 50, "Common Dialog Test Page", 23);
    Rectangle(pd.hDC, 50, 90, 625, 105);
    Escape(pd.hDC, NEWFRAME, 0, NULL, NULL);
    Escape(pd.hDC, ENDDOC, 0, NULL, NULL);
    DeleteDC(pd.hDC);
    if (pd.hDevMode != NULL)
        GlobalFree(pd.hDevMode);
    if (pd.hDevNames != NULL)
        GlobalFree(pd.hDevNames);
}
else
    ErrorHandler();
```

Using Find and Replace dialog boxes

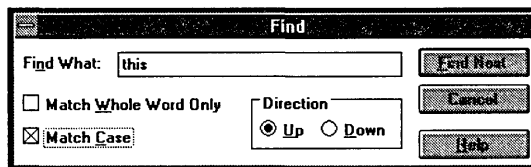
The Find dialog box and the Replace dialog box are similar in appearance. You can use the Find dialog box to add string-search capabilities to your application and use the Replace dialog box to add both string-search and string-substitution capabilities.

Displaying the Find dialog box

The Find dialog box contains controls that make it possible for a user to specify the following:

- The string that the application should find
- Whether the string specifies a complete word or part of a word
- Whether the application should match the case of the specified string
- The direction in which the application should search (preceding or following the current cursor location)
- Whether the application should resume the search, searching for the next occurrence of the string

Following is a Find dialog box.



To display the Find dialog box, you need to initialize a **FINDREPLACE** structure and call the **FindText** function. Members of the **FINDREPLACE** structure contain information about such items as the following:

- Which window owns the dialog box
- How the application should perform the search
- A character buffer that is to receive the string

To initialize the **FINDREPLACE** structure, you need to perform the following tasks:

1. Set the **lStructSize** member by using the **sizeof** operator.
2. Set the **hwndOwner** member by using the handle that identifies the owner window of the dialog box.
3. If you are customizing the Find dialog box, set the **hInstance** member to identify the instance of the module that contains your custom dialog box template.
4. Set the **Flags** member to indicate the selection state of the dialog box options. (For example, setting the **FR_NOUPDOWN** flag disables the Up and Down buttons, setting the **FR_NOWHOLEWORD** flag disables the Match Whole Word Only check box, and setting the **FR_NOMATCHCASE** flag disables the Match Case check box).
5. If you are supplying a custom dialog box template or hook function, set additional flags in the **Flags** member.
6. Set the **lpstrFindWhat** member to point to the buffer that will receive the string to be found.
7. Set the **wFindWhatLen** member to specify the size, in bytes, of the buffer to which **lpstrFindWhat** points.
8. Set the **lCustData** member with any custom data your application may need to access.
9. If your application customizes the Find dialog box, set the **lpfnHook** member to point to your hook function.
10. If your application uses a custom dialog box template, set the **lpTemplateName** member to point to the string that identifies the template.

The following example initializes the **FINDREPLACE** structure and then calls the **FindText** function. This structure should be global or declared as a **static** variable.

```
FINDREPLACE fr;

/* Set all structure fields to zero. */

memset(&fr, 0, sizeof(FINDREPLACE));

fr.lStructSize = sizeof(FINDREPLACE);
fr.hwndOwner = hwnd;
fr.lpstrFindWhat = szFindWhat;
fr.wFindWhatLen = sizeof(szFindWhat);

hDlg = FindText(&fr);

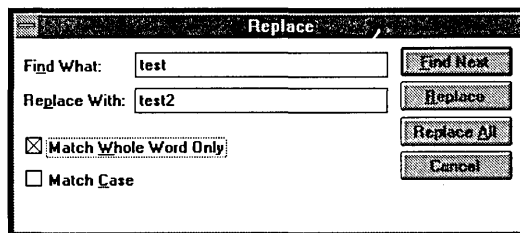
break;
```

Displaying the Replace dialog box

The Replace dialog box is similar to the Find dialog box. However, the Replace dialog box has no Direction box and has three additional controls that make it possible for the user to specify the following:

- The replacement string
- Whether the application should replace the occurrence of the string that is currently highlighted
- Whether the application should replace all occurrences of the string

Following is a Replace dialog box.



To display the Replace dialog box, you need to initialize a **FINDREPLACE** structure and call the **ReplaceText** function.

Processing dialog box messages for a Find or Replace dialog box

The Find and Replace dialog boxes differ from the other common dialogs in two respects: First, they are modeless; and second, their respective dialog box procedures send messages to the application that calls the **FindText** or **ReplaceText** function. These messages contain data specified by the user in the dialog box controls, such as the direction in which the application should search for a string, whether the application should match the case of the specified string, and whether the application should match the entire string.

To process messages from a Find or Replace dialog box, an application must register the dialog box's unique message, **FINDMSGSTRING**.

The application registers this message with the **RegisterWindowMessage** function. Once the application registers the message, it uses the function's return value to identify messages from the Find or Replace dialog box. The following example registers the message with the **RegisterWindowMessage** function:

```
UINT uFindReplaceMsg;

/* Register the FindReplace message. */

uFindReplaceMsg = RegisterWindowMessage(FINDMSGSTRING);
```

After the application registers this message, it can process messages for the Find or Replace dialog box by using the **RegisterWindowMessage** return value. The following example processes messages for the Find dialog box and then calls its own **SearchFile** function to locate the string of text. If the user is closing the dialog box (that is, if the **Flags** member of **FINDREPLACE** is **FR_DIALOGTERM**), the handle should be invalidated and the procedure should return zero.

```
LRESULT CALLBACK MainWndProc(HWND hwnd, UINT msg, WPARAM wParam,
LPARAM lParam)
{
    FINDREPLACE FAR* lpfr;

    if (msg == uFindReplaceMsg) {
        lpfr = (FINDREPLACE FAR*) lParam;
        SearchFile((BOOL) (lpfr->Flags & FR_DOWN),
                    (BOOL) (lpfr->Flags & FR_MATCHCASE));
        return 0;
    }
}
```

Customizing common dialog boxes

A custom common dialog box is a common dialog box that has been altered to suit a particular Windows application. The customization may be complex and include the hiding of original controls, the addition of new controls, or a change in the size of the original dialog box; or it may be simple, such as the alteration of a single existing control.

Developers who need to customize a common dialog box must provide a special hook function and, in most cases, a custom dialog box template. Customizations of this kind require a significant amount of additional code—displaying a customized common dialog box is not as simple as initializing the members of a structure and calling a single function.

Applications that subclass controls in any of the common dialog boxes must do so while processing the `WM_INITDIALOG` message in the application's hook function. This allows the application to receive the control-specific messages first, because it will have subclassed the control after the common dialog box has installed its subclassing procedures. (The previous hook function should be called for all messages that are not handled by the application's subclass function, as is standard for subclassing.)

An application cannot subclass a control by defining a local class to override a specific control type. The reason is that the data segment would not be correctly initialized when the class was called—the data segment would be the common dialog box's data segment, not the application's data segment.

Appropriate and inappropriate customizations

From the user's perspective, the chief benefit of the common dialog box is its consistent appearance and functionality from application to application. Therefore, it becomes important that a developer only customize a common dialog box when it is absolutely necessary for an application. Otherwise, the consistent appearance and simple coding interface are lost. Appropriate customizations leave intact as many of the original controls as possible. Increasing the size of the dialog box or adding new controls in available space that already appears in the dialog box would be an appropriate customization. Hiding original controls

or otherwise changing the intended functionality of the original controls would be an inappropriate customization.

Hook functions and custom dialog box templates

Each common dialog box uses the dialog box procedure and dialog box template provided for it in `COMMDDL.dll`. The dialog box procedure processes messages and notifications for the common dialog box and its controls. The dialog box template defines the appearance of the dialog box—its dimensions, its location, and the dimensions and locations of controls that appear within it.

In addition to the provided dialog box procedure and dialog box template, a custom dialog box requires a hook function that you provide and, usually, a custom version of the dialog box template.

Hook function The dialog box procedure provided in `COMMDDL.dll` for a common dialog box calls the application's hook function if the application sets the appropriate flag and pointer in the structure for that common dialog box. The structure for each common dialog box contains a **Flags** member that specifies whether the application supplies a hook function and contains an **lpfnHook** member that points to the hook function if one exists. If the application sets the **Flags** member to indicate that a hook function exists, it must also set the **lpfnHook** member. The following example sets the **Flags** and **lpfnHook** members of an **OPENFILENAME** structure to support an application's hook function:

```
#define STRICT

#include <windows.h> /* required for all Windows applications */
#include <commdlg.h>
#include <string.h>
#include "header.h" /* specific to this program */

OPENFILENAME ofn;

/* Get the system directory name, and store in szDirName. */
GetSystemDirectory((LPSTR) szDirName, 255);

/* Initialize the OPENFILENAME members. */

szFile[0] = '\0';
ofn.lStructSize = sizeof(OPENFILENAME);
ofn.hwndOwner = hwnd;
ofn.hInstance = hInst;
```

```

ofn.lpstrFilter = szFilter[0];
ofn.lpstrCustomFilter = NULL;
ofn.nMaxCustFilter = 0L;
ofn.nFilterIndex = 1L;
ofn.lpstrFile = szFile;
ofn.nMaxFile = sizeof(szFile);
ofn.lpstrFileName = szFileName;
ofn.nMaxFileName = sizeof(szFileName);
ofn.lpstrInitialDir = szDirName;
ofn.lpstrTitle = NULL;
ofn.Flags = OFN_ENABLEHOOK | OFN_ENABLETEMPLATE;
ofn.nFileOffset = 0;
ofn.nFileExtension = 0;
ofn.lpstrDefExt = NULL;
ofn.lpfnHook = MakeProcInstance((FARPROC) FileOpenHookProc, hInst);
ofn.lpTemplateName = "FileOpen";

```

In the previous example, the **MakeProcInstance** function is called to create a procedure-instance address for the hook function. This address is assigned to the **lpfnHook** member of the **OPENFILENAME** structure. If the hook function is part of a dynamic-link library (rather than an application), the procedure address is obtained by calling the **GetProcAddress** function (instead of **MakeProcInstance**).

The hook function processes any messages or notifications that the custom dialog box requires. With the exception of one message (WM_INITDIALOG), the hook function receives messages and notifications before the dialog box procedure provided in COMMDLG.DLL receives them. In the case of WM_INITDIALOG, the hook function receives the message after the dialog box procedure and should process it. When the hook function finishes processing a message, it returns a value that indicates whether the dialog box procedure provided in COMMDLG.DLL should also process the message. If the dialog box procedure should process the message, the return value is FALSE; if the dialog box procedure should ignore the message, the return value is TRUE.

To process the message from the OK button after the dialog box procedure processes it, an application must post a message to itself when the OK message is received. When the application receives the message it has posted, the common dialog box procedure will have finished processing messages for the dialog box. This technique is particularly useful when working with the Find and Replace dialog boxes, because the **Flags** member of the **FINDREPLACE** structure does not reflect changes to the dialog box until after the messages have been processed by COMMDLG.DLL.

The following example shows a hook function for a custom Open dialog box:

```
UINTCALLBACKFileOpenHookProc(HWNDhdlg, UINTmsg, WPARAM
    wParam, LPARAM lParam)
{
    switch(msg) {
        case WM_INITDIALOG:
            return TRUE;

        case WM_COMMAND:

            /* Use IsDlgButtonChecked to set lCustData. */

            if (wParam == IDOK) {

                /* Set backup flag. */

                ofn.lCustData =
                    (DWORD) IsDlgButtonChecked(hdlg, ID_CUSTCHX);
            }

            return FALSE; /* Allow standard processing. */

        /* Allow standard processing. */

        return FALSE;
    }
}
```

This hook function tests a custom check box when the user chooses the OK button. If the check box was selected, the hook function sets the **lCustData** member of the **OPENFILENAME** structure to 1; otherwise, it sets the **lCustData** member to 0.

A hook function should never call the **EndDialog** function. Instead, if a hook function contains code that abnormally terminates a common dialog box, this code should pass the **IDABORT** value to the dialog box procedure by using the **PostMessage** function as shown in the following example:

```
PostMessage(hDlg, WM_COMMAND, IDABORT, (LONG)FALSE);
```

When a hook function posts the **IDABORT** value, the common dialog box function returns the value contained in the low word of the *lParam* parameter. For example, if the hook function for **GetOpenFileName** called the **PostMessage** function with (LONG) 100 as the last parameter, **GetOpenFileName** would return 100.

A hook function must be exported in an application's module-definition (.DEF) file as shown in the following example:

```
NAME cd

EXETYPE WINDOWS

STUB    'WINSTUB.EXE'

CODE    PRELOAD MOVEABLE DISCARDABLE

DATA    PRELOAD MOVEABLE MULTIPLE

HEAPSIZE 1024

STACKSIZE8192

EXPORTS
    FILEOPENHOOKPROC @1
```

Customizing a dialog box template

The dialog box template provided in COMMMDLG.DLL for each common dialog box contains the data that the dialog box procedure uses to display that common dialog box. Most applications that customize a common dialog box also need to create a custom dialog box template to use instead of the dialog box template in COMMMDLG.DLL. (A custom dialog box template is not required for all custom dialog boxes. For instance, a template would not be necessary if an application changed a dialog box in a relatively minor way and only in an unusual situation.)

A developer should create a custom dialog box template by modifying the appropriate dialog box template in COMMMDLG.DLL. Following are the template filenames and the names of their corresponding common dialog boxes:

Template filename	Corresponding dialog box
COLOR.DLG	Color
FILEOPEN.DLG	Open (single selection)
FILEOPEN.DLG	Open (multiple selection)
FINDTEXT.DLG	Find
FINDTEXT.DLG	Replace
FONT.DLG	Font
PRNSETUP.DLG	Print
PRNSETUP.DLG	Print Setup

The following excerpt is from a custom dialog box template created for an Open dialog box:

```
CONTROL "&Backup File", ID_CUSTCHX, "button",
        BS_AUTOCHECKBOX | WS_CHILD | WS_TABSTOP | WS_GROUP,
        208, 86, 50, 12

END
```

This entry supports the addition of a new Backup File check box immediately below the existing Read Only check box.

The custom template should be added to the application's resource file.

Displaying the custom dialog box

After your application creates the hook function and the dialog box template, it should set the members of the structure for the common dialog box being customized and call the appropriate function to display the custom dialog box.

The following example calls the **GetOpenFileName** function and creates a backup file if the user selected the custom Backup File check box in the custom Open dialog box:

```
/* Open the file and create a backup. */

if(GetOpenFileName(&ofn)){

    hf = _lopen(ofn.lpstrFile, OF_READWRITE);

    /* Create the backup file. */

    if (ofn.lCustData) {

        /* Process files with extension. */

        if (ofn.nFileExtension){

            for (i=0; i<(int)ofn.nFileExtension; i++)
                szChar[i] = *ofn.lpstrFile++;

        }/*endif */

        /* Process files without extension. */

        else {

            i=0;

            while (*ofn.lpstrFile!='\0')
                szChar[i++] = *ofn.lpstrFile++;

        }

    }

}
```

```

        szChar[i]='.';
    }/*end else*/

    pszNewPAFN = lstrcat(szChar, "BAK");

    /* Create the backup file. */

    hfBackup = _lcreat(pszNewPAFN, 0);

    /* Copy contents of original file to the backup file. */

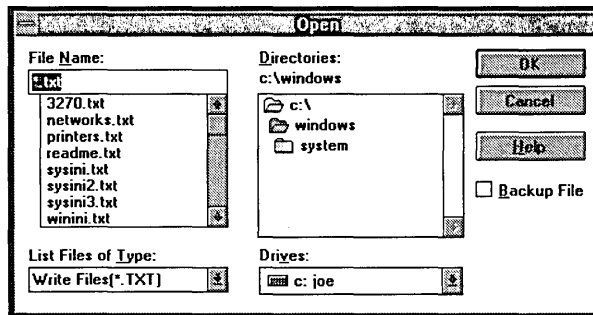
    while ((cBufLngth=_lread(hf, cBuf1, 256)) == 256)
        _lwrite(hfBackup, cBuf1, cBufLngth);
    _lwrite(hfBackup, cBuf1, cBufLngth);
    _lclose(hfBackup);
} /*endif GetOpenFileName*/

/* File operations begin here. */

} /* endif (GetOpenFileName) */

```

The following is the custom Open dialog box. The new Backup File check box appears in the lower-right corner.



Supporting help for the common dialog boxes

An application can display a Help button in any of the common dialog boxes by setting the appropriate flag in the **Flags** member of the structure for that common dialog box. Following are the structures for the common dialog boxes and the Help flag that corresponds to each structure:

Structure	Flag value
OPENFILENAME	OFN_SHOWHELP
CHOOSECOLOR	CC_SHOWHELP
FINDREPLACE	FR_SHOWHELP
CHOOSEFONT	CF_SHOWHELP
PRINTDLG	PD_SHOWHELP

If an application displays the Help button, it must process the user's request for Help. This can be done either in one of the application's window procedures or in a hook function.

If the application processes the request for Help in one of the application's window procedures, it must first create a new message identifier for the string defined by the **HELPMMSGSTRING** constant. The application creates the new message identifier by calling the **RegisterWindowMessage** function and passing this constant as the single parameter. (For more information about processing registered window messages, see "Using Find and Replace dialog boxes.") In addition to creating a new message identifier, the application must set the **hwndOwner** member of the appropriate structure for the common dialog box so that this member contains the handle of the dialog box's owner window. After the message identifier is created and the **hwndOwner** member is set, the dialog box procedure notifies the window procedure of the owner window whenever the user chooses the Help button.

The following example processes a user's request for Help in the window procedure of its owner window. The **if** statement should be in the **default:** section of the switch statement that processes messages.

```

MyHelpMsg = RegisterWindowMessage(HELPMMSGSTRING);
.
.
.
if (message == MyHelpMsg)
    WinHelp(hWnd, "appfile.hlp", HELP_CONTEXT, ID_MY_CONTEXT);

```

If the application processes the request for Help in a hook function, it should test for the following condition in the WM_COMMAND message:

```
wParam == pshHelp
```

When this condition is true, the hook function should call the **WinHelp** function as shown in the preceding example. (To process Help in a hook function, you must include the header file DLGS.H in the source file that contains the hook-function code.)

Error detection

Whenever a common dialog box function fails, an application can call the **CommDlgExtendedError** function to find out the cause of the failure. The **CommDlgExtendedError** function returns an error value that identifies the cause of the most recent error.

Six constants are defined in the CDERR.H header file that identify the ranges of error values for categories of errors returned by **CommDlgExtendedError**. Following are these constants in ascending order by value range:

Constant	Meaning
CDERR_GENERALCODES	General error codes for common dialog boxes. These errors are in the range 0x0000 through 0x0FFF.
PDERR_PRINTERCODES	Error codes for the Print common dialog box. These errors are in the range 0x1000 through 0x1FFF.
CFERR_CHOOSEFONTCODES	Error codes for the Font common dialog box. These errors are in the range 0x2000 through 0x2FFF.
FNERR_FILENAMECODES	Error codes for the Open and Save As common dialog boxes. These errors are in the range 0x3000 through 0x3FFF.

FRERR_FINDREPLACECODES	Error codes for the Find and Replace common dialog boxes. These errors are in the range 0x4000 through 0x4FFF.
CCERR_CHOOSSECOLORCODES	Error codes for the Color common dialog box. These errors are in the range 0x5000 through 0x5FFF.

A

ABC structure, 539
 Advise transaction, DDEML, 59
 Application (service) name, DDE servers, 40
 Asynchronous transaction, DDEML, 61

B

BN_HILITE message, 536
 BN_PAINT message, 536
 BN_UNHILITE message, 536

C

CB_ADDSTRING message, 503
 CB_DELETETESTRING message, 504
 CB_FINDSTRINGEXACT message, 505
 CB_GETDROPPEDCONTROLRECT message, 505
 CB_GETDROPPEDSTATE message, 506
 CB_GETTEXTENDEDUI message, 507
 CB_GETITEMHEIGHT message, 507
 CB_SETTEXTENDEDUI message, 508
 CB_SETITEMHEIGHT message, 509
 CBN_CLOSEUP message, 537
 CBN_SELENDNCANCEL message, 537
 CBN_SELENDOK message, 538
 CBT_CREATEWND structure, 540
 CBTACTIVATESTRUCT structure, 540
 ChooseColor function, 8, 9
 CHOOSECOLOR structure, 7, 8, 541
 ChooseFont function, 11
 CHOOSEFONT structure, 11, 544
 Class Name Object command,
 OLE applications, 109

CLASSENTRY structure, 551

Client applications

 DDE transactions, 39

 OLE client applications

 asynchronous operations, 99

 Class Name Object command, 109

 closing, 110

 closing documents, 99

 compound documents, opening, 97

 copying objects, 103

 creating objects, 105

 DDE, direct use of, 124

 deleting objects, 103

 described, 80

 displaying objects, 102

 opening and closing objects, 103

 Paste and Paste Link commands, 107

 printing objects, 102

 saving documents, 99

 starting, 96

 Undo command, 108

Client user interface, OLE applications, 88

ClientCallback function, OLECLEINTVTBL
 structure, 618

Clipboard

 formats, 82

 OLE conventions, 81

Close function, OLESERVERDOCVTBL
 structure, 631

Color dialog box

 described, 3

 displaying basic colors, 7

 displaying custom colors, 8

- HSL color model, 6
- RGB color model, 5
- COLOROKSTRING message, 719
- CommDlgExtendedError function, 35
- Common dialog box library
 - Color dialog box
 - described, 3
 - displaying basic colors, 7
 - displaying custom colors, 8
 - HSL color model, 6
 - RGB color model, 5
 - COMMDDL.DLL library, 1
 - common dialog boxes, described, 1
 - customizing common dialog boxes
 - described, 27
 - dialog box template, 31
 - displaying custom dialog boxes, 32
 - hook function, 28
 - error detection, 35
 - Find dialog box, 23, 26
 - Font dialog box, 11
 - Help button in common dialog boxes, 34
 - Open dialog box
 - displaying, 13
 - monitoring filenames, 19
 - monitoring list box controls, 18
 - Print dialog box, 20, 21
 - Print Setup dialog box, 20
 - Replace dialog box, 25, 26
 - Save As dialog box
 - displaying, 16
 - monitoring filenames, 19
 - monitoring list box controls, 18
- Compound document, OLE applications
 - described, 72
 - illustrated, 72
 - opening, 97
- COMSTAT structure, 552
- CONVCONTEXT structure, 553
- CONVINFO structure, 53, 554
- Copy command
 - OLE client applications, 103
 - OLE server applications, 92, 115
- CPLINFO structure, 557

- Create function, OLESERVERVTBL structure, 638
- CreateFromTemplate function,
 - OLESERVERVTBL structure, 639
- CTLINFO structure, 558
- CTLSTYLE structure, 559
- CTLTYPE structure, 561
- Cut command
 - OLE client applications, 103
 - OLE server applications, 92, 115

D

- Data handle
 - dynamic data exchange, 54
- Data types, defined, 493
- DdeAbandonTransaction function, 62
- DdeAccessData function
 - command strings, 60
 - global memory objects, 56
- DDEACK structure, 562
- DdeAddData function, 57
- DDEADVISE structure, 563
- DdeCallback function, 44
- DdeClientTransaction function
 - advise transaction, 59
 - execute transaction, 60
 - poke transaction, 58
 - request transaction, 57
 - synchronous and asynchronous transactions, 61
- DdeConnect function, 49
- DdeConnectList function, 52, 53
- DdeCreateDataHandle function, 54
- DdeCreateStringHandle function, 45
- DDEDATA structure, 564
- DdeDisconnect function, 52, 54
- DdeDisconnectList function, 54
- DdeEnableCallback function, 62
- DdeFreeDataHandle function, 57
- DdeFreeStringHandle function, 46
- DdeGetData function, 56
- DdeInitialize function
 - initializing DDEML, 42
 - monitoring DDE applications, 66

- DdeKeepStringHandle function, 46
- DdeNameService function, 47, 48
- DDEPOKE structure, 565
- DdePostAdvise function, 59
- DdeQueryConvInfo function, 52, 53, 62
- DdeQueryNextServer function, 53
- DdeQueryString function, 45
- DdeReconnect function, 52
- DdeSetUserHandle function, 62
- DdeUnaccessData function, 56
- DdeUninitialize function, 43
- DEBUGHOOKINFO structure, 566
- DECLARE_HANDLE macro, 695
- DECLARE_HANDLE32 macro, 695
- DefLoadFromStream function, 123
- DEVNAMES structure, 567
- DllCreateFromClip function, 122
- DllLoadFromStream function, 123
- DOCINFO structure, 568
- DoVerb function, OLEOBJECTVTBL structure, 625
- DRIVERINFOSTRUCT structure, 569
- DRVCONFIGINFO structure, 569
- Dynamic data exchange (DDE)
 - described, 37
 - OLE libraries
 - client applications, 124
 - conversations, 128
 - execute strings, 131, 132
 - server applications, 127
 - standard item names, 129
 - System topic, items for, 128
 - using for standard DDE operations, 77
- Dynamic Data Exchange Management Library (DDEML)
 - callback function, 44
 - client and server interaction, 39
 - conversations
 - multiple conversations, 52
 - single conversations, 49
 - suspending, 62
 - terminating, 43
 - data management, 54

- described, 37
- error detection, 66
- initializing, 42
- item names, 40
- monitoring applications, 66
- vs. OLE, 76
- OLE, using with DDEML, 79
- service names
 - described, 40
 - registering, 47
 - service-name filter, 48
- string management, 45
- System topic, 40
- topic names, 40
- transaction management
 - advise transaction, 59
 - asynchronous transactions, 61
 - controlling transactions, 62
 - execute transaction, 60
 - poke transaction, 58
 - request transaction, 57
 - synchronous transactions, 61
 - transaction classes, 63
 - transaction summary, 64
- transaction, defined, 39

E

- Edit function, OLESERVERVTBL structure, 640
- EM_GETFIRSTVISIBLELINE message, 510
- EM_GETPASSWORDCHAR message, 510
- EM_GETWORDBREAKPROC message, 511
- EM_SETREADONLY message, 511
- EM_SETWORDBREAKPROC message, 512
- Embedded object
 - defined, 74
- EnumClipboardFormats function, 108
- Error detection
 - common dialog boxes, 35
 - DDEML functions, 66
- EVENTMSG structure, 570
- Execute function
 - OLESERVERDOCVTBL structure, 636
 - OLESERVERVTBL structure, 642

Execute strings, OLE
 international execute commands, 131
 required commands, 132
 syntax for standard commands, 131
Execute transaction, DDEML, 60
Exit function, OLESERVERVTBL structure, 641
ExtDeviceMode function, 21

F

FIELDOFFSET macro, 696
FILEOKSTRING message, 720
Find dialog box
 displaying, 23
 processing messages, 26
FINDMSGSTRING message, 26, 721
FINDREPLACE structure, 23, 25, 571
FindText function, 23
FIXED structure, 575
FMS_GETDRIVEINFO structure, 576
FMS_GETFILESEL structure, 577
FMS_LOAD structure, 578
Font dialog box, 11
Functions
 DdeCallback function, 44
 OLE functions
 asynchronous operations, 101
 document management, 98
 object creation, 105
 object handlers, 120
 server applications, 111

G

Get function, OLESTREAMVTBL structure, 644
GetBValue macro, 696
GetData function, 79
GetData function, OLEOBJECTVTBL structure, 626
GetGValue macro, 697
GetObject function, 79
GetObject function, OLESERVERDOCVTBL structure, 633
GetOpenFileName function, 13
728

GetRValue macro, 697
GetSaveFileName function, 16
GetWinFlags function
 initializing DDEML, 42
GLOBAENTRY structure, 579
GLOBALINFO structure, 582
GLYPHMETRICS structure, 583

H

HARDWAREHOOKSTRUCT structure, 584
Help button in common dialog boxes, 34
HELPMSGSTRING message, 721
HELPWININFO structure, 585
Hook function, common dialog boxes, 28
HSL color model, 6
HSZPAIR structure, 52, 585

I

Insert Object command, OLE applications, 106
Item name, DDE servers, 40

J

JUST_VALUE_STRUCT structure, 703

K

KERNINGPAIR structure, 586

L

LB_FINDSTRINGEXACT message, 513
LB_GETCARETINDEX message, 514
LB_SETCARETINDEX message, 514
LBN_SELCANCEL message, 538
LBSELCHSTRING message, 722
Linked object
 defined, 73
LoadString function, 16
LOCALENTRY structure, 587
LOCALINFO structure, 590
LOGFONT structure
 Font dialog box, 13
 TrueType fonts, server applications, 116

M

- MAKELP macro, 697
- MAKELPARAM macro, 698
- MAKELRESULT macro, 698
- MAT2 structure, 591
- MEMMANINFO structure, 592
- Metafile
 - OLE server applications, 116
- METAHEADER structure, 593
- METARECORD structure, 594
- MINMAXINFO structure, 595
- MODULEENTRY structure, 596
- MONCBSTRUCT structure, 597
- MONCONVSTRUCT structure, 598
- MONERRSTRUCT structure, 599
- MONHSZSTRUCT structure, 600
- Monitoring applications, 66
- MONLINKSTRUCT structure, 602
- MONMSGSTRUCT structure, 603
- MOUSEHOOKSTRUCT structure, 604
- MOUSETRAILS printer escape, 701

N

- Native clipboard format, 82
- NCCALCSIZE_PARAMS structure, 605
- NEWCPINFO structure, 606
- NEWTEXTMETRIC structure, 607
- NFYLOADSEG structure, 612
- NFYLOGERROR structure, 613
- NFYLOGPARAMERROR structure, 614
- NFYRIP structure, 615
- NFYSTARTDLL structure, 616

O

- Object handler, OLE libraries
 - creating objects in, 122
 - described, 80
 - implementing, 119
- Object linking and embedding (OLE)
 - benefits of OLE, 75
 - client applications
 - asynchronous operations, 99
 - Class Name Object command, 109

- closing, 110
- closing documents, 99
- copying objects, 103
- creating objects, 105
- deleting objects, 103
- described, 95
- displaying objects, 102
- document management, 98
- opening and closing objects, 103
- Paste and Paste Link commands, 107
- printing objects, 102
- saving documents, 99
- starting, 96
- Undo command, 108
- compound documents
 - described, 72
 - illustrated, 72
 - opening, 97
- data transfer
 - client applications, 80
 - client user interface, 88
 - clipboard conventions, 81
 - commands, new and changed, 88
 - communication between libraries, 81
 - object handlers, 80
 - packages, 91
 - registration database, 85
 - server applications, 80
 - server user interface, 92
 - version control for servers, 87
- DDEML
 - vs. OLE, 76
 - using with OLE, 79
- dynamic data exchange
 - client applications, 124
 - conversations, 128
 - DDE operations, using OLE for, 77
 - execute strings, 131, 132
 - server applications, 127
 - standard item names, 129
 - System topic, items for, 128
- embedded object, defined, 74
- formats for storing objects, 93
- linked object, defined, 73

- object handlers
 - creating objects in, 122
 - implementing, 119
- OLECLI.DLL library, 80
- OLESVR.DLL library, 80
- packages, 74
- server applications
 - closing, 117
 - Cut and Copy commands, 115
 - functions, 111
 - opening documents or objects, 115
 - Save and Save As commands, 116
 - starting, 112
 - Update command, 116
- verbs, 74
- ObjectLink clipboard format, 82
- ObjectLong function, OLEOBJECTVTBL structure, 627
- OFFSETOF macro, 699
- OleActivate function
 - Class Name Object command, implementing, 109
 - opening objects, 103
- OleBlockServer function
 - asynchronous operations, 100
 - queued client-library requests, 114
- OLECLIENT structure, 617
 - object handlers, 121
 - opening compound documents, 98
 - starting client applications, 96
- OLECLIENTVTBL structure, 96, 617
- OleClone function
 - copying objects to the clipboard, 105
 - restoring updated objects, 108
- OleClose function, 103
- OleCopyFromLink function, 106
- OleCopyToClipboard function, 89, 103
- OleCreate function, 106
- OleCreateFromClip function
 - client applications, 107
 - object handlers, 122
- OleCreateFromFile function, 78
- OleCreateFromTemplate function, 106
- OleCreateLinkFromClip function, 107

- OleDelete function, 103
- OleDraw function, 102
- OleEnumFormats function, 103
- OleGetData function, 110
- OleGetLinkUpdateOptions command, 109
- OleLoadFromStream function, 97
- OLEOBJECT structure, 620
 - client applications, creating objects, 106
 - object handlers, 120
 - server applications
 - opening objects, 115
 - starting, 113
- OleObjectConvert function, 110
- OLEOBJECTVTBL structure, 120, 621
- OleQueryBounds function, 102
- OleQueryCreateFromClip function, 107
- OleQueryLinkFromClip function, 107
- OleQueryOpen function, 103
- OleQueryReleaseError function
 - closing client applications, 110
 - creating objects, 106
- OleQueryReleaseMethod function, 110
- OleQueryReleaseStatus function
 - activating objects, 103
 - asynchronous operations, 101
 - closing client applications, 110
- OleQuerySize function, 99
- OleQueryType function, 107
- OleReconnect function, 103
- OleRegisterClientDoc function, 97, 105
- OleRegisterServer function
 - DDE operations, 79
 - server applications, starting, 112
- OleRegisterServerDoc function
 - DDE operations, 79
 - opening documents or objects, 115
 - starting server applications, 112
- OleRelease function
 - closing client applications, 110
 - closing documents, 99
 - closing objects, 103
- OleRenameServerDoc function, 116
- OleRequestData function, 87
- OleRevertClientDoc function, 99

- OleRevokeClientDoc function, 99, 105
- OleRevokeObject function, 118
- OleRevokeServerDoc function, 118
- OleSavedClientDoc function, 99, 105
- OleSavedServerDoc function, 116
- OleSaveToStream function, 99, 105
- OLESERVER structure, 629
 - object handlers, 121
 - starting server applications, 112
- OLESERVERDOC structure, 630
 - object handlers, 121
 - opening documents, 115
- OLESERVERDOCVTBL structure, 630
 - DDE operations, 79
 - starting server applications, 112
- OLESERVERVTBL structure, 636
 - closing server applications, 117
 - opening documents or objects, 115
 - starting server applications, 112
 - updating documents, 117
- OleSetBounds function, 102
- OleSetData function
 - changing links, 110
 - DDE operations, using OLE for, 78
 - registering data formats, 87
- OleSetHostNames function, 103
- OleSetLinkUpdateOptions command, 109
- OleSetTargetDevice function, 102
- OLESTREAM structure, 643
 - object handlers, 121
 - opening compound documents, 98
 - starting client applications, 96
- OLESTREAMVTBL structure, 97, 98, 643
- OLETARGETDEVICE structure, 645
- OleUnblockServer function, 114
- OleUpdate function
 - displaying objects, 102
 - updating links, 109
- Open dialog box
 - displaying, 13
 - filenames, monitoring, 19
 - list box controls, monitoring, 18
- Open function, OLESERVERVTBL structure, 637

- OPENFILENAME structure, 646
 - Open dialog box, 13
 - Save As dialog box, 16
- OUTLINETEXTMETRIC structure, 655
- OwnerLink clipboard format, 82

P

- Package, OLE applications, 74, 91
- PANOSE structure, 659
- Paste command, OLE applications, 107
- Paste Link command, OLE applications, 107
- Paste Special command, OLE applications, 108
- POINTFX structure, 664
- Poke transaction, DDEML, 58
- POSTSCRIPT_DATA printer escape
 - See* PASSTHROUGH printer escape
- POSTSCRIPT_IGNORE printer escape, 702
- Print dialog box, 20, 21
- Print Setup dialog box, 20
- PrintDlg function, 21
- PRINTDLG structure, 21, 665
- Printer
 - default printer, 21
- Put function, OLESTREAMVTBL structure, 644

R

- RASTERIZER_STATUS structure, 673
- RegisterClipboardFormat function, 87, 113
- RegisterWindowMessage function
 - Color dialog box, 10
 - filenames, monitoring, 19
 - Find and Replace dialog boxes, 26
 - Help button in common dialog boxes, 34
 - list box controls, monitoring, 18
 - Open dialog box, 16
- Registration database
 - OLE applications, 85
- Release function
 - OLEOBJECTVTBL structure, 624
 - OLESERVERDOCVTBL structure, 634
 - OLESERVERVTBL structure, 641
- Replace dialog box

- displaying, 25
- processing messages, 26
- ReplaceText function, 25
- Request transaction, DDEML, 57
- RGB color model, 5

S

- Save As command, OLE server applications, 116
- Save As dialog box
 - displaying, 16
 - filenames, monitoring, 19
 - list box controls, monitoring, 18
- Save command, OLE server applications, 116
- Save function, OLESERVERDOCVTBL structure, 631
- SearchFile function, 26
- SEGINFO structure, 673
- SELECTOROF macro, 699
- Server applications
 - DDE transactions, 39
 - OLE servers
 - closing, 117
 - Cut and Copy commands, 115
 - DDE, direct use of, 127
 - DDE, required commands, 132
 - described, 80
 - functions, 111
 - opening documents or objects, 115
 - Save and Save As commands, 116
 - server user interface, 92
 - starting, 112
 - Update command, 116
 - version control, 87
- Service name, DDE servers, 40
- SETALLJUSTVALUES printer escape, 703
- SetClipboardData function
 - client applications, 103
 - server applications, 115
- SetColorScheme function
 - OLEOBJECTVTBL structure, 628
 - OLESERVERDOCVTBL structure, 635
- SetData function, OLEOBJECTVTBL structure, 626

- SetDocDimensions function, OLESERVERDOCVTBL structure, 633
- SetHostNames function OLESERVERDOCVTBL structure, 632
- SETRGBSTRING message, 723
- SetTargetDevice function, OLEOBJECTVTBL structure, 627
- SHAREVISTRING message, 723
- Shell library
 - OLE applications, 85
- Show function, OLEOBJECTVTBL structure, 624
- SIZE structure, 675
- STACKTRACEENTRY structure, 676
- STM_GETICON message, 515
- STM_SETICON message, 515
- String handle, DDE, 45
- Synchronous transaction, DDEML, 61
- SYSHEAPINFO structure, 677
- System topic, DDEML, 40
- Systems topic, DDE-based OLE, 128

T

- TASKENTRY structure, 678
- Template, common dialog box, 31
- TIMERINFO structure, 679
- Topic name, DDE servers, 40
- Transaction, DDE
 - defined, 39
- TrueType fonts, server applications, 116
- TTPOLYCURVE structure, 680
- TTPOLYGONHEADER structure, 681

U

- Undo command, OLE applications, 108
- Update command, OLE server applications, 116

V

- Verb, object linking and embedding, 74
- Version control for OLE servers, 87
- VS_FIXEDFILEINFO structure, 682

W

WINDEBUGINFO structure, 686
WINDOWPLACEMENT structure, 690
WINDOWPOS structure, 692
Windows data types, defined, 493
WinHelp function, 34
WM_CHOOSEFONT_GETLOGFONT message,
 13, 516
WM_COMMNOTIFY message, 517
WM_DDE_ACK message, 517
WM_DDE_ADVISE message, 79, 520
WM_DDE_DATA message, 521
WM_DDE_EXECUTE message, 523
WM_DDE_INITIATE message, 524
WM_DDE_POKE message, 78, 526
WM_DDE_REQUEST message, 527
WM_DDE_TERMINATE message, 528
WM_DDE_UNADVISE message, 529
WM_DROPFILES message, 107, 530
WM_INITDIALOG message, 27
WM_PALETTEISCHANGING message, 530
WM_POWER message, 531
WM_QUEUESYNC message, 532
WM_SYSTEMERROR message, 532
WM_USER message, 533
WM_WINDOWPOSCHANGED message, 534
WM_WINDOWPOSCHANGING message, 535

X

XTYP_ADVDATA transaction, 705
XTYP_ADVREQ transaction, 706
XTYP_ADVSTART transaction, 59, 707
XTYP_ADVSTOP transaction, 60, 708
XTYP_CONNECT transaction, 49, 708
XTYP_CONNECT_CONFIRM transaction, 49, 52, 709
XTYP_DISCONNECT transaction, 52, 54, 710
XTYP_ERROR transaction, 711
XTYP_EXECUTE transaction, 60, 711
XTYP_MONITOR transaction, 68, 712

XTYP_POKE transaction, 58, 713
XTYP_REGISTER transaction, 47, 714
XTYP_REQUEST transaction, 45, 57, 715
XTYP_UNREGISTER transaction, 715
XTYP_WILDCONNECT transaction, 52, 716
XTYP_XACT_COMPLETE transaction, 61, 717

WINDOWS API

VOLUME III

B O R L A N D

Corporate Headquarters: 1800 Green Hills Road, P.O. Box 660001, Scotts Valley, CA 95067-0001, (408) 438-5300. Offices in: Australia, Belgium, Canada, Denmark, France, Germany, Hong Kong, Italy, Japan, Korea, Malaysia, Netherlands, New Zealand, Singapore, Spain, Sweden, Taiwan and United Kingdom ■ Part #14MN-API03-31 ■ BOR 3985